[ucore/lab8_result/user/ls.c] ++++++++++++++++++++++++++++++

```c
#include <ulib.h>
#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <file.h>
#include <stat.h>
#include <dirent.h>
#include <unistd.h>

#define printf(...)                     fprintf(1, __VA_ARGS__)
#define BUFSIZE                         4096

static char
getmode(uint32_t st_mode) {
    char mode = '?';
    if (S_ISREG(st_mode)) mode = '-';
    if (S_ISDIR(st_mode)) mode = 'd';
    if (S_ISLNK(st_mode)) mode = 'l';
    if (S_ISCHR(st_mode)) mode = 'c';
    if (S_ISBLK(st_mode)) mode = 'b';
    return mode;
}

static int
getstat(const char *name, struct stat *stat) {
    int fd, ret;
    if ((fd = open(name, O_RDONLY)) < 0) {
        return fd;
    }
    ret = fstat(fd, stat);
    close(fd);
    return ret;
}

void
lsstat(struct stat *stat, const char *filename) {
    printf("   [%c]", getmode(stat->st_mode));
    printf(" %3d(h)", stat->st_nlinks);
    printf(" %8d(b)", stat->st_blocks);
    printf(" %8d(s)", stat->st_size);
    printf("   %s\n", filename);
}

int
lsdir(const char *path) {
    struct stat __stat, *stat = &__stat;
    int ret;
    DIR *dirp = opendir(".");

    if (dirp == NULL) {
        return -1;
    }
    struct dirent *direntp;
    while ((direntp = readdir(dirp)) != NULL) {
        if ((ret = getstat(direntp->name, stat)) != 0) {
            goto failed;
        }
        lsstat(stat, direntp->name);
    }
    printf("lsdir: step 4\n");
    closedir(dirp);
    return 0;
failed:
    closedir(dirp);
```

```
    return ret;
}

int
ls(const char *path) {
    struct stat __stat, *stat = &__stat;
    int ret, type;
    if ((ret = getstat(path, stat)) != 0) {
        return ret;
    }

    static const char *filetype[] = {
        " [  file   ]",
        " [directory]",
        " [ symlink ]",
        " [character]",
        " [  block  ]",
        " [  ?????  ]",
    };
    switch (getmode(stat->st_mode)) {
    case '0': type = 0; break;
    case 'd': type = 1; break;
    case 'l': type = 2; break;
    case 'c': type = 3; break;
    case 'b': type = 4; break;
    default:  type = 5; break;
    }

    printf(" @ is %s", filetype[type]);
    printf(" %d(hlinks)", stat->st_nlinks);
    printf(" %d(blocks)", stat->st_blocks);
    printf(" %d(bytes) : @'%s'\n", stat->st_size, path);
    if (S_ISDIR(stat->st_mode)) {
        return lsdir(path);
    }
    return 0;
}

int
main(int argc, char **argv) {
    if (argc == 1) {
        return ls(".");
    }
    else {
        int i, ret;
        for (i = 1; i < argc; i ++) {
            if ((ret = ls(argv[i])) != 0) {
                return ret;
            }
        }
    }
    return 0;
}
```

[ucore/lab8_result/user/faultread.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    cprintf("I read %8x from 0.\n", *(unsigned int *)0);
    panic("FAIL: T.T\n");
}
```

[ucore/lab8_result/user/faultreadkernel.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    cprintf("I read %08x from 0xfac00000!\n", *(unsigned *)0xfac00000);
    panic("FAIL: T.T\n");
}
```

[ucore/lab8_result/user/sleepkill.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    int pid;
    if ((pid = fork()) == 0) {
        sleep(~0);
        exit(0xdead);
    }
    assert(pid > 0);

    sleep(100);
    assert(kill(pid) == 0);
    cprintf("sleepkill pass.\n");
    return 0;
}
```

[ucore/lab8_result/user/hello.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    cprintf("Hello world!!.\n");
    cprintf("I am process %d.\n", getpid());
    cprintf("hello pass.\n");
    return 0;
}
```

[ucore/lab8_result/user/badarg.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    int pid, exit_code;
    if ((pid = fork()) == 0) {
        cprintf("fork ok.\n");
        int i;
        for (i = 0; i < 10; i ++) {
            yield();
        }
        exit(0xbeaf);
    }
    assert(pid > 0);
    assert(waitpid(-1, NULL) != 0);
    assert(waitpid(pid, (void *)0xC0000000) != 0);
    assert(waitpid(pid, &exit_code) == 0 && exit_code == 0xbeaf);
    cprintf("badarg pass.\n");
    return 0;
}
```

[ucore/lab8_result/user/exit.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
```

```c
#include <ulib.h>

int magic = -0x10384;

int
main(void) {
    int pid, code;
    cprintf("I am the parent. Forking the child...\n");
    if ((pid = fork()) == 0) {
        cprintf("I am the child.\n");
        yield();
        yield();
        yield();
        yield();
        yield();
        yield();
        yield();
        exit(magic);
    }
    else {
        cprintf("I am parent, fork a child pid %d\n",pid);
    }
    assert(pid > 0);
    cprintf("I am the parent, waiting now..\n");

    assert(waitpid(pid, &code) == 0 && code == magic);
    assert(waitpid(pid, &code) != 0 && wait() != 0);
    cprintf("waitpid %d ok.\n", pid);

    cprintf("exit pass.\n");
    return 0;
}
```

[ucore/lab8_result/user/softint.c] ++++++++++++++++++++++++++++++

```c
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    asm volatile("int $14");
    panic("FAIL: T.T\n");
}
```

[ucore/lab8_result/user/priority.c] ++++++++++++++++++++++++++++++

```c
#include <ulib.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TOTAL 5
/* to get enough accuracy, MAX_TIME (the running time of each process) should >1000 mseconds.
*/
#define MAX_TIME  1000
#define SLEEP_TIME 400
unsigned int acc[TOTAL];
int status[TOTAL];
int pids[TOTAL];

static void
spin_delay(void)
{
     int i;
     volatile int j;
     for (i = 0; i != 200; ++ i)
     {
```

```
            j = !j;
        }
}

int
main(void) {
        int i,time;
        cprintf("priority process will sleep %d ticks\n",SLEEP_TIME);
        sleep(SLEEP_TIME);
        memset(pids, 0, sizeof(pids));
        lab6_set_priority(TOTAL + 1);

        for (i = 0; i < TOTAL; i ++) {
            acc[i]=0;
            if ((pids[i] = fork()) == 0) {
                lab6_set_priority(i + 1);
                acc[i] = 0;
                while (1) {
                    spin_delay();
                    ++ acc[i];
                    if(acc[i]%4000==0) {
                        if((time=gettime_msec())>SLEEP_TIME+MAX_TIME) {
                            cprintf("child pid %d, acc %d, time %d\n",getpid(),acc[i],time);
                            exit(acc[i]);
                        }
                    }
                }

            }
            if (pids[i] < 0) {
                goto failed;
            }
        }

        cprintf("main: fork ok,now need to wait pids.\n");

        for (i = 0; i < TOTAL; i ++) {
            status[i]=0;
            waitpid(pids[i],&status[i]);
            cprintf("main: pid %d, acc %d, time %d\n",pids[i],status[i],gettime_msec());
        }
        cprintf("main: wait pids over\n");
        cprintf("stride sched correct result:");
        for (i = 0; i < TOTAL; i ++)
        {
            cprintf(" %d", (status[i] * 2 / status[0] + 1) / 2);
        }
        cprintf("\n");

        return 0;

failed:
        for (i = 0; i < TOTAL; i ++) {
            if (pids[i] > 0) {
                kill(pids[i]);
            }
        }
        panic("FAIL: T.T\n");
}

[ucore/lab8_result/user/badsegment.c] ++++++++++++++++++++++++++++++
#include <stdio.h>
#include <ulib.h>

/* try to load the kernel's TSS selector into the DS register */
```

```
int
main(void) {
    asm volatile("movw $0x28,%ax; movw %ax,%ds");
    panic("FAIL: T.T\n");
}

[ucore/lab8_result/user/forktest.c] ++++++++++++++++++++++++++++++
#include <ulib.h>
#include <stdio.h>

const int max_child = 32;

int
main(void) {
    int n, pid;
    for (n = 0; n < max_child; n ++) {
        if ((pid = fork()) == 0) {
            cprintf("I am child %d\n", n);
            exit(0);
        }
        assert(pid > 0);
    }

    if (n > max_child) {
        panic("fork claimed to work %d times!\n", n);
    }

    for (; n > 0; n --) {
        if (wait() != 0) {
            panic("wait stopped early\n");
        }
    }

    if (wait() == 0) {
        panic("wait got too many\n");
    }

    cprintf("forktest pass.\n");
    return 0;
}

[ucore/lab8_result/user/waitkill.c] ++++++++++++++++++++++++++++++
#include <ulib.h>
#include <stdio.h>

void
do_yield(void) {
    yield();
    yield();
    yield();
    yield();
    yield();
    yield();
}

int parent, pid1, pid2;

void
loop(void) {
    cprintf("child 1.\n");
    while (1);
}

void
```

```
work(void) {
    cprintf("child 2.\n");
    do_yield();
    if (kill(parent) == 0) {
        cprintf("kill parent ok.\n");
        do_yield();
        if (kill(pid1) == 0) {
            cprintf("kill child1 ok.\n");
            exit(0);
        }
    }
    exit(-1);
}

int
main(void) {
    parent = getpid();
    if ((pid1 = fork()) == 0) {
        loop();
    }

    assert(pid1 > 0);

    if ((pid2 = fork()) == 0) {
        work();
    }
    if (pid2 > 0) {
        cprintf("wait child 1.\n");
        waitpid(pid1, NULL);
        panic("waitpid %d returns\n", pid1);
    }
    else {
        kill(pid1);
    }
    panic("FAIL: T.T\n");
}
```

[ucore/lab8_result/user/yield.c] ++++++++++++++++++++++++++++++

```
#include <ulib.h>
#include <stdio.h>

int
main(void) {
    int i;
    cprintf("Hello, I am process %d.\n", getpid());
    for (i = 0; i < 5; i ++) {
        yield();
        cprintf("Back in process %d, iteration %d.\n", getpid(), i);
    }
    cprintf("All done in process %d.\n", getpid());
    cprintf("yield pass.\n");
    return 0;
}
```

[ucore/lab8_result/user/matrix.c] ++++++++++++++++++++++++++++++

```
#include <ulib.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MATSIZE     10

static int mata[MATSIZE][MATSIZE];
static int matb[MATSIZE][MATSIZE];
static int matc[MATSIZE][MATSIZE];
```

```
void
work(unsigned int times) {
    int i, j, k, size = MATSIZE;
    for (i = 0; i < size; i ++) {
        for (j = 0; j < size; j ++) {
            mata[i][j] = matb[i][j] = 1;
        }
    }

    yield();

    cprintf("pid %d is running (%d times)!.\n", getpid(), times);

    while (times -- > 0) {
        for (i = 0; i < size; i ++) {
            for (j = 0; j < size; j ++) {
                matc[i][j] = 0;
                for (k = 0; k < size; k ++) {
                    matc[i][j] += mata[i][k] * matb[k][j];
                }
            }
        }
        for (i = 0; i < size; i ++) {
            for (j = 0; j < size; j ++) {
                mata[i][j] = matb[i][j] = matc[i][j];
            }
        }
    }
    cprintf("pid %d done!.\n", getpid());
    exit(0);
}

const int total = 21;

int
main(void) {
    int pids[total];
    memset(pids, 0, sizeof(pids));

    int i;
    for (i = 0; i < total; i ++) {
        if ((pids[i] = fork()) == 0) {
            srand(i * i);
            int times = (((unsigned int)rand()) % total);
            times = (times * times + 10) * 100;
            work(times);
        }
        if (pids[i] < 0) {
            goto failed;
        }
    }

    cprintf("fork ok.\n");

    for (i = 0; i < total; i ++) {
        if (wait() != 0) {
            cprintf("wait failed.\n");
            goto failed;
        }
    }

    cprintf("matrix pass.\n");
    return 0;
```

```
failed:
    for (i = 0; i < total; i ++) {
        if (pids[i] > 0) {
            kill(pids[i]);
        }
    }
    panic("FAIL: T.T\n");
}

[ucore/lab8_result/user/pgdir.c] ++++++++++++++++++++++++++++++
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    cprintf("I am %d, print pgdir.\n", getpid());
    print_pgdir();
    cprintf("pgdir pass.\n");
    return 0;
}

[ucore/lab8_result/user/sh.c] ++++++++++++++++++++++++++++++
#include <ulib.h>
#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <file.h>
#include <error.h>
#include <unistd.h>

#define printf(...)                     fprintf(1, __VA_ARGS__)
#define putc(c)                         printf("%c", c)

#define BUFSIZE                         4096
#define WHITESPACE                      " \t\r\n"
#define SYMBOLS                         "<|>&;"

char shcwd[BUFSIZE];

int
gettoken(char **p1, char **p2) {
    char *s;
    if ((s = *p1) == NULL) {
        return 0;
    }
    while (strchr(WHITESPACE, *s) != NULL) {
        *s ++ = '\0';
    }
    if (*s == '\0') {
        return 0;
    }

    *p2 = s;
    int token = 'w';
    if (strchr(SYMBOLS, *s) != NULL) {
        token = *s, *s ++ = '\0';
    }
    else {
        bool flag = 0;
        while (*s != '\0' && (flag || strchr(WHITESPACE SYMBOLS, *s) == NULL)) {
            if (*s == '"') {
                *s = ' ', flag = !flag;
            }
            s ++;
        }
```

```
    }
    *p1 = (*s != '\0' ? s : NULL);
    return token;
}

char *
readline(const char *prompt) {
    static char buffer[BUFSIZE];
    if (prompt != NULL) {
        printf("%s", prompt);
    }
    int ret, i = 0;
    while (1) {
        char c;
        if ((ret = read(0, &c, sizeof(char))) < 0) {
            return NULL;
        }
        else if (ret == 0) {
            if (i > 0) {
                buffer[i] = '\0';
                break;
            }
            return NULL;
        }

        if (c == 3) {
            return NULL;
        }
        else if (c >= ' ' && i < BUFSIZE - 1) {
            putc(c);
            buffer[i ++] = c;
        }
        else if (c == '\b' && i > 0) {
            putc(c);
            i --;
        }
        else if (c == '\n' || c == '\r') {
            putc(c);
            buffer[i] = '\0';
            break;
        }
    }
    return buffer;
}

void
usage(void) {
    printf("usage: sh [command-file]\n");
}

int
reopen(int fd2, const char *filename, uint32_t open_flags) {
    int ret, fd1;
    close(fd2);
    if ((ret = open(filename, open_flags)) >= 0 && ret != fd2) {
        close(fd2);
        fd1 = ret, ret = dup2(fd1, fd2);
        close(fd1);
    }
    return ret < 0 ? ret : 0;
}

int
testfile(const char *name) {
    int ret;
```

```
    if ((ret = open(name, O_RDONLY)) < 0) {
        return ret;
    }
    close(ret);
    return 0;
}

int
runcmd(char *cmd) {
    static char argv0[BUFSIZE];
    const char *argv[EXEC_MAX_ARG_NUM + 1];
    char *t;
    int argc, token, ret, p[2];
again:
    argc = 0;
    while (1) {
        switch (token = gettoken(&cmd, &t)) {
        case 'w':
            if (argc == EXEC_MAX_ARG_NUM) {
                printf("sh error: too many arguments\n");
                return -1;
            }
            argv[argc ++] = t;
            break;
        case '<':
            if (gettoken(&cmd, &t) != 'w') {
                printf("sh error: syntax error: < not followed by word\n");
                return -1;
            }
            if ((ret = reopen(0, t, O_RDONLY)) != 0) {
                return ret;
            }
            break;
        case '>':
            if (gettoken(&cmd, &t) != 'w') {
                printf("sh error: syntax error: > not followed by word\n");
                return -1;
            }
            if ((ret = reopen(1, t, O_RDWR | O_TRUNC | O_CREAT)) != 0) {
                return ret;
            }
            break;
        case '|':
          //  if ((ret = pipe(p)) != 0) {
          //      return ret;
          //  }
            if ((ret = fork()) == 0) {
                close(0);
                if ((ret = dup2(p[0], 0)) < 0) {
                    return ret;
                }
                close(p[0]), close(p[1]);
                goto again;
            }
            else {
                if (ret < 0) {
                    return ret;
                }
                close(1);
                if ((ret = dup2(p[1], 1)) < 0) {
                    return ret;
                }
                close(p[0]), close(p[1]);
                goto runit;
            }
```

```
            break;
        case 0:
            goto runit;
        case ';':
            if ((ret = fork()) == 0) {
                goto runit;
            }
            else {
                if (ret < 0) {
                    return ret;
                }
                waitpid(ret, NULL);
                goto again;
            }
            break;
        default:
            printf("sh error: bad return %d from gettoken\n", token);
            return -1;
        }
    }

runit:
    if (argc == 0) {
        return 0;
    }
    else if (strcmp(argv[0], "cd") == 0) {
        if (argc != 2) {
            return -1;
        }
        strcpy(shcwd, argv[1]);
        return 0;
    }
    if ((ret = testfile(argv[0])) != 0) {
        if (ret != -E_NOENT) {
            return ret;
        }
        snprintf(argv0, sizeof(argv0), "/%s", argv[0]);
        argv[0] = argv0;
    }
    argv[argc] = NULL;
    return __exec(NULL, argv);
}

int
main(int argc, char **argv) {
    printf("user sh is running!!!");
    int ret, interactive = 1;
    if (argc == 2) {
        if ((ret = reopen(0, argv[1], O_RDONLY)) != 0) {
            return ret;
        }
        interactive = 0;
    }
    else if (argc > 2) {
        usage();
        return -1;
    }
    //shcwd = malloc(BUFSIZE);
    assert(shcwd != NULL);

    char *buffer;
    while ((buffer = readline((interactive) ? "$ " : NULL)) != NULL) {
        shcwd[0] = '\0';
        int pid;
        if ((pid = fork()) == 0) {
```

```
            ret = runcmd(buffer);
            exit(ret);
        }
        assert(pid >= 0);
        if (waitpid(pid, &ret) == 0) {
            if (ret == 0 && shcwd[0] != '\0') {
                ret = 0;
            }
            if (ret != 0) {
                printf("error: %d - %e\n", ret, ret);
            }
        }
    }
    return 0;
}

[ucore/lab8_result/user/sfs_filetest1.c] ++++++++++++++++++++++++++++++
#include <ulib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stat.h>
#include <file.h>
#include <dir.h>
#include <unistd.h>

#define printf(...)                 fprintf(1, __VA_ARGS__)

static int safe_open(const char *path, int open_flags)
{
        int fd = open(path, open_flags);
        printf("fd is %d\n",fd);
        assert(fd >= 0);
        return fd;
}

static struct stat *safe_fstat(int fd)
{
        static struct stat __stat, *stat = &__stat;
        int ret = fstat(fd, stat);
        assert(ret == 0);
        return stat;
}


static void safe_read(int fd, void *data, size_t len)
{
        int ret = read(fd, data, len);
        assert(ret == len);
}


int main(void)
{
        int fd1 = safe_open("sfs_filetest1", O_RDONLY);
        struct stat *stat = safe_fstat(fd1);
        assert(stat->st_size >= 0 && stat->st_blocks >= 0);
        printf("sfs_filetest1 pass.\n");
        return 0;
}
[ucore/lab8_result/user/testbss.c] ++++++++++++++++++++++++++++++
#include <stdio.h>
#include <ulib.h>

#define ARRAYSIZE (1024*1024)
```

```
uint32_t bigarray[ARRAYSIZE];

int
main(void) {
    cprintf("Making sure bss works right...\n");
    int i;
    for (i = 0; i < ARRAYSIZE; i ++) {
        if (bigarray[i] != 0) {
            panic("bigarray[%d] isn't cleared!\n", i);
        }
    }
    for (i = 0; i < ARRAYSIZE; i ++) {
        bigarray[i] = i;
    }
    for (i = 0; i < ARRAYSIZE; i ++) {
        if (bigarray[i] != i) {
            panic("bigarray[%d] didn't hold its value!\n", i);
        }
    }

    cprintf("Yes, good.  Now doing a wild write off the end...\n");
    cprintf("testbss may pass.\n");

    bigarray[ARRAYSIZE + 1024] = 0;
    asm volatile ("int $0x14");
    panic("FAIL: T.T\n");
}

[ucore/lab8_result/user/libs/syscall.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <unistd.h>
#include <stdarg.h>
#include <syscall.h>
#include <stat.h>
#include <dirent.h>


#define MAX_ARGS            5

static inline int
syscall(int num, ...) {
    va_list ap;
    va_start(ap, num);
    uint32_t a[MAX_ARGS];
    int i, ret;
    for (i = 0; i < MAX_ARGS; i ++) {
        a[i] = va_arg(ap, uint32_t);
    }
    va_end(ap);

    asm volatile (
        "int %1;"
        : "=a" (ret)
        : "i" (T_SYSCALL),
          "a" (num),
          "d" (a[0]),
          "c" (a[1]),
          "b" (a[2]),
          "D" (a[3]),
          "S" (a[4])
        : "cc", "memory");
    return ret;
}
```

```
int
sys_exit(int error_code) {
    return syscall(SYS_exit, error_code);
}

int
sys_fork(void) {
    return syscall(SYS_fork);
}

int
sys_wait(int pid, int *store) {
    return syscall(SYS_wait, pid, store);
}

int
sys_yield(void) {
    return syscall(SYS_yield);
}

int
sys_kill(int pid) {
    return syscall(SYS_kill, pid);
}

int
sys_getpid(void) {
    return syscall(SYS_getpid);
}

int
sys_putc(int c) {
    return syscall(SYS_putc, c);
}

int
sys_pgdir(void) {
    return syscall(SYS_pgdir);
}

void
sys_lab6_set_priority(uint32_t priority)
{
    syscall(SYS_lab6_set_priority, priority);
}

int
sys_sleep(unsigned int time) {
    return syscall(SYS_sleep, time);
}

size_t
sys_gettime(void) {
    return syscall(SYS_gettime);
}

int
sys_exec(const char *name, int argc, const char **argv) {
    return syscall(SYS_exec, name, argc, argv);
}

int
sys_open(const char *path, uint32_t open_flags) {
    return syscall(SYS_open, path, open_flags);
}
```

```
int
sys_close(int fd) {
    return syscall(SYS_close, fd);
}

int
sys_read(int fd, void *base, size_t len) {
    return syscall(SYS_read, fd, base, len);
}

int
sys_write(int fd, void *base, size_t len) {
    return syscall(SYS_write, fd, base, len);
}

int
sys_seek(int fd, off_t pos, int whence) {
    return syscall(SYS_seek, fd, pos, whence);
}

int
sys_fstat(int fd, struct stat *stat) {
    return syscall(SYS_fstat, fd, stat);
}

int
sys_fsync(int fd) {
    return syscall(SYS_fsync, fd);
}

int
sys_getcwd(char *buffer, size_t len) {
    return syscall(SYS_getcwd, buffer, len);
}

int
sys_getdirentry(int fd, struct dirent *dirent) {
    return syscall(SYS_getdirentry, fd, dirent);
}

int
sys_dup(int fd1, int fd2) {
    return syscall(SYS_dup, fd1, fd2);
}
[ucore/lab8_result/user/libs/file.h] ++++++++++++++++++++++++++++++
#ifndef __USER_LIBS_FILE_H__
#define __USER_LIBS_FILE_H__

#include <defs.h>

struct stat;

int open(const char *path, uint32_t open_flags);
int close(int fd);
int read(int fd, void *base, size_t len);
int write(int fd, void *base, size_t len);
int seek(int fd, off_t pos, int whence);
int fstat(int fd, struct stat *stat);
int fsync(int fd);
int dup(int fd);
int dup2(int fd1, int fd2);
int pipe(int *fd_store);
int mkfifo(const char *name, uint32_t open_flags);
```

```
void print_stat(const char *name, int fd, struct stat *stat);

#endif /* !__USER_LIBS_FILE_H__ */

[ucore/lab8_result/user/libs/panic.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdarg.h>
#include <stdio.h>
#include <ulib.h>
#include <error.h>

void
__panic(const char *file, int line, const char *fmt, ...) {
    // print the 'message'
    va_list ap;
    va_start(ap, fmt);
    cprintf("user panic at %s:%d:\n    ", file, line);
    vcprintf(fmt, ap);
    cprintf("\n");
    va_end(ap);
    exit(-E_PANIC);
}

void
__warn(const char *file, int line, const char *fmt, ...) {
    va_list ap;
    va_start(ap, fmt);
    cprintf("user warning at %s:%d:\n    ", file, line);
    vcprintf(fmt, ap);
    cprintf("\n");
    va_end(ap);
}

[ucore/lab8_result/user/libs/umain.c] ++++++++++++++++++++++++++++++
#include <ulib.h>
#include <unistd.h>
#include <file.h>
#include <stat.h>

int main(int argc, char *argv[]);

static int
initfd(int fd2, const char *path, uint32_t open_flags) {
    int fd1, ret;
    if ((fd1 = open(path, open_flags)) < 0) {
        return fd1;
    }
    if (fd1 != fd2) {
        close(fd2);
        ret = dup2(fd1, fd2);
        close(fd1);
    }
    return ret;
}

void
umain(int argc, char *argv[]) {
    int fd;
    if ((fd = initfd(0, "stdin:", O_RDONLY)) < 0) {
        warn("open <stdin> failed: %e.\n", fd);
    }
    if ((fd = initfd(1, "stdout:", O_WRONLY)) < 0) {
        warn("open <stdout> failed: %e.\n", fd);
    }
    int ret = main(argc, argv);
```

```
    exit(ret);
}
```

[ucore/lab8_result/user/libs/dir.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <string.h>
#include <syscall.h>
#include <stat.h>
#include <dirent.h>
#include <file.h>
#include <dir.h>
#include <error.h>
#include <unistd.h>

DIR dir, *dirp=&dir;
DIR *
opendir(const char *path) {

    if ((dirp->fd = open(path, O_RDONLY)) < 0) {
        goto failed;
    }
    struct stat __stat, *stat = &__stat;
    if (fstat(dirp->fd, stat) != 0 || !S_ISDIR(stat->st_mode)) {
        goto failed;
    }
    dirp->dirent.offset = 0;
    return dirp;

failed:
    return NULL;
}

struct dirent *
readdir(DIR *dirp) {
    if (sys_getdirentry(dirp->fd, &(dirp->dirent)) == 0) {
        return &(dirp->dirent);
    }
    return NULL;
}

void
closedir(DIR *dirp) {
    close(dirp->fd);
}

int
getcwd(char *buffer, size_t len) {
    return sys_getcwd(buffer, len);
}
```

[ucore/lab8_result/user/libs/ulib.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <syscall.h>
#include <stdio.h>
#include <ulib.h>
#include <stat.h>
#include <string.h>
#include <lock.h>

static lock_t fork_lock = INIT_LOCK;

void
lock_fork(void) {
    lock(&fork_lock);
}
```

```
void
unlock_fork(void) {
    unlock(&fork_lock);
}

void
exit(int error_code) {
    sys_exit(error_code);
    cprintf("BUG: exit failed.\n");
    while (1);
}

int
fork(void) {
    return sys_fork();
}

int
wait(void) {
    return sys_wait(0, NULL);
}

int
waitpid(int pid, int *store) {
    return sys_wait(pid, store);
}

void
yield(void) {
    sys_yield();
}

int
kill(int pid) {
    return sys_kill(pid);
}

int
getpid(void) {
    return sys_getpid();
}

//print_pgdir - print the PDT&PT
void
print_pgdir(void) {
    sys_pgdir();
}

void
lab6_set_priority(uint32_t priority)
{
    sys_lab6_set_priority(priority);
}

int
sleep(unsigned int time) {
    return sys_sleep(time);
}

unsigned int
gettime_msec(void) {
    return (unsigned int)sys_gettime();
}
```

```
int
__exec(const char *name, const char **argv) {
    int argc = 0;
    while (argv[argc] != NULL) {
        argc ++;
    }
    return sys_exec(name, argc, argv);
}
[ucore/lab8_result/user/libs/stdio.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <syscall.h>
#include <file.h>
#include <ulib.h>
#include <unistd.h>

/* *
 * cputch - writes a single character @c to stdout, and it will
 * increace the value of counter pointed by @cnt.
 * */
static void
cputch(int c, int *cnt) {
    sys_putc(c);
    (*cnt) ++;
}

/* *
 * vcprintf - format a string and writes it to stdout
 *
 * The return value is the number of characters which would be
 * written to stdout.
 *
 * Call this function if you are already dealing with a va_list.
 * Or you probably want cprintf() instead.
 * */
int
vcprintf(const char *fmt, va_list ap) {
    int cnt = 0;
    vprintfmt((void*)cputch, NO_FD, &cnt, fmt, ap);
    return cnt;
}

/* *
 * cprintf - formats a string and writes it to stdout
 *
 * The return value is the number of characters which would be
 * written to stdout.
 * */
int
cprintf(const char *fmt, ...) {
    va_list ap;

    va_start(ap, fmt);
    int cnt = vcprintf(fmt, ap);
    va_end(ap);

    return cnt;
}

/* *
 * cputs- writes the string pointed by @str to stdout and
 * appends a newline character.
 * */
int
cputs(const char *str) {
```

```
    int cnt = 0;
    char c;
    while ((c = *str ++) != '\0') {
        cputch(c, &cnt);
    }
    cputch('\n', &cnt);
    return cnt;
}


static void
fputch(char c, int *cnt, int fd) {
    write(fd, &c, sizeof(char));
    (*cnt) ++;
}

int
vfprintf(int fd, const char *fmt, va_list ap) {
    int cnt = 0;
    vprintfmt((void*)fputch, fd, &cnt, fmt, ap);
    return cnt;
}

int
fprintf(int fd, const char *fmt, ...) {
    va_list ap;

    va_start(ap, fmt);
    int cnt = vfprintf(fd, fmt, ap);
    va_end(ap);

    return cnt;
}
[ucore/lab8_result/user/libs/initcode.S] ++++++++++++++++++++++++++++++++
.text
.globl _start
_start:
    # set ebp for backtrace
    movl $0x0, %ebp

    # load argc and argv
    movl (%esp), %ebx
    lea 0x4(%esp), %ecx


    # move down the esp register
    # since it may cause page fault in backtrace
    subl $0x20, %esp

    # save argc and argv on stack
    pushl %ecx
    pushl %ebx

    # call user-program function
    call umain
1:  jmp 1b


[ucore/lab8_result/user/libs/syscall.h] ++++++++++++++++++++++++++++++++
#ifndef __USER_LIBS_SYSCALL_H__
#define __USER_LIBS_SYSCALL_H__

int sys_exit(int error_code);
int sys_fork(void);
int sys_wait(int pid, int *store);
```

```
int sys_exec(const char *name, int argc, const char **argv);
int sys_yield(void);
int sys_kill(int pid);
int sys_getpid(void);
int sys_putc(int c);
int sys_pgdir(void);
int sys_sleep(unsigned int time);
size_t sys_gettime(void);

struct stat;
struct dirent;

int sys_open(const char *path, uint32_t open_flags);
int sys_close(int fd);
int sys_read(int fd, void *base, size_t len);
int sys_write(int fd, void *base, size_t len);
int sys_seek(int fd, off_t pos, int whence);
int sys_fstat(int fd, struct stat *stat);
int sys_fsync(int fd);
int sys_getcwd(char *buffer, size_t len);
int sys_getdirentry(int fd, struct dirent *dirent);
int sys_dup(int fd1, int fd2);
void sys_lab6_set_priority(uint32_t priority); //only for lab6


#endif /* !__USER_LIBS_SYSCALL_H__ */

[ucore/lab8_result/user/libs/ulib.h] ++++++++++++++++++++++++++++++
#ifndef __USER_LIBS_ULIB_H__
#define __USER_LIBS_ULIB_H__

#include <defs.h>

void __warn(const char *file, int line, const char *fmt, ...);
void __noreturn __panic(const char *file, int line, const char *fmt, ...);

#define warn(...)                                               \
    __warn(__FILE__, __LINE__, __VA_ARGS__)

#define panic(...)                                              \
    __panic(__FILE__, __LINE__, __VA_ARGS__)

#define assert(x)                                               \
    do {                                                        \
        if (!(x)) {                                             \
            panic("assertion failed: %s", #x);                  \
        }                                                       \
    } while (0)

// static_assert(x) will generate a compile-time error if 'x' is false.
#define static_assert(x)                                        \
    switch (x) { case 0: case (x): ; }

int fprintf(int fd, const char *fmt, ...);

void __noreturn exit(int error_code);
int fork(void);
int wait(void);
int waitpid(int pid, int *store);
void yield(void);
int kill(int pid);
int getpid(void);
void print_pgdir(void);
int sleep(unsigned int time);
unsigned int gettime_msec(void);
```

```
int __exec(const char *name, const char **argv);

#define __exec0(name, path, ...)                \
({ const char *argv[] = {path, ##__VA_ARGS__, NULL}; __exec(name, argv); })

#define exec(path, ...)                         __exec0(NULL, path, ##__VA_ARGS__)
#define nexec(name, path, ...)                  __exec0(name, path, ##__VA_ARGS__)

void lab6_set_priority(uint32_t priority); //only for lab6

#endif /* !__USER_LIBS_ULIB_H__ */

[ucore/lab8_result/user/libs/dir.h] ++++++++++++++++++++++++++++++
#ifndef __USER_LIBS_DIR_H__
#define __USER_LIBS_DIR_H__

#include <defs.h>
#include <dirent.h>

typedef struct {
    int fd;
    struct dirent dirent;
} DIR;

DIR *opendir(const char *path);
struct dirent *readdir(DIR *dirp);
void closedir(DIR *dirp);
int chdir(const char *path);
int getcwd(char *buffer, size_t len);

#endif /* !__USER_LIBS_DIR_H__ */

[ucore/lab8_result/user/libs/file.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <syscall.h>
#include <stdio.h>
#include <stat.h>
#include <error.h>
#include <unistd.h>

int
open(const char *path, uint32_t open_flags) {
    return sys_open(path, open_flags);
}

int
close(int fd) {
    return sys_close(fd);
}

int
read(int fd, void *base, size_t len) {
    return sys_read(fd, base, len);
}

int
write(int fd, void *base, size_t len) {
    return sys_write(fd, base, len);
}

int
seek(int fd, off_t pos, int whence) {
    return sys_seek(fd, pos, whence);
}
```

```
int
fstat(int fd, struct stat *stat) {
    return sys_fstat(fd, stat);
}

int
fsync(int fd) {
    return sys_fsync(fd);
}

int
dup2(int fd1, int fd2) {
    return sys_dup(fd1, fd2);
}

static char
transmode(struct stat *stat) {
    uint32_t mode = stat->st_mode;
    if (S_ISREG(mode)) return 'r';
    if (S_ISDIR(mode)) return 'd';
    if (S_ISLNK(mode)) return 'l';
    if (S_ISCHR(mode)) return 'c';
    if (S_ISBLK(mode)) return 'b';
    return '-';
}

void
print_stat(const char *name, int fd, struct stat *stat) {
    cprintf("[%03d] %s\n", fd, name);
    cprintf("    mode    : %c\n", transmode(stat));
    cprintf("    links   : %lu\n", stat->st_nlinks);
    cprintf("    blocks  : %lu\n", stat->st_blocks);
    cprintf("    size    : %lu\n", stat->st_size);
}
```

[ucore/lab8_result/user/libs/lock.h] ++++++++++++++++++++++++++++++

```
#ifndef __USER_LIBS_LOCK_H__
#define __USER_LIBS_LOCK_H__

#include <defs.h>
#include <atomic.h>
#include <ulib.h>

#define INIT_LOCK           {0}

typedef volatile bool lock_t;

static inline void
lock_init(lock_t *l) {
    *l = 0;
}

static inline bool
try_lock(lock_t *l) {
    return test_and_set_bit(0, l);
}

static inline void
lock(lock_t *l) {
    if (try_lock(l)) {
        int step = 0;
        do {
            yield();
            if (++ step == 100) {
```

```
                step = 0;
                sleep(10);
            }
        } while (try_lock(l));
    }
}

static inline void
unlock(lock_t *l) {
    test_and_clear_bit(0, l);
}

#endif /* !__USER_LIBS_LOCK_H__ */
```

[ucore/lab8_result/user/spin.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <ulib.h>

int
main(void) {
    int pid, ret, i ,j;
    cprintf("I am the parent. Forking the child...\n");
    pid = fork();
    if (pid== 0) {
        cprintf("I am the child. spinning ...\n");
        while (1);
    }else if (pid<0) {
        panic("fork child error\n");
    }
    cprintf("I am the parent. Running the child...\n");

    yield();
    yield();
    yield();

    cprintf("I am the parent.  Killing the child...\n");

    assert((ret = kill(pid)) == 0);
    cprintf("kill returns %d\n", ret);

    assert((ret = waitpid(pid, NULL)) == 0);
    cprintf("wait returns %d\n", ret);

    cprintf("spin may pass.\n");
    return 0;
}
```

[ucore/lab8_result/user/sleep.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <ulib.h>

void
sleepy(int pid) {
    int i, time = 100;
    for (i = 0; i < 10; i ++) {
        sleep(time);
        cprintf("sleep %d x %d slices.\n", i + 1, time);
    }
    exit(0);
}

int
main(void) {
    unsigned int time = gettime_msec();
    int pid1, exit_code;
```

```
    if ((pid1 = fork()) == 0) {
        sleepy(pid1);
    }

    assert(waitpid(pid1, &exit_code) == 0 && exit_code == 0);
    cprintf("use %04d msecs.\n", gettime_msec() - time);
    cprintf("sleep pass.\n");
    return 0;
}

[ucore/lab8_result/user/forktree.c] ++++++++++++++++++++++++++++++
#include <ulib.h>
#include <stdio.h>
#include <string.h>

#define DEPTH 4
#define SLEEP_TIME 400
void forktree(const char *cur);

void
forkchild(const char *cur, char branch) {
    char nxt[DEPTH + 1];

    if (strlen(cur) >= DEPTH)
        return;

    snprintf(nxt, DEPTH + 1, "%s%c", cur, branch);
    if (fork() == 0) {
        forktree(nxt);
        yield();
        exit(0);
    }
}

void
forktree(const char *cur) {
    cprintf("%04x: I am '%s'\n", getpid(), cur);

    forkchild(cur, '0');
    forkchild(cur, '1');
}

int
main(void) {
    cprintf("forktree process will sleep %d ticks\n",SLEEP_TIME);
    sleep(SLEEP_TIME);
    forktree("");
    return 0;
}

[ucore/lab8_result/user/divzero.c] ++++++++++++++++++++++++++++++
#include <stdio.h>
#include <ulib.h>

int zero;

int
main(void) {
    cprintf("value is %d.\n", 1 / zero);
    panic("FAIL: T.T\n");
}

[ucore/lab8_result/tools/sign.c] ++++++++++++++++++++++++++++++
#include <stdio.h>
```

```c
#include <errno.h>
#include <string.h>
#include <sys/stat.h>

int
main(int argc, char *argv[]) {
    struct stat st;
    if (argc != 3) {
        fprintf(stderr, "Usage: <input filename> <output filename>\n");
        return -1;
    }
    if (stat(argv[1], &st) != 0) {
        fprintf(stderr, "Error opening file '%s': %s\n", argv[1], strerror(errno));
        return -1;
    }
    printf("'%s' size: %lld bytes\n", argv[1], (long long)st.st_size);
    if (st.st_size > 510) {
        fprintf(stderr, "%lld >> 510!!\n", (long long)st.st_size);
        return -1;
    }
    char buf[512];
    memset(buf, 0, sizeof(buf));
    FILE *ifp = fopen(argv[1], "rb");
    int size = fread(buf, 1, st.st_size, ifp);
    if (size != st.st_size) {
        fprintf(stderr, "read '%s' error, size is %d.\n", argv[1], size);
        return -1;
    }
    fclose(ifp);
    buf[510] = 0x55;
    buf[511] = 0xAA;
    FILE *ofp = fopen(argv[2], "wb+");
    size = fwrite(buf, 1, 512, ofp);
    if (size != 512) {
        fprintf(stderr, "write '%s' error, size is %d.\n", argv[2], size);
        return -1;
    }
    fclose(ofp);
    printf("build 512 bytes boot sector: '%s' success!\n", argv[2]);
    return 0;
}

[ucore/lab8_result/tools/mksfs.c] ++++++++++++++++++++++++++++++
/* prefer to compile mksfs on 64-bit linux systems.

Use a compiler-specific macro.

For example:

#if defined(__i386__)
// IA-32
#elif defined(__x86_64__)
// AMD64
#else
# error Unsupported architecture
#endif

*/

#define _GNU_SOURCE
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <limits.h>
```

```c
#include <dirent.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <errno.h>
#include <assert.h>

typedef int bool;

#define __error(msg, quit, ...)                                                 \
    do {                                                                        \
        fprintf(stderr, #msg ": function %s - line %d: ", __FUNCTION__, __LINE__);      \
        if (errno != 0) {                                                       \
            fprintf(stderr, "[error] %s: ", strerror(errno));                   \
        }                                                                       \
        fprintf(stderr, "\n\t"), fprintf(stderr, __VA_ARGS__);                  \
        errno = 0;                                                              \
        if (quit) {                                                             \
            exit(-1);                                                           \
        }                                                                       \
    } while (0)

#define warn(...)           __error(warn, 0, __VA_ARGS__)
#define bug(...)            __error(bug, 1, __VA_ARGS__)

/*
static_assert(cond, msg) is defined in /usr/include/assert.h
#define static_assert(x)                                                        \
    switch (x) {case 0: case (x): ; }
*/

/* 2^31 + 2^29 - 2^25 + 2^22 - 2^19 - 2^16 + 1 */
#define GOLDEN_RATIO_PRIME_32       0x9e370001UL

#define HASH_SHIFT                              10
#define HASH_LIST_SIZE                          (1 << HASH_SHIFT)

static inline uint32_t
__hash32(uint32_t val, unsigned int bits) {
    uint32_t hash = val * GOLDEN_RATIO_PRIME_32;
    return (hash >> (32 - bits));
}

static uint32_t
hash32(uint32_t val) {
    return __hash32(val, HASH_SHIFT);
}

static uint32_t
hash64(uint64_t val) {
    return __hash32((uint32_t)val, HASH_SHIFT);
}

void *
safe_malloc(size_t size) {
    void *ret;
    if ((ret = malloc(size)) == NULL) {
        bug("malloc %lu bytes failed.\n", (long unsigned)size);
    }
    return ret;
}

char *
safe_strdup(const char *str) {
```

```
    char *ret;
    if ((ret = strdup(str)) == NULL) {
        bug("strdup failed: %s\n", str);
    }
    return ret;
}

struct stat *
safe_stat(const char *filename) {
    static struct stat __stat;
    if (stat(filename, &__stat) != 0) {
        bug("stat %s failed.\n", filename);
    }
    return &__stat;
}

struct stat *
safe_fstat(int fd) {
    static struct stat __stat;
    if (fstat(fd, &__stat) != 0) {
        bug("fstat %d failed.\n", fd);
    }
    return &__stat;
}

struct stat *
safe_lstat(const char *name) {
    static struct stat __stat;
    if (lstat(name, &__stat) != 0) {
        bug("lstat '%s' failed.\n", name);
    }
    return &__stat;
}

void
safe_fchdir(int fd) {
    if (fchdir(fd) != 0) {
        bug("fchdir failed %d.\n", fd);
    }
}

#define SFS_MAGIC                           0x2f8dbe2a
#define SFS_NDIRECT                         12
#define SFS_BLKSIZE                         4096                                // 4K
#define SFS_MAX_NBLKS                       (1024UL * 512)                      // 4K
* 512K
#define SFS_MAX_INFO_LEN                    31
#define SFS_MAX_FNAME_LEN                   255
#define SFS_MAX_FILE_SIZE                   (1024UL * 1024 * 128)               // 128
M

#define SFS_BLKBITS                         (SFS_BLKSIZE * CHAR_BIT)
#define SFS_TYPE_FILE                       1
#define SFS_TYPE_DIR                        2
#define SFS_TYPE_LINK                       3

#define SFS_BLKN_SUPER                      0
#define SFS_BLKN_ROOT                       1
#define SFS_BLKN_FREEMAP                    2

struct cache_block {
    uint32_t ino;
    struct cache_block *hash_next;
    void *cache;
};
```

```
struct cache_inode {
    struct inode {
        uint32_t size;
        uint16_t type;
        uint16_t nlinks;
        uint32_t blocks;
        uint32_t direct[SFS_NDIRECT];
        uint32_t indirect;
        uint32_t db_indirect;
    } inode;
    ino_t real;
    uint32_t ino;
    uint32_t nblks;
    struct cache_block *l1, *l2;
    struct cache_inode *hash_next;
};

struct sfs_fs {
    struct {
        uint32_t magic;
        uint32_t blocks;
        uint32_t unused_blocks;
        char info[SFS_MAX_INFO_LEN + 1];
    } super;
    struct subpath {
        struct subpath *next, *prev;
        char *subname;
    } __sp_nil, *sp_root, *sp_end;
    int imgfd;
    uint32_t ninos, next_ino;
    struct cache_inode *root;
    struct cache_inode *inodes[HASH_LIST_SIZE];
    struct cache_block *blocks[HASH_LIST_SIZE];
};

struct sfs_entry {
    uint32_t ino;
    char name[SFS_MAX_FNAME_LEN + 1];
};

static uint32_t
sfs_alloc_ino(struct sfs_fs *sfs) {
    if (sfs->next_ino < sfs->ninos) {
        sfs->super.unused_blocks --;
        return sfs->next_ino ++;
    }
    bug("out of disk space.\n");
}

static struct cache_block *
alloc_cache_block(struct sfs_fs *sfs, uint32_t ino) {
    struct cache_block *cb = safe_malloc(sizeof(struct cache_block));
    cb->ino = (ino != 0) ? ino : sfs_alloc_ino(sfs);
    cb->cache = memset(safe_malloc(SFS_BLKSIZE), 0, SFS_BLKSIZE);
    struct cache_block **head = sfs->blocks + hash32(ino);
    cb->hash_next = *head, *head = cb;
    return cb;
}

struct cache_block *
search_cache_block(struct sfs_fs *sfs, uint32_t ino) {
    struct cache_block *cb = sfs->blocks[hash32(ino)];
    while (cb != NULL && cb->ino != ino) {
        cb = cb->hash_next;
```

```
    }
    return cb;
}

static struct cache_inode *
alloc_cache_inode(struct sfs_fs *sfs, ino_t real, uint32_t ino, uint16_t type) {
    struct cache_inode *ci = safe_malloc(sizeof(struct cache_inode));
    ci->ino = (ino != 0) ? ino : sfs_alloc_ino(sfs);
    ci->real = real, ci->nblks = 0, ci->l1 = ci->l2 = NULL;
    struct inode *inode = &(ci->inode);
    memset(inode, 0, sizeof(struct inode));
    inode->type = type;
    struct cache_inode **head = sfs->inodes + hash64(real);
    ci->hash_next = *head, *head = ci;
    return ci;
}

struct cache_inode *
search_cache_inode(struct sfs_fs *sfs, ino_t real) {
    struct cache_inode *ci = sfs->inodes[hash64(real)];
    while (ci != NULL && ci->real != real) {
        ci = ci->hash_next;
    }
    return ci;
}

struct sfs_fs *
create_sfs(int imgfd) {
    uint32_t ninos, next_ino;
    struct stat *stat = safe_fstat(imgfd);
    if ((ninos = stat->st_size / SFS_BLKSIZE) > SFS_MAX_NBLKS) {
        ninos = SFS_MAX_NBLKS;
        warn("img file is too big (%llu bytes, only use %u blocks).\n",
                (unsigned long long)stat->st_size, ninos);
    }
    if ((next_ino = SFS_BLKN_FREEMAP + (ninos + SFS_BLKBITS - 1) / SFS_BLKBITS) >= ninos) {
        bug("img file is too small (%llu bytes, %u blocks, bitmap use at least %u blocks).\n",
                (unsigned long long)stat->st_size, ninos, next_ino - 2);
    }

    struct sfs_fs *sfs = safe_malloc(sizeof(struct sfs_fs));
    sfs->super.magic = SFS_MAGIC;
    sfs->super.blocks = ninos, sfs->super.unused_blocks = ninos - next_ino;
    snprintf(sfs->super.info, SFS_MAX_INFO_LEN, "simple file system");

    sfs->ninos = ninos, sfs->next_ino = next_ino, sfs->imgfd = imgfd;
    sfs->sp_root = sfs->sp_end = &(sfs->__sp_nil);
    sfs->sp_end->prev = sfs->sp_end->next = NULL;

    int i;
    for (i = 0; i < HASH_LIST_SIZE; i ++) {
        sfs->inodes[i] = NULL;
        sfs->blocks[i] = NULL;
    }

    sfs->root = alloc_cache_inode(sfs, 0, SFS_BLKN_ROOT, SFS_TYPE_DIR);
    return sfs;
}

static void
subpath_push(struct sfs_fs *sfs, const char *subname) {
    struct subpath *subpath = safe_malloc(sizeof(struct subpath));
    subpath->subname = safe_strdup(subname);
    sfs->sp_end->next = subpath;
    subpath->prev = sfs->sp_end;
```

```
    subpath->next = NULL;
    sfs->sp_end = subpath;
}

static void
subpath_pop(struct sfs_fs *sfs) {
    assert(sfs->sp_root != sfs->sp_end);
    struct subpath *subpath = sfs->sp_end;
    sfs->sp_end = sfs->sp_end->prev, sfs->sp_end->next = NULL;
    free(subpath->subname), free(subpath);
}

static void
subpath_show(FILE *fout, struct sfs_fs *sfs, const char *name) {
    struct subpath *subpath = sfs->sp_root;
    fprintf(fout, "current is: /");
    while ((subpath = subpath->next) != NULL) {
        fprintf(fout, "%s/", subpath->subname);
    }
    if (name != NULL) {
        fprintf(fout, "%s", name);
    }
    fprintf(fout, "\n");
}

static void
write_block(struct sfs_fs *sfs, void *data, size_t len, uint32_t ino) {
    assert(len <= SFS_BLKSIZE && ino < sfs->ninos);
    static char buffer[SFS_BLKSIZE];
    if (len != SFS_BLKSIZE) {
        memset(buffer, 0, sizeof(buffer));
        data = memcpy(buffer, data, len);
    }
    off_t offset = (off_t)ino * SFS_BLKSIZE;
    ssize_t ret;
    if ((ret = pwrite(sfs->imgfd, data, SFS_BLKSIZE, offset)) != SFS_BLKSIZE) {
        bug("write %u block failed: (%d/%d).\n", ino, (int)ret, SFS_BLKSIZE);
    }
}

static void
flush_cache_block(struct sfs_fs *sfs, struct cache_block *cb) {
    write_block(sfs, cb->cache, SFS_BLKSIZE, cb->ino);
}

static void
flush_cache_inode(struct sfs_fs *sfs, struct cache_inode *ci) {
    write_block(sfs, &(ci->inode), sizeof(ci->inode), ci->ino);
}

void
close_sfs(struct sfs_fs *sfs) {
    static char buffer[SFS_BLKSIZE];
    uint32_t i, j, ino = SFS_BLKN_FREEMAP;
    uint32_t ninos = sfs->ninos, next_ino = sfs->next_ino;
    for (i = 0; i < ninos; ino ++, i += SFS_BLKBITS) {
        memset(buffer, 0, sizeof(buffer));
        if (i + SFS_BLKBITS > next_ino) {
            uint32_t start = 0, end = SFS_BLKBITS;
            if (i < next_ino) {
                start = next_ino - i;
            }
            if (i + SFS_BLKBITS > ninos) {
                end = ninos - i;
            }
```

```
            uint32_t *data = (uint32_t *)buffer;
            const uint32_t bits = sizeof(bits) * CHAR_BIT;
            for (j = start; j < end; j ++) {
                data[j / bits] |= (1 << (j % bits));
            }
        }
        write_block(sfs, buffer, sizeof(buffer), ino);
    }
    write_block(sfs, &(sfs->super), sizeof(sfs->super), SFS_BLKN_SUPER);

    for (i = 0; i < HASH_LIST_SIZE; i ++) {
        struct cache_block *cb = sfs->blocks[i];
        while (cb != NULL) {
            flush_cache_block(sfs, cb);
            cb = cb->hash_next;
        }
        struct cache_inode *ci = sfs->inodes[i];
        while (ci != NULL) {
            flush_cache_inode(sfs, ci);
            ci = ci->hash_next;
        }
    }
}

struct sfs_fs *
open_img(const char *imgname) {
    const char *expect = ".img", *ext = imgname + strlen(imgname) - strlen(expect);
    if (ext <= imgname || strcmp(ext, expect) != 0) {
        bug("invalid .img file name '%s'.\n", imgname);
    }
    int imgfd;
    if ((imgfd = open(imgname, O_WRONLY)) < 0) {
        bug("open '%s' failed.\n", imgname);
    }
    return create_sfs(imgfd);
}

#define open_bug(sfs, name, ...)                                                \
    do {                                                                        \
        subpath_show(stderr, sfs, name);                                        \
        bug(__VA_ARGS__);                                                       \
    } while (0)

#define show_fullpath(sfs, name) subpath_show(stderr, sfs, name)

void open_dir(struct sfs_fs *sfs, struct cache_inode *current, struct cache_inode *parent);
void open_file(struct sfs_fs *sfs, struct cache_inode *file, const char *filename, int fd);
void open_link(struct sfs_fs *sfs, struct cache_inode *file, const char *filename);

#define SFS_BLK_NENTRY                          (SFS_BLKSIZE / sizeof(uint32_t))
#define SFS_L0_NBLKS                            SFS_NDIRECT
#define SFS_L1_NBLKS                            (SFS_BLK_NENTRY + SFS_L0_NBLKS)
#define SFS_L2_NBLKS                            (SFS_BLK_NENTRY * SFS_BLK_NENTRY + SFS_L1_NBLK
S)
#define SFS_LN_NBLKS                            (SFS_MAX_FILE_SIZE / SFS_BLKSIZE)

static void
update_cache(struct sfs_fs *sfs, struct cache_block **cbp, uint32_t *inop) {
    uint32_t ino = *inop;
    struct cache_block *cb = *cbp;
    if (ino == 0) {
        cb = alloc_cache_block(sfs, 0);
        ino = cb->ino;
    }
    else if (cb == NULL || cb->ino != ino) {
```

```
        cb = search_cache_block(sfs, ino);
        assert(cb != NULL && cb->ino == ino);
    }
    *cbp = cb, *inop = ino;
}

static void
append_block(struct sfs_fs *sfs, struct cache_inode *file, size_t size, uint32_t ino, const ch
ar *filename) {
    static_assert(SFS_LN_NBLKS <= SFS_L2_NBLKS, "SFS_LN_NBLKS <= SFS_L2_NBLKS");
    assert(size <= SFS_BLKSIZE);
    uint32_t nblks = file->nblks;
    struct inode *inode = &(file->inode);
    if (nblks >= SFS_LN_NBLKS) {
        open_bug(sfs, filename, "file is too big.\n");
    }
    if (nblks < SFS_L0_NBLKS) {
        inode->direct[nblks] = ino;
    }
    else if (nblks < SFS_L1_NBLKS) {
        nblks -= SFS_L0_NBLKS;
        update_cache(sfs, &(file->l1), &(inode->indirect));
        uint32_t *data = file->l1->cache;
        data[nblks] = ino;
    }
    else if (nblks < SFS_L2_NBLKS) {
        nblks -= SFS_L1_NBLKS;
        update_cache(sfs, &(file->l2), &(inode->db_indirect));
        uint32_t *data2 = file->l2->cache;
        update_cache(sfs, &(file->l1), &data2[nblks / SFS_BLK_NENTRY]);
        uint32_t *data1 = file->l1->cache;
        data1[nblks % SFS_BLK_NENTRY] = ino;
    }
    file->nblks ++;
    inode->size += size;
    inode->blocks ++;
}

static void
add_entry(struct sfs_fs *sfs, struct cache_inode *current, struct cache_inode *file, const cha
r *name) {
    static struct sfs_entry __entry, *entry = &__entry;
    assert(current->inode.type == SFS_TYPE_DIR && strlen(name) <= SFS_MAX_FNAME_LEN);
    entry->ino = file->ino, strcpy(entry->name, name);
    uint32_t entry_ino = sfs_alloc_ino(sfs);
    write_block(sfs, entry, sizeof(struct sfs_entry), entry_ino);
    append_block(sfs, current, sizeof(entry->name), entry_ino, name);
    file->inode.nlinks ++;
}

static void
add_dir(struct sfs_fs *sfs, struct cache_inode *parent, const char *dirname, int curfd, int fd
, ino_t real) {
    assert(search_cache_inode(sfs, real) == NULL);
    struct cache_inode *current = alloc_cache_inode(sfs, real, 0, SFS_TYPE_DIR);
    safe_fchdir(fd), subpath_push(sfs, dirname);
    open_dir(sfs, current, parent);
    safe_fchdir(curfd), subpath_pop(sfs);
    add_entry(sfs, parent, current, dirname);
}

static void
add_file(struct sfs_fs *sfs, struct cache_inode *current, const char *filename, int fd, ino_t
real) {
    struct cache_inode *file;
```

```
    if ((file = search_cache_inode(sfs, real)) == NULL) {
        file = alloc_cache_inode(sfs, real, 0, SFS_TYPE_FILE);
        open_file(sfs, file, filename, fd);
    }
    add_entry(sfs, current, file, filename);
}

static void
add_link(struct sfs_fs *sfs, struct cache_inode *current, const char *filename, ino_t real) {
    struct cache_inode *file = alloc_cache_inode(sfs, real, 0, SFS_TYPE_LINK);
    open_link(sfs, file, filename);
    add_entry(sfs, current, file, filename);
}

void
open_dir(struct sfs_fs *sfs, struct cache_inode *current, struct cache_inode *parent) {
    DIR *dir;
    if ((dir = opendir(".")) == NULL) {
        open_bug(sfs, NULL, "opendir failed.\n");
    }
    add_entry(sfs, current, current, ".");
    add_entry(sfs, current, parent, "..");
    struct dirent *direntp;
    while ((direntp = readdir(dir)) != NULL) {
        const char *name = direntp->d_name;
        if (strcmp(name, ".") == 0 || strcmp(name, "..") == 0) {
            continue ;
        }
        if (name[0] == '.') {
            continue ;
        }
        if (strlen(name) > SFS_MAX_FNAME_LEN) {
            open_bug(sfs, NULL, "file name is too long: %s\n", name);
        }
        struct stat *stat = safe_lstat(name);
        if (S_ISLNK(stat->st_mode)) {
            add_link(sfs, current, name, stat->st_ino);
        }
        else {
            int fd;
            if ((fd = open(name, O_RDONLY)) < 0) {
                open_bug(sfs, NULL, "open failed: %s\n", name);
            }
            if (S_ISDIR(stat->st_mode)) {
                add_dir(sfs, current, name, dirfd(dir), fd, stat->st_ino);
            }
            else if (S_ISREG(stat->st_mode)) {
                add_file(sfs, current, name, fd, stat->st_ino);
            }
            else {
                char mode = '?';
                if (S_ISFIFO(stat->st_mode)) mode = 'f';
                if (S_ISSOCK(stat->st_mode)) mode = 's';
                if (S_ISCHR(stat->st_mode)) mode = 'c';
                if (S_ISBLK(stat->st_mode)) mode = 'b';
                show_fullpath(sfs, NULL);
                warn("unsupported mode %07x (%c): file %s\n", stat->st_mode, mode, name);
            }
            close(fd);
        }
    }
    closedir(dir);
}

void
```

```
open_file(struct sfs_fs *sfs, struct cache_inode *file, const char *filename, int fd) {
    static char buffer[SFS_BLKSIZE];
    ssize_t ret, last = SFS_BLKSIZE;
    while ((ret = read(fd, buffer, sizeof(buffer))) != 0) {
        assert(last == SFS_BLKSIZE);
        uint32_t ino = sfs_alloc_ino(sfs);
        write_block(sfs, buffer, ret, ino);
        append_block(sfs, file, ret, ino, filename);
        last = ret;
    }
    if (ret < 0) {
        open_bug(sfs, filename, "read file failed.\n");
    }
}

void
open_link(struct sfs_fs *sfs, struct cache_inode *file, const char *filename) {
    static char buffer[SFS_BLKSIZE];
    uint32_t ino = sfs_alloc_ino(sfs);
    ssize_t ret = readlink(filename, buffer, sizeof(buffer));
    if (ret < 0 || ret == SFS_BLKSIZE) {
        open_bug(sfs, filename, "read link failed, %d", (int)ret);
    }
    write_block(sfs, buffer, ret, ino);
    append_block(sfs, file, ret, ino, filename);
}

int
create_img(struct sfs_fs *sfs, const char *home) {
    int curfd, homefd;
    if ((curfd = open(".", O_RDONLY)) < 0) {
        bug("get current fd failed.\n");
    }
    if ((homefd = open(home, O_RDONLY | O_NOFOLLOW)) < 0) {
        bug("open home directory '%s' failed.\n", home);
    }
    safe_fchdir(homefd);
    open_dir(sfs, sfs->root, sfs->root);
    safe_fchdir(curfd);
    close(curfd), close(homefd);
    close_sfs(sfs);
    return 0;
}

static void
static_check(void) {
#if defined(__i386__)
// IA-32, gcc with -D_FILE_OFFSET_BITS=64
        static_assert(sizeof(off_t) == 8, "sizeof off_t should be 8 in i386");
    static_assert(sizeof(ino_t) == 8,"sizeof ino_t should be 8 in i386");
    printf("in i386 system, need more testing\n");
#elif defined(__x86_64__)
// AMD64, Recommend, gcc with -D_FILE_OFFSET_BITS=64
    static_assert(sizeof(off_t) == 8, "sizeof off_t should be 8 in x86_64");
    static_assert(sizeof(ino_t) == 8, "sizeof ino_t should be 8 in x86_64");
#else
# error Unsupported architecture
#endif
    static_assert(SFS_MAX_NBLKS <= 0x80000000UL, "SFS_MAX_NBLKS <= 0x80000000UL");
    static_assert(SFS_MAX_FILE_SIZE <= 0x80000000UL,"SFS_MAX_FILE_SIZE <= 0x80000000UL");
}

int
main(int argc, char **argv) {
    static_check();
```

```
    if (argc != 3) {
        bug("usage: <input *.img> <input dirname>\n");
    }
    const char *imgname = argv[1], *home = argv[2];
    if (create_img(open_img(imgname), home) != 0) {
        bug("create img failed.\n");
    }
    printf("create %s (%s) successfully.\n", imgname, home);
    return 0;
}
```

[ucore/lab8_result/tools/vector.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>

int
main(void) {
    printf("# handler\n");
    printf(".text\n");
    printf(".globl __alltraps\n");

    int i;
    for (i = 0; i < 256; i ++) {
        printf(".globl vector%d\n", i);
        printf("vector%d:\n", i);
        if ((i < 8 || i > 14) && i != 17) {
            printf("  pushl $0\n");
        }
        printf("  pushl $%d\n", i);
        printf("  jmp __alltraps\n");
    }
    printf("\n");
    printf("# vector table\n");
    printf(".data\n");
    printf(".globl __vectors\n");
    printf("__vectors:\n");
    for (i = 0; i < 256; i ++) {
        printf("  .long vector%d\n", i);
    }
    return 0;
}
```

[ucore/lab8_result/libs/string.h] ++++++++++++++++++++++++++++++

```
#ifndef __LIBS_STRING_H__
#define __LIBS_STRING_H__

#include <defs.h>

size_t strlen(const char *s);
size_t strnlen(const char *s, size_t len);

char *strcpy(char *dst, const char *src);
char *strncpy(char *dst, const char *src, size_t len);
char *strcat(char *dst, const char *src);
char *strdup(const char *src);
char *stradd(const char *src1, const char *src2);

int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);

char *strchr(const char *s, char c);
char *strfind(const char *s, char c);
long strtol(const char *s, char **endptr, int base);

void *memset(void *s, char c, size_t n);
void *memmove(void *dst, const void *src, size_t n);
```

```
void *memcpy(void *dst, const void *src, size_t n);
int memcmp(const void *v1, const void *v2, size_t n);

#endif /* !__LIBS_STRING_H__ */

[ucore/lab8_result/libs/error.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_ERROR_H__
#define __LIBS_ERROR_H__

/* kernel error codes -- keep in sync with list in lib/printfmt.c */
#define E_UNSPECIFIED       1   // Unspecified or unknown problem
#define E_BAD_PROC          2   // Process doesn't exist or otherwise
#define E_INVAL             3   // Invalid parameter
#define E_NO_MEM            4   // Request failed due to memory shortage
#define E_NO_FREE_PROC      5   // Attempt to create a new process beyond
#define E_FAULT             6   // Memory fault
#define E_SWAP_FAULT        7   // SWAP READ/WRITE fault
#define E_INVAL_ELF         8   // Invalid elf file
#define E_KILLED            9   // Process is killed
#define E_PANIC             10  // Panic Failure
#define E_TIMEOUT           11  // Timeout
#define E_TOO_BIG           12  // Argument is Too Big
#define E_NO_DEV            13  // No such Device
#define E_NA_DEV            14  // Device Not Available
#define E_BUSY              15  // Device/File is Busy
#define E_NOENT             16  // No Such File or Directory
#define E_ISDIR             17  // Is a Directory
#define E_NOTDIR            18  // Not a Directory
#define E_XDEV              19  // Cross Device-Link
#define E_UNIMP             20  // Unimplemented Feature
#define E_SEEK              21  // Illegal Seek
#define E_MAX_OPEN          22  // Too Many Files are Open
#define E_EXISTS            23  // File/Directory Already Exists
#define E_NOTEMPTY          24  // Directory is Not Empty
/* the maximum allowed */
#define MAXERROR            24

#endif /* !__LIBS_ERROR_H__ */

[ucore/lab8_result/libs/dirent.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_DIRENT_H__
#define __LIBS_DIRENT_H__

#include <defs.h>
#include <unistd.h>

struct dirent {
    off_t offset;
    char name[FS_MAX_FNAME_LEN + 1];
};

#endif /* !__LIBS_DIRENT_H__ */

[ucore/lab8_result/libs/stdio.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_STDIO_H__
#define __LIBS_STDIO_H__

#include <defs.h>
#include <stdarg.h>

/* kern/libs/stdio.c */
int cprintf(const char *fmt, ...);
int vcprintf(const char *fmt, va_list ap);
void cputchar(int c);
int cputs(const char *str);
```

```
int getchar(void);

/* kern/libs/readline.c */
char *readline(const char *prompt);

/* libs/printfmt.c */
void printfmt(void (*putch)(int, void *, int), int fd, void *putdat, const char *fmt, ...);
void vprintfmt(void (*putch)(int, void *, int), int fd, void *putdat, const char *fmt, va_list
 ap);
int snprintf(char *str, size_t size, const char *fmt, ...);
int vsnprintf(char *str, size_t size, const char *fmt, va_list ap);

#endif /* !__LIBS_STDIO_H__ */

[ucore/lab8_result/libs/string.c] ++++++++++++++++++++++++++++++
#include <string.h>
#include <x86.h>

/* *
 * strlen - calculate the length of the string @s, not including
 * the terminating '\0' character.
 * @s:      the input string
 *
 * The strlen() function returns the length of string @s.
 * */
size_t
strlen(const char *s) {
    size_t cnt = 0;
    while (*s ++ != '\0') {
        cnt ++;
    }
    return cnt;
}

/* *
 * strnlen - calculate the length of the string @s, not including
 * the terminating '\0' char acter, but at most @len.
 * @s:      the input string
 * @len:    the max-length that function will scan
 *
 * Note that, this function looks only at the first @len characters
 * at @s, and never beyond @s + @len.
 *
 * The return value is strlen(s), if that is less than @len, or
 * @len if there is no '\0' character among the first @len characters
 * pointed by @s.
 * */
size_t
strnlen(const char *s, size_t len) {
    size_t cnt = 0;
    while (cnt < len && *s ++ != '\0') {
        cnt ++;
    }
    return cnt;
}

/* *
 * strcat - appends a copy of the @src string to the @dst string. The terminating null
 * character in @dst is overwritten by the first character of @src, and a new null-character
 * is appended at the end of the new string formed by the concatenation of both in @dst.
 * @dst:    pointer to the @dst array, which should be large enough to contain the concatenate
d
 *          resulting string.
 * @src:    string to be appended, this should not overlap @dst
 * */
```

```
char *
strcat(char *dst, const char *src) {
    return strcpy(dst + strlen(dst), src);
}

/* *
 * strcpy - copies the string pointed by @src into the array pointed by @dst,
 * including the terminating null character.
 * @dst:    pointer to the destination array where the content is to be copied
 * @src:    string to be copied
 *
 * The return value is @dst.
 *
 * To avoid overflows, the size of array pointed by @dst should be long enough to
 * contain the same string as @src (including the terminating null character), and
 * should not overlap in memory with @src.
 * */
char *
strcpy(char *dst, const char *src) {
#ifdef __HAVE_ARCH_STRCPY
    return __strcpy(dst, src);
#else
    char *p = dst;
    while ((*p ++ = *src ++) != '\0')
        /* nothing */;
    return dst;
#endif /* __HAVE_ARCH_STRCPY */
}

/* *
 * strncpy - copies the first @len characters of @src to @dst. If the end of string @src
 * if found before @len characters have been copied, @dst is padded with '\0' until a
 * total of @len characters have been written to it.
 * @dst:    pointer to the destination array where the content is to be copied
 * @src:    string to be copied
 * @len:    maximum number of characters to be copied from @src
 *
 * The return value is @dst
 * */
char *
strncpy(char *dst, const char *src, size_t len) {
    char *p = dst;
    while (len > 0) {
        if ((*p = *src) != '\0') {
            src ++;
        }
        p ++, len --;
    }
    return dst;
}

/* *
 * strcmp - compares the string @s1 and @s2
 * @s1:     string to be compared
 * @s2:     string to be compared
 *
 * This function starts comparing the first character of each string. If
 * they are equal to each other, it continues with the following pairs until
 * the characters differ or until a terminanting null-character is reached.
 *
 * Returns an integral value indicating the relationship between the strings:
 * - A zero value indicates that both strings are equal;
 * - A value greater than zero indicates that the first character that does
 *   not match has a greater value in @s1 than in @s2;
 * - And a value less than zero indicates the opposite.
```

```
 * */
int
strcmp(const char *s1, const char *s2) {
#ifdef __HAVE_ARCH_STRCMP
    return __strcmp(s1, s2);
#else
    while (*s1 != '\0' && *s1 == *s2) {
        s1 ++, s2 ++;
    }
    return (int)((unsigned char)*s1 - (unsigned char)*s2);
#endif /* __HAVE_ARCH_STRCMP */
}

/* *
 * strncmp - compares up to @n characters of the string @s1 to those of the string @s2
 * @s1:     string to be compared
 * @s2:     string to be compared
 * @n:      maximum number of characters to compare
 *
 * This function starts comparing the first character of each string. If
 * they are equal to each other, it continues with the following pairs until
 * the characters differ, until a terminating null-character is reached, or
 * until @n characters match in both strings, whichever happens first.
 * */
int
strncmp(const char *s1, const char *s2, size_t n) {
    while (n > 0 && *s1 != '\0' && *s1 == *s2) {
        n --, s1 ++, s2 ++;
    }
    return (n == 0) ? 0 : (int)((unsigned char)*s1 - (unsigned char)*s2);
}

/* *
 * strchr - locates first occurrence of character in string
 * @s:      the input string
 * @c:      character to be located
 *
 * The strchr() function returns a pointer to the first occurrence of
 * character in @s. If the value is not found, the function returns 'NULL'.
 * */
char *
strchr(const char *s, char c) {
    while (*s != '\0') {
        if (*s == c) {
            return (char *)s;
        }
        s ++;
    }
    return NULL;
}

/* *
 * strfind - locates first occurrence of character in string
 * @s:      the input string
 * @c:      character to be located
 *
 * The strfind() function is like strchr() except that if @c is
 * not found in @s, then it returns a pointer to the null byte at the
 * end of @s, rather than 'NULL'.
 * */
char *
strfind(const char *s, char c) {
    while (*s != '\0') {
        if (*s == c) {
            break;
```

```
        }
        s ++;
    }
    return (char *)s;
}

/* *
 * strtol - converts string to long integer
 * @s:      the input string that contains the representation of an integer number
 * @endptr: reference to an object of type char *, whose value is set by the
 *          function to the next character in @s after the numerical value. This
 *          parameter can also be a null pointer, in which case it is not used.
 * @base:   x
 *
 * The function first discards as many whitespace characters as necessary until
 * the first non-whitespace character is found. Then, starting from this character,
 * takes as many characters as possible that are valid following a syntax that
 * depends on the base parameter, and interprets them as a numerical value. Finally,
 * a pointer to the first character following the integer representation in @s
 * is stored in the object pointed by @endptr.
 *
 * If the value of base is zero, the syntax expected is similar to that of
 * integer constants, which is formed by a succession of:
 * - An optional plus or minus sign;
 * - An optional prefix indicating octal or hexadecimal base ("0" or "0x" respectively)
 * - A sequence of decimal digits (if no base prefix was specified) or either octal
 *   or hexadecimal digits if a specific prefix is present
 *
 * If the base value is between 2 and 36, the format expected for the integral number
 * is a succession of the valid digits and/or letters needed to represent integers of
 * the specified radix (starting from '0' and up to 'z'/'Z' for radix 36). The
 * sequence may optionally be preceded by a plus or minus sign and, if base is 16,
 * an optional "0x" or "0X" prefix.
 *
 * The strtol() function returns the converted integral number as a long int value.
 * */
long
strtol(const char *s, char **endptr, int base) {
    int neg = 0;
    long val = 0;

    // gobble initial whitespace
    while (*s == ' ' || *s == '\t') {
        s ++;
    }

    // plus/minus sign
    if (*s == '+') {
        s ++;
    }
    else if (*s == '-') {
        s ++, neg = 1;
    }

    // hex or octal base prefix
    if ((base == 0 || base == 16) && (s[0] == '0' && s[1] == 'x')) {
        s += 2, base = 16;
    }
    else if (base == 0 && s[0] == '0') {
        s ++, base = 8;
    }
    else if (base == 0) {
        base = 10;
    }
```

```
    // digits
    while (1) {
        int dig;

        if (*s >= '0' && *s <= '9') {
            dig = *s - '0';
        }
        else if (*s >= 'a' && *s <= 'z') {
            dig = *s - 'a' + 10;
        }
        else if (*s >= 'A' && *s <= 'Z') {
            dig = *s - 'A' + 10;
        }
        else {
            break;
        }
        if (dig >= base) {
            break;
        }
        s ++, val = (val * base) + dig;
        // we don't properly detect overflow!
    }

    if (endptr) {
        *endptr = (char *) s;
    }
    return (neg ? -val : val);
}

/* *
 * memset - sets the first @n bytes of the memory area pointed by @s
 * to the specified value @c.
 * @s:      pointer the the memory area to fill
 * @c:      value to set
 * @n:      number of bytes to be set to the value
 *
 * The memset() function returns @s.
 * */
void *
memset(void *s, char c, size_t n) {
#ifdef __HAVE_ARCH_MEMSET
    return __memset(s, c, n);
#else
    char *p = s;
    while (n -- > 0) {
        *p ++ = c;
    }
    return s;
#endif /* __HAVE_ARCH_MEMSET */
}

/* *
 * memmove - copies the values of @n bytes from the location pointed by @src to
 * the memory area pointed by @dst. @src and @dst are allowed to overlap.
 * @dst     pointer to the destination array where the content is to be copied
 * @src     pointer to the source of data to by copied
 * @n:      number of bytes to copy
 *
 * The memmove() function returns @dst.
 * */
void *
memmove(void *dst, const void *src, size_t n) {
#ifdef __HAVE_ARCH_MEMMOVE
    return __memmove(dst, src, n);
#else
```

```
    const char *s = src;
    char *d = dst;
    if (s < d && s + n > d) {
        s += n, d += n;
        while (n -- > 0) {
            *-- d = *-- s;
        }
    } else {
        while (n -- > 0) {
            *d ++ = *s ++;
        }
    }
    return dst;
#endif /* __HAVE_ARCH_MEMMOVE */
}

/* *
 * memcpy - copies the value of @n bytes from the location pointed by @src to
 * the memory area pointed by @dst.
 * @dst     pointer to the destination array where the content is to be copied
 * @src     pointer to the source of data to by copied
 * @n:      number of bytes to copy
 *
 * The memcpy() returns @dst.
 *
 * Note that, the function does not check any terminating null character in @src,
 * it always copies exactly @n bytes. To avoid overflows, the size of arrays pointed
 * by both @src and @dst, should be at least @n bytes, and should not overlap
 * (for overlapping memory area, memmove is a safer approach).
 * */
void *
memcpy(void *dst, const void *src, size_t n) {
#ifdef __HAVE_ARCH_MEMCPY
    return __memcpy(dst, src, n);
#else
    const char *s = src;
    char *d = dst;
    while (n -- > 0) {
        *d ++ = *s ++;
    }
    return dst;
#endif /* __HAVE_ARCH_MEMCPY */
}

/* *
 * memcmp - compares two blocks of memory
 * @v1:     pointer to block of memory
 * @v2:     pointer to block of memory
 * @n:      number of bytes to compare
 *
 * The memcmp() functions returns an integral value indicating the
 * relationship between the content of the memory blocks:
 * - A zero value indicates that the contents of both memory blocks are equal;
 * - A value greater than zero indicates that the first byte that does not
 *   match in both memory blocks has a greater value in @v1 than in @v2
 *   as if evaluated as unsigned char values;
 * - And a value less than zero indicates the opposite.
 * */
int
memcmp(const void *v1, const void *v2, size_t n) {
    const char *s1 = (const char *)v1;
    const char *s2 = (const char *)v2;
    while (n -- > 0) {
        if (*s1 != *s2) {
            return (int)((unsigned char)*s1 - (unsigned char)*s2);
```

```
        }
        s1 ++, s2 ++;
    }
    return 0;
}

[ucore/lab8_result/libs/list.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_LIST_H__
#define __LIBS_LIST_H__

#ifndef __ASSEMBLER__

#include <defs.h>

/* *
 * Simple doubly linked list implementation.
 *
 * Some of the internal functions ("__xxx") are useful when manipulating
 * whole lists rather than single entries, as sometimes we already know
 * the next/prev entries and we can generate better code by using them
 * directly rather than using the generic single-entry routines.
 * */

struct list_entry {
    struct list_entry *prev, *next;
};

typedef struct list_entry list_entry_t;

static inline void list_init(list_entry_t *elm) __attribute__((always_inline));
static inline void list_add(list_entry_t *listelm, list_entry_t *elm) __attribute__((always_in
line));
static inline void list_add_before(list_entry_t *listelm, list_entry_t *elm) __attribute__((al
ways_inline));
static inline void list_add_after(list_entry_t *listelm, list_entry_t *elm) __attribute__((alw
ays_inline));
static inline void list_del(list_entry_t *listelm) __attribute__((always_inline));
static inline void list_del_init(list_entry_t *listelm) __attribute__((always_inline));
static inline bool list_empty(list_entry_t *list) __attribute__((always_inline));
static inline list_entry_t *list_next(list_entry_t *listelm) __attribute__((always_inline));
static inline list_entry_t *list_prev(list_entry_t *listelm) __attribute__((always_inline));

static inline void __list_add(list_entry_t *elm, list_entry_t *prev, list_entry_t *next) __att
ribute__((always_inline));
static inline void __list_del(list_entry_t *prev, list_entry_t *next) __attribute__((always_in
line));

/* *
 * list_init - initialize a new entry
 * @elm:        new entry to be initialized
 * */
static inline void
list_init(list_entry_t *elm) {
    elm->prev = elm->next = elm;
}

/* *
 * list_add - add a new entry
 * @listelm:    list head to add after
 * @elm:        new entry to be added
 *
 * Insert the new element @elm *after* the element @listelm which
 * is already in the list.
 * */
static inline void
```

```
list_add(list_entry_t *listelm, list_entry_t *elm) {
    list_add_after(listelm, elm);
}

/* *
 * list_add_before - add a new entry
 * @listelm:    list head to add before
 * @elm:        new entry to be added
 *
 * Insert the new element @elm *before* the element @listelm which
 * is already in the list.
 * */
static inline void
list_add_before(list_entry_t *listelm, list_entry_t *elm) {
    __list_add(elm, listelm->prev, listelm);
}

/* *
 * list_add_after - add a new entry
 * @listelm:    list head to add after
 * @elm:        new entry to be added
 *
 * Insert the new element @elm *after* the element @listelm which
 * is already in the list.
 * */
static inline void
list_add_after(list_entry_t *listelm, list_entry_t *elm) {
    __list_add(elm, listelm, listelm->next);
}

/* *
 * list_del - deletes entry from list
 * @listelm:    the element to delete from the list
 *
 * Note: list_empty() on @listelm does not return true after this, the entry is
 * in an undefined state.
 * */
static inline void
list_del(list_entry_t *listelm) {
    __list_del(listelm->prev, listelm->next);
}

/* *
 * list_del_init - deletes entry from list and reinitialize it.
 * @listelm:    the element to delete from the list.
 *
 * Note: list_empty() on @listelm returns true after this.
 * */
static inline void
list_del_init(list_entry_t *listelm) {
    list_del(listelm);
    list_init(listelm);
}

/* *
 * list_empty - tests whether a list is empty
 * @list:       the list to test.
 * */
static inline bool
list_empty(list_entry_t *list) {
    return list->next == list;
}

/* *
 * list_next - get the next entry
```

```
 * @listelm:    the list head
 **/
static inline list_entry_t *
list_next(list_entry_t *listelm) {
    return listelm->next;
}

/* *
 * list_prev - get the previous entry
 * @listelm:    the list head
 **/
static inline list_entry_t *
list_prev(list_entry_t *listelm) {
    return listelm->prev;
}

/* *
 * Insert a new entry between two known consecutive entries.
 *
 * This is only for internal list manipulation where we know
 * the prev/next entries already!
 * */
static inline void
__list_add(list_entry_t *elm, list_entry_t *prev, list_entry_t *next) {
    prev->next = next->prev = elm;
    elm->next = next;
    elm->prev = prev;
}

/* *
 * Delete a list entry by making the prev/next entries point to each other.
 *
 * This is only for internal list manipulation where we know
 * the prev/next entries already!
 * */
static inline void
__list_del(list_entry_t *prev, list_entry_t *next) {
    prev->next = next;
    next->prev = prev;
}

#endif /* !__ASSEMBLER__ */

#endif /* !__LIBS_LIST_H__ */

[ucore/lab8_result/libs/printfmt.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <x86.h>
#include <error.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

/* *
 * Space or zero padding and a field width are supported for the numeric
 * formats only.
 *
 * The special format %e takes an integer error code
 * and prints a string describing the error.
 * The integer may be positive or negative,
 * so that -E_NO_MEM and E_NO_MEM are equivalent.
 * */

static const char * const error_string[MAXERROR + 1] = {
        [0]                     NULL,
```

```
        [E_UNSPECIFIED]         "unspecified error",
        [E_BAD_PROC]            "bad process",
        [E_INVAL]               "invalid parameter",
        [E_NO_MEM]              "out of memory",
        [E_NO_FREE_PROC]        "out of processes",
        [E_FAULT]               "segmentation fault",
        [E_INVAL_ELF]           "invalid elf file",
        [E_KILLED]              "process is killed",
        [E_PANIC]               "panic failure",
        [E_NO_DEV]              "no such device",
        [E_NA_DEV]              "device not available",
        [E_BUSY]                "device/file is busy",
        [E_NOENT]               "no such file or directory",
        [E_ISDIR]               "is a directory",
        [E_NOTDIR]              "not a directory",
        [E_XDEV]                "cross device link",
        [E_UNIMP]               "unimplemented feature",
        [E_SEEK]                "illegal seek",
        [E_MAX_OPEN]            "too many files are open",
        [E_EXISTS]              "file or directory already exists",
        [E_NOTEMPTY]            "directory is not empty",
};

/* *
 * printnum - print a number (base <= 16) in reverse order
 * @putch:      specified putch function, print a single character
 * @fd:         file descriptor
 * @putdat:     used by @putch function
 * @num:        the number will be printed
 * @base:       base for print, must be in [1, 16]
 * @width:      maximum number of digits, if the actual width is less than @width, use @padc i
nstead
 * @padc:       character that padded on the left if the actual width is less than @width
 * */
static void
printnum(void (*putch)(int, void*, int), int fd, void *putdat,
                unsigned long long num, unsigned base, int width, int padc) {
        unsigned long long result = num;
        unsigned mod = do_div(result, base);

        // first recursively print all preceding (more significant) digits
        if (num >= base) {
                printnum(putch, fd, putdat, result, base, width - 1, padc);
        } else {
                // print any needed pad characters before first digit
                while (-- width > 0)
                        putch(padc, putdat, fd);
        }
        // then print this (the least significant) digit
        putch("0123456789abcdef"[mod], putdat, fd);
}

/* *
 * getuint - get an unsigned int of various possible sizes from a varargs list
 * @ap:         a varargs list pointer
 * @lflag:      determines the size of the vararg that @ap points to
 * */
static unsigned long long
getuint(va_list *ap, int lflag) {
        if (lflag >= 2) {
                return va_arg(*ap, unsigned long long);
        }
        else if (lflag) {
                return va_arg(*ap, unsigned long);
        }
```

```
        else {
                return va_arg(*ap, unsigned int);
        }
}

/* *
 * getint - same as getuint but signed, we can't use getuint because of sign extension
 * @ap:         a varargs list pointer
 * @lflag:      determines the size of the vararg that @ap points to
 * */
static long long
getint(va_list *ap, int lflag) {
        if (lflag >= 2) {
                return va_arg(*ap, long long);
        }
        else if (lflag) {
                return va_arg(*ap, long);
        }
        else {
                return va_arg(*ap, int);
        }
}

/* *
 * printfmt - format a string and print it by using putch
 * @putch:      specified putch function, print a single character
 * @fd:         file descriptor
 * @putdat:     used by @putch function
 * @fmt:        the format string to use
 * */
void
printfmt(void (*putch)(int, void*, int), int fd, void *putdat, const char *fmt, ...) {
        va_list ap;

        va_start(ap, fmt);
        vprintfmt(putch, fd, putdat, fmt, ap);
        va_end(ap);
}

/* *
 * vprintfmt - format a string and print it by using putch, it's called with a va_list
 * instead of a variable number of arguments
 * @fd:         file descriptor
 * @putch:      specified putch function, print a single character
 * @putdat:     used by @putch function
 * @fmt:        the format string to use
 * @ap:         arguments for the format string
 *
 * Call this function if you are already dealing with a va_list.
 * Or you probably want printfmt() instead.
 * */
void
vprintfmt(void (*putch)(int, void*, int), int fd, void *putdat, const char *fmt, va_list ap) {
        register const char *p;
        register int ch, err;
        unsigned long long num;
        int base, width, precision, lflag, altflag;

        while (1) {
                while ((ch = *(unsigned char *)fmt ++) != '%') {
                        if (ch == '\0') {
                                return;
                        }
                        putch(ch, putdat, fd);
                }
```

```
                // Process a %-escape sequence
                char padc = ' ';
                width = precision = -1;
                lflag = altflag = 0;

reswitch:
                switch (ch = *(unsigned char *)fmt ++) {

                        // flag to pad on the right
                        case '-':
                                padc = '-';
                                goto reswitch;

                                // flag to pad with 0's instead of spaces
                        case '0':
                                padc = '0';
                                goto reswitch;

                                // width field
                        case '1' ... '9':
                                for (precision = 0; ; ++ fmt) {
                                        precision = precision * 10 + ch - '0';
                                        ch = *fmt;
                                        if (ch < '0' || ch > '9') {
                                                break;
                                        }
                                }
                                goto process_precision;

                        case '*':
                                precision = va_arg(ap, int);
                                goto process_precision;

                        case '.':
                                if (width < 0)
                                        width = 0;
                                goto reswitch;

                        case '#':
                                altflag = 1;
                                goto reswitch;

process_precision:
                                if (width < 0)
                                        width = precision, precision = -1;
                                goto reswitch;

                                // long flag (doubled for long long)
                        case 'l':
                                lflag ++;
                                goto reswitch;

                                // character
                        case 'c':
                                putch(va_arg(ap, int), putdat, fd);
                                break;

                                // error message
                        case 'e':
                                err = va_arg(ap, int);
                                if (err < 0) {
                                        err = -err;
                                }
                                if (err > MAXERROR || (p = error_string[err]) == NULL) {
```

```
                                        printfmt(putch, fd, putdat, "error %d", err);
                                }
                                else {
                                        printfmt(putch, fd, putdat, "%s", p);
                                }
                                break;

                                // string
                        case 's':
                                if ((p = va_arg(ap, char *)) == NULL) {
                                        p = "(null)";
                                }
                                if (width > 0 && padc != '-') {
                                        for (width -= strnlen(p, precision); width > 0; width
--) {
                                                putch(padc, putdat, fd);
                                        }
                                }
                                for (; (ch = *p ++) != '\0' && (precision < 0 || -- precision
>= 0); width --) {
                                        if (altflag && (ch < ' ' || ch > '~')) {
                                                putch('?', putdat, fd);
                                        }
                                        else {
                                                putch(ch, putdat, fd);
                                        }
                                }
                                for (; width > 0; width --) {
                                        putch(' ', putdat, fd);
                                }
                                break;

                                // (signed) decimal
                        case 'd':
                                num = getint(&ap, lflag);
                                if ((long long)num < 0) {
                                        putch('-', putdat, fd);
                                        num = -(long long)num;
                                }
                                base = 10;
                                goto number;

                                // unsigned decimal
                        case 'u':
                                num = getuint(&ap, lflag);
                                base = 10;
                                goto number;

                                // (unsigned) octal
                        case 'o':
                                num = getuint(&ap, lflag);
                                base = 8;
                                goto number;

                                // pointer
                        case 'p':
                                putch('0', putdat, fd);
                                putch('x', putdat, fd);
                                num = (unsigned long long)(uintptr_t)va_arg(ap, void *);
                                base = 16;
                                goto number;

                                // (unsigned) hexadecimal
                        case 'x':
                                num = getuint(&ap, lflag);
```

```
                                base = 16;
number:
                                printnum(putch, fd, putdat, num, base, width, padc);
                                break;

                                // escaped '%' character
                        case '%':
                                putch(ch, putdat, fd);
                                break;

                                // unrecognized escape sequence - just print it literally
                        default:
                                putch('%', putdat, fd);
                                for (fmt --; fmt[-1] != '%'; fmt --)
                                        /* do nothing */;
                                break;
                }
        }
}

/* sprintbuf is used to save enough information of a buffer */
struct sprintbuf {
        char *buf;          // address pointer points to the first unused memory
        char *ebuf;         // points the end of the buffer
        int cnt;            // the number of characters that have been placed in this buffer
};

/* *
 * sprintputch - 'print' a single character in a buffer
 * @ch:         the character will be printed
 * @b:          the buffer to place the character @ch
 * */
static void
sprintputch(int ch, struct sprintbuf *b) {
        b->cnt ++;
        if (b->buf < b->ebuf) {
                *b->buf ++ = ch;
        }
}

/* *
 * snprintf - format a string and place it in a buffer
 * @str:        the buffer to place the result into
 * @size:       the size of buffer, including the trailing null space
 * @fmt:        the format string to use
 * */
int
snprintf(char *str, size_t size, const char *fmt, ...) {
        va_list ap;
        int cnt;
        va_start(ap, fmt);
        cnt = vsnprintf(str, size, fmt, ap);
        va_end(ap);
        return cnt;
}

/* *
 * vsnprintf - format a string and place it in a buffer, it's called with a va_list
 * instead of a variable number of arguments
 * @str:        the buffer to place the result into
 * @size:       the size of buffer, including the trailing null space
 * @fmt:        the format string to use
 * @ap:         arguments for the format string
 *
 * The return value is the number of characters which would be generated for the
```

```
 * given input, excluding the trailing '\0'.
 *
 * Call this function if you are already dealing with a va_list.
 * Or you probably want snprintf() instead.
 * */
int
vsnprintf(char *str, size_t size, const char *fmt, va_list ap) {
        struct sprintbuf b = {str, str + size - 1, 0};
        if (str == NULL || b.buf > b.ebuf) {
                return -E_INVAL;
        }
        // print the string to the buffer
        vprintfmt((void*)sprintputch, NO_FD, &b, fmt, ap);
        // null terminate the buffer
        *b.buf = '\0';
        return b.cnt;
}

[ucore/lab8_result/libs/elf.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_ELF_H__
#define __LIBS_ELF_H__

#include <defs.h>

#define ELF_MAGIC   0x464C457FU         // "\x7FELF" in little endian

/* file header */
struct elfhdr {
    uint32_t e_magic;     // must equal ELF_MAGIC
    uint8_t e_elf[12];
    uint16_t e_type;      // 1=relocatable, 2=executable, 3=shared object, 4=core image
    uint16_t e_machine;   // 3=x86, 4=68K, etc.
    uint32_t e_version;   // file version, always 1
    uint32_t e_entry;     // entry point if executable
    uint32_t e_phoff;     // file position of program header or 0
    uint32_t e_shoff;     // file position of section header or 0
    uint32_t e_flags;     // architecture-specific flags, usually 0
    uint16_t e_ehsize;    // size of this elf header
    uint16_t e_phentsize; // size of an entry in program header
    uint16_t e_phnum;     // number of entries in program header or 0
    uint16_t e_shentsize; // size of an entry in section header
    uint16_t e_shnum;     // number of entries in section header or 0
    uint16_t e_shstrndx;  // section number that contains section name strings
};

/* program section header */
struct proghdr {
    uint32_t p_type;   // loadable code or data, dynamic linking info,etc.
    uint32_t p_offset; // file offset of segment
    uint32_t p_va;     // virtual address to map segment
    uint32_t p_pa;     // physical address, not used
    uint32_t p_filesz; // size of segment in file
    uint32_t p_memsz;  // size of segment in memory (bigger if contains bssï¼\211
    uint32_t p_flags;  // read/write/execute bits
    uint32_t p_align;  // required alignment, invariably hardware page size
};

/* values for Proghdr::p_type */
#define ELF_PT_LOAD                     1

/* flag bits for Proghdr::p_flags */
#define ELF_PF_X                        1
#define ELF_PF_W                        2
#define ELF_PF_R                        4
```

```
#endif /* !__LIBS_ELF_H__ */

[ucore/lab8_result/libs/rand.c] ++++++++++++++++++++++++++++++
#include <x86.h>
#include <stdlib.h>

static unsigned long long next = 1;

/* *
 * rand - returns a pseudo-random integer
 *
 * The rand() function return a value in the range [0, RAND_MAX].
 * */
int
rand(void) {
    next = (next * 0x5DEECE66DLL + 0xBLL) & ((1LL << 48) - 1);
    unsigned long long result = (next >> 12);
    return (int)do_div(result, RAND_MAX + 1);
}

/* *
 * srand - seed the random number generator with the given number
 * @seed:   the required seed number
 * */
void
srand(unsigned int seed) {
    next = seed;
}

[ucore/lab8_result/libs/defs.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_DEFS_H__
#define __LIBS_DEFS_H__

#ifndef NULL
#define NULL ((void *)0)
#endif

#define __always_inline inline __attribute__((always_inline))
#define __noinline __attribute__((noinline))
#define __noreturn __attribute__((noreturn))

#define CHAR_BIT        8

/* Represents true-or-false values */
typedef int bool;

/* Explicitly-sized versions of integer types */
typedef char int8_t;
typedef unsigned char uint8_t;
typedef short int16_t;
typedef unsigned short uint16_t;
typedef int int32_t;
typedef unsigned int uint32_t;
typedef long long int64_t;
typedef unsigned long long uint64_t;

/* *
 * Pointers and addresses are 32 bits long.
 * We use pointer types to represent addresses,
 * uintptr_t to represent the numerical values of addresses.
 * */
typedef int32_t intptr_t;
typedef uint32_t uintptr_t;

/* size_t is used for memory object sizes */
```

```
typedef uintptr_t size_t;

/* off_t is used for file offsets and lengths */
typedef intptr_t off_t;

/* used for page numbers */
typedef size_t ppn_t;

/* *
 * Rounding operations (efficient when n is a power of 2)
 * Round down to the nearest multiple of n
 * */
#define ROUNDDOWN(a, n) ({                                          \
            size_t __a = (size_t)(a);                               \
            (typeof(a))(__a - __a % (n));                           \
        })

/* Round up to the nearest multiple of n */
#define ROUNDUP(a, n) ({                                            \
            size_t __n = (size_t)(n);                               \
            (typeof(a))(ROUNDDOWN((size_t)(a) + __n - 1, __n));     \
        })

/* Round up the result of dividing of n */
#define ROUNDUP_DIV(a, n) ({                                        \
uint32_t __n = (uint32_t)(n);                                \
(typeof(a))(((a) + __n - 1) / __n);                          \
})

/* Return the offset of 'member' relative to the beginning of a struct type */
#define offsetof(type, member)                                      \
    ((size_t)(&((type *)0)->member))

/* *
 * to_struct - get the struct from a ptr
 * @ptr:    a struct pointer of member
 * @type:   the type of the struct this is embedded in
 * @member: the name of the member within the struct
 * */
#define to_struct(ptr, type, member)                                \
    ((type *)((char *)(ptr) - offsetof(type, member)))

#endif /* !__LIBS_DEFS_H__ */

[ucore/lab8_result/libs/stat.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_STAT_H__
#define __LIBS_STAT_H__

#include <defs.h>

struct stat {
    uint32_t st_mode;                   // protection mode and file type
    size_t st_nlinks;                   // number of hard links
    size_t st_blocks;                   // number of blocks file is using
    size_t st_size;                     // file size (bytes)
};

#define S_IFMT          070000          // mask for type of file
#define S_IFREG         010000          // ordinary regular file
#define S_IFDIR         020000          // directory
#define S_IFLNK         030000          // symbolic link
#define S_IFCHR         040000          // character device
#define S_IFBLK         050000          // block device

#define S_ISREG(mode)                   (((mode) & S_IFMT) == S_IFREG)      // regular file
```

```
#define S_ISDIR(mode)                  (((mode) & S_IFMT) == S_IFDIR)       // directory
#define S_ISLNK(mode)                  (((mode) & S_IFMT) == S_IFLNK)       // symlink
#define S_ISCHR(mode)                  (((mode) & S_IFMT) == S_IFCHR)       // char device
#define S_ISBLK(mode)                  (((mode) & S_IFMT) == S_IFBLK)       // block device

#endif /* !__LIBS_STAT_H__ */
```

```
[ucore/lab8_result/libs/hash.c] ++++++++++++++++++++++++++++++
#include <stdlib.h>

/* 2^31 + 2^29 - 2^25 + 2^22 - 2^19 - 2^16 + 1 */
#define GOLDEN_RATIO_PRIME_32       0x9e370001UL

/* *
 * hash32 - generate a hash value in the range [0, 2^@bits - 1]
 * @val:    the input value
 * @bits:   the number of bits in a return value
 *
 * High bits are more random, so we use them.
 * */
uint32_t
hash32(uint32_t val, unsigned int bits) {
    uint32_t hash = val * GOLDEN_RATIO_PRIME_32;
    return (hash >> (32 - bits));
}
```

```
[ucore/lab8_result/libs/x86.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_X86_H__
#define __LIBS_X86_H__

#include <defs.h>

#define do_div(n, base) ({                                              \
            unsigned long __upper, __low, __high, __mod, __base;        \
            __base = (base);                                            \
            asm ("" : "=a" (__low), "=d" (__high) : "A" (n));           \
            __upper = __high;                                           \
            if (__high != 0) {                                          \
                __upper = __high % __base;                              \
                __high = __high / __base;                               \
            }                                                           \
            asm ("divl %2" : "=a" (__low), "=d" (__mod)                 \
                : "rm" (__base), "0" (__low), "1" (__upper));           \
            asm ("" : "=A" (n) : "a" (__low), "d" (__high));            \
            __mod;                                                      \
        })

#define barrier() __asm__ __volatile__ ("" ::: "memory")

static inline uint8_t inb(uint16_t port) __attribute__((always_inline));
static inline uint16_t inw(uint16_t port) __attribute__((always_inline));
static inline void insl(uint32_t port, void *addr, int cnt) __attribute__((always_inline));
static inline void outb(uint16_t port, uint8_t data) __attribute__((always_inline));
static inline void outw(uint16_t port, uint16_t data) __attribute__((always_inline));
static inline void outsl(uint32_t port, const void *addr, int cnt) __attribute__((always_inlin
e));
static inline uint32_t read_ebp(void) __attribute__((always_inline));
static inline void breakpoint(void) __attribute__((always_inline));
static inline uint32_t read_dr(unsigned regnum) __attribute__((always_inline));
static inline void write_dr(unsigned regnum, uint32_t value) __attribute__((always_inline));

/* Pseudo-descriptors used for LGDT, LLDT(not used) and LIDT instructions. */
struct pseudodesc {
    uint16_t pd_lim;        // Limit
    uintptr_t pd_base;      // Base address
```

```
} __attribute__ ((packed));

static inline void lidt(struct pseudodesc *pd) __attribute__((always_inline));
static inline void sti(void) __attribute__((always_inline));
static inline void cli(void) __attribute__((always_inline));
static inline void ltr(uint16_t sel) __attribute__((always_inline));
static inline uint32_t read_eflags(void) __attribute__((always_inline));
static inline void write_eflags(uint32_t eflags) __attribute__((always_inline));
static inline void lcr0(uintptr_t cr0) __attribute__((always_inline));
static inline void lcr3(uintptr_t cr3) __attribute__((always_inline));
static inline uintptr_t rcr0(void) __attribute__((always_inline));
static inline uintptr_t rcr1(void) __attribute__((always_inline));
static inline uintptr_t rcr2(void) __attribute__((always_inline));
static inline uintptr_t rcr3(void) __attribute__((always_inline));
static inline void invlpg(void *addr) __attribute__((always_inline));

static inline uint8_t
inb(uint16_t port) {
    uint8_t data;
    asm volatile ("inb %1, %0" : "=a" (data) : "d" (port) : "memory");
    return data;
}

static inline uint16_t
inw(uint16_t port) {
    uint16_t data;
    asm volatile ("inw %1, %0" : "=a" (data) : "d" (port));
    return data;
}

static inline void
insl(uint32_t port, void *addr, int cnt) {
    asm volatile (
        "cld;"
        "repne; insl;"
        : "=D" (addr), "=c" (cnt)
        : "d" (port), "0" (addr), "1" (cnt)
        : "memory", "cc");
}

static inline void
outb(uint16_t port, uint8_t data) {
    asm volatile ("outb %0, %1" :: "a" (data), "d" (port) : "memory");
}

static inline void
outw(uint16_t port, uint16_t data) {
    asm volatile ("outw %0, %1" :: "a" (data), "d" (port) : "memory");
}

static inline void
outsl(uint32_t port, const void *addr, int cnt) {
    asm volatile (
        "cld;"
        "repne; outsl;"
        : "=S" (addr), "=c" (cnt)
        : "d" (port), "0" (addr), "1" (cnt)
        : "memory", "cc");
}

static inline uint32_t
read_ebp(void) {
    uint32_t ebp;
    asm volatile ("movl %%ebp, %0" : "=r" (ebp));
    return ebp;
```

```
}

static inline void
breakpoint(void) {
    asm volatile ("int $3");
}

static inline uint32_t
read_dr(unsigned regnum) {
    uint32_t value = 0;
    switch (regnum) {
    case 0: asm volatile ("movl %%db0, %0" : "=r" (value)); break;
    case 1: asm volatile ("movl %%db1, %0" : "=r" (value)); break;
    case 2: asm volatile ("movl %%db2, %0" : "=r" (value)); break;
    case 3: asm volatile ("movl %%db3, %0" : "=r" (value)); break;
    case 6: asm volatile ("movl %%db6, %0" : "=r" (value)); break;
    case 7: asm volatile ("movl %%db7, %0" : "=r" (value)); break;
    }
    return value;
}

static void
write_dr(unsigned regnum, uint32_t value) {
    switch (regnum) {
    case 0: asm volatile ("movl %0, %%db0" :: "r" (value)); break;
    case 1: asm volatile ("movl %0, %%db1" :: "r" (value)); break;
    case 2: asm volatile ("movl %0, %%db2" :: "r" (value)); break;
    case 3: asm volatile ("movl %0, %%db3" :: "r" (value)); break;
    case 6: asm volatile ("movl %0, %%db6" :: "r" (value)); break;
    case 7: asm volatile ("movl %0, %%db7" :: "r" (value)); break;
    }
}

static inline void
lidt(struct pseudodesc *pd) {
    asm volatile ("lidt (%0)" :: "r" (pd) : "memory");
}

static inline void
sti(void) {
    asm volatile ("sti");
}

static inline void
cli(void) {
    asm volatile ("cli" ::: "memory");
}

static inline void
ltr(uint16_t sel) {
    asm volatile ("ltr %0" :: "r" (sel) : "memory");
}

static inline uint32_t
read_eflags(void) {
    uint32_t eflags;
    asm volatile ("pushfl; popl %0" : "=r" (eflags));
    return eflags;
}

static inline void
write_eflags(uint32_t eflags) {
    asm volatile ("pushl %0; popfl" :: "r" (eflags));
}
```

```
static inline void
lcr0(uintptr_t cr0) {
    asm volatile ("mov %0, %%cr0" :: "r" (cr0) : "memory");
}

static inline void
lcr3(uintptr_t cr3) {
    asm volatile ("mov %0, %%cr3" :: "r" (cr3) : "memory");
}

static inline uintptr_t
rcr0(void) {
    uintptr_t cr0;
    asm volatile ("mov %%cr0, %0" : "=r" (cr0) :: "memory");
    return cr0;
}

static inline uintptr_t
rcr1(void) {
    uintptr_t cr1;
    asm volatile ("mov %%cr1, %0" : "=r" (cr1) :: "memory");
    return cr1;
}

static inline uintptr_t
rcr2(void) {
    uintptr_t cr2;
    asm volatile ("mov %%cr2, %0" : "=r" (cr2) :: "memory");
    return cr2;
}

static inline uintptr_t
rcr3(void) {
    uintptr_t cr3;
    asm volatile ("mov %%cr3, %0" : "=r" (cr3) :: "memory");
    return cr3;
}

static inline void
invlpg(void *addr) {
    asm volatile ("invlpg (%0)" :: "r" (addr) : "memory");
}

static inline int __strcmp(const char *s1, const char *s2) __attribute__((always_inline));
static inline char *__strcpy(char *dst, const char *src) __attribute__((always_inline));
static inline void *__memset(void *s, char c, size_t n) __attribute__((always_inline));
static inline void *__memmove(void *dst, const void *src, size_t n) __attribute__((always_inli
ne));
static inline void *__memcpy(void *dst, const void *src, size_t n) __attribute__((always_inlin
e));

#ifndef __HAVE_ARCH_STRCMP
#define __HAVE_ARCH_STRCMP
static inline int
__strcmp(const char *s1, const char *s2) {
    int d0, d1, ret;
    asm volatile (
        "1: lodsb;"
        "scasb;"
        "jne 2f;"
        "testb %%al, %%al;"
        "jne 1b;"
        "xorl %%eax, %%eax;"
        "jmp 3f;"
        "2: sbbl %%eax, %%eax;"
```

```
        "orb $1, %%al;"
        "3:"
        : "=a" (ret), "=&S" (d0), "=&D" (d1)
        : "1" (s1), "2" (s2)
        : "memory");
    return ret;
}

#endif /* __HAVE_ARCH_STRCMP */

#ifndef __HAVE_ARCH_STRCPY
#define __HAVE_ARCH_STRCPY
static inline char *
__strcpy(char *dst, const char *src) {
    int d0, d1, d2;
    asm volatile (
        "1: lodsb;"
        "stosb;"
        "testb %%al, %%al;"
        "jne 1b;"
        : "=&S" (d0), "=&D" (d1), "=&a" (d2)
        : "0" (src), "1" (dst) : "memory");
    return dst;
}
#endif /* __HAVE_ARCH_STRCPY */

#ifndef __HAVE_ARCH_MEMSET
#define __HAVE_ARCH_MEMSET
static inline void *
__memset(void *s, char c, size_t n) {
    int d0, d1;
    asm volatile (
        "rep; stosb;"
        : "=&c" (d0), "=&D" (d1)
        : "0" (n), "a" (c), "1" (s)
        : "memory");
    return s;
}
#endif /* __HAVE_ARCH_MEMSET */

#ifndef __HAVE_ARCH_MEMMOVE
#define __HAVE_ARCH_MEMMOVE
static inline void *
__memmove(void *dst, const void *src, size_t n) {
    if (dst < src) {
        return __memcpy(dst, src, n);
    }
    int d0, d1, d2;
    asm volatile (
        "std;"
        "rep; movsb;"
        "cld;"
        : "=&c" (d0), "=&S" (d1), "=&D" (d2)
        : "0" (n), "1" (n - 1 + src), "2" (n - 1 + dst)
        : "memory");
    return dst;
}
#endif /* __HAVE_ARCH_MEMMOVE */

#ifndef __HAVE_ARCH_MEMCPY
#define __HAVE_ARCH_MEMCPY
static inline void *
__memcpy(void *dst, const void *src, size_t n) {
    int d0, d1, d2;
    asm volatile (
```

```
        "rep; movsl;"
        "movl %4, %%ecx;"
        "andl $3, %%ecx;"
        "jz 1f;"
        "rep; movsb;"
        "1:"
        : "=&c" (d0), "=&D" (d1), "=&S" (d2)
        : "0" (n / 4), "g" (n), "1" (dst), "2" (src)
        : "memory");
    return dst;
}
#endif /* __HAVE_ARCH_MEMCPY */

#endif /* !__LIBS_X86_H__ */

[ucore/lab8_result/libs/unistd.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_UNISTD_H__
#define __LIBS_UNISTD_H__

#define T_SYSCALL           0x80

/* syscall number */
#define SYS_exit            1
#define SYS_fork            2
#define SYS_wait            3
#define SYS_exec            4
#define SYS_clone           5
#define SYS_yield           10
#define SYS_sleep           11
#define SYS_kill            12
#define SYS_gettime         17
#define SYS_getpid          18
#define SYS_mmap            20
#define SYS_munmap          21
#define SYS_shmem           22
#define SYS_putc            30
#define SYS_pgdir           31
#define SYS_open            100
#define SYS_close           101
#define SYS_read            102
#define SYS_write           103
#define SYS_seek            104
#define SYS_fstat           110
#define SYS_fsync           111
#define SYS_getcwd          121
#define SYS_getdirentry     128
#define SYS_dup             130
/* OLNY FOR LAB6 */
#define SYS_lab6_set_priority 255

/* SYS_fork flags */
#define CLONE_VM            0x00000100  // set if VM shared between processes
#define CLONE_THREAD        0x00000200  // thread group
#define CLONE_FS            0x00000800  // set if shared between processes

/* VFS flags */
// flags for open: choose one of these
#define O_RDONLY            0           // open for reading only
#define O_WRONLY            1           // open for writing only
#define O_RDWR              2           // open for reading and writing
// then or in any of these:
#define O_CREAT             0x00000004  // create file if it does not exist
#define O_EXCL              0x00000008  // error if O_CREAT and the file exists
#define O_TRUNC             0x00000010  // truncate file upon open
#define O_APPEND            0x00000020  // append on each write
```

```
// additonal related definition
#define O_ACCMODE           3           // mask for O_RDONLY / O_WRONLY / O_RDWR

#define NO_FD               -0x9527     // invalid fd

/* lseek codes */
#define LSEEK_SET           0           // seek relative to beginning of file
#define LSEEK_CUR           1           // seek relative to current position in file
#define LSEEK_END           2           // seek relative to end of file

#define FS_MAX_DNAME_LEN    31
#define FS_MAX_FNAME_LEN    255
#define FS_MAX_FPATH_LEN    4095

#define EXEC_MAX_ARG_NUM    32
#define EXEC_MAX_ARG_LEN    4095

#endif /* !__LIBS_UNISTD_H__ */

[ucore/lab8_result/libs/stdarg.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_STDARG_H__
#define __LIBS_STDARG_H__

/* compiler provides size of save area */
typedef __builtin_va_list va_list;

#define va_start(ap, last)              (__builtin_va_start(ap, last))
#define va_arg(ap, type)                (__builtin_va_arg(ap, type))
#define va_end(ap)                      /*nothing*/

#endif /* !__LIBS_STDARG_H__ */

[ucore/lab8_result/libs/stdlib.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_STDLIB_H__
#define __LIBS_STDLIB_H__

#include <defs.h>

/* the largest number rand will return */
#define RAND_MAX    2147483647UL

/* libs/rand.c */
int rand(void);
void srand(unsigned int seed);

/* libs/hash.c */
uint32_t hash32(uint32_t val, unsigned int bits);

#endif /* !__LIBS_RAND_H__ */

[ucore/lab8_result/libs/atomic.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_ATOMIC_H__
#define __LIBS_ATOMIC_H__

/* Atomic operations that C can't guarantee us. Useful for resource counting etc.. */

static inline void set_bit(int nr, volatile void *addr) __attribute__((always_inline));
static inline void clear_bit(int nr, volatile void *addr) __attribute__((always_inline));
static inline void change_bit(int nr, volatile void *addr) __attribute__((always_inline));
static inline bool test_and_set_bit(int nr, volatile void *addr) __attribute__((always_inline)
);
static inline bool test_and_clear_bit(int nr, volatile void *addr) __attribute__((always_inlin
e));
static inline bool test_bit(int nr, volatile void *addr) __attribute__((always_inline));
```

```
/* *
 * set_bit - Atomically set a bit in memory
 * @nr:     the bit to set
 * @addr:   the address to start counting from
 *
 * Note that @nr may be almost arbitrarily large; this function is not
 * restricted to acting on a single-word quantity.
 * */
static inline void
set_bit(int nr, volatile void *addr) {
    asm volatile ("btsl %1, %0" :"=m" (*(volatile long *)addr) : "Ir" (nr));
}

/* *
 * clear_bit - Atomically clears a bit in memory
 * @nr:     the bit to clear
 * @addr:   the address to start counting from
 * */
static inline void
clear_bit(int nr, volatile void *addr) {
    asm volatile ("btrl %1, %0" :"=m" (*(volatile long *)addr) : "Ir" (nr));
}

/* *
 * change_bit - Atomically toggle a bit in memory
 * @nr:     the bit to change
 * @addr:   the address to start counting from
 * */
static inline void
change_bit(int nr, volatile void *addr) {
    asm volatile ("btcl %1, %0" :"=m" (*(volatile long *)addr) : "Ir" (nr));
}

/* *
 * test_bit - Determine whether a bit is set
 * @nr:     the bit to test
 * @addr:   the address to count from
 * */
static inline bool
test_bit(int nr, volatile void *addr) {
    int oldbit;
    asm volatile ("btl %2, %1; sbbl %0,%0" : "=r" (oldbit) : "m" (*(volatile long *)addr), "Ir
" (nr));
    return oldbit != 0;
}

/* *
 * test_and_set_bit - Atomically set a bit and return its old value
 * @nr:     the bit to set
 * @addr:   the address to count from
 * */
static inline bool
test_and_set_bit(int nr, volatile void *addr) {
    int oldbit;
    asm volatile ("btsl %2, %1; sbbl %0, %0" : "=r" (oldbit), "=m" (*(volatile long *)addr) :
"Ir" (nr) : "memory");
    return oldbit != 0;
}

/* *
 * test_and_clear_bit - Atomically clear a bit and return its old value
 * @nr:     the bit to clear
 * @addr:   the address to count from
 * */
static inline bool
```

```
test_and_clear_bit(int nr, volatile void *addr) {
    int oldbit;
    asm volatile ("btrl %2, %1; sbbl %0, %0" : "=r" (oldbit), "=m" (*(volatile long *)addr) :
"Ir" (nr) : "memory");
    return oldbit != 0;
}
#endif /* !__LIBS_ATOMIC_H__ */

[ucore/lab8_result/libs/skew_heap.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_SKEW_HEAP_H__
#define __LIBS_SKEW_HEAP_H__

struct skew_heap_entry {
     struct skew_heap_entry *parent, *left, *right;
};

typedef struct skew_heap_entry skew_heap_entry_t;

typedef int(*compare_f)(void *a, void *b);

static inline void skew_heap_init(skew_heap_entry_t *a) __attribute__((always_inline));
static inline skew_heap_entry_t *skew_heap_merge(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp);
static inline skew_heap_entry_t *skew_heap_insert(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp) __attribute__((always_inline));
static inline skew_heap_entry_t *skew_heap_remove(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp) __attribute__((always_inline));

static inline void
skew_heap_init(skew_heap_entry_t *a)
{
     a->left = a->right = a->parent = NULL;
}

static inline skew_heap_entry_t *
skew_heap_merge(skew_heap_entry_t *a, skew_heap_entry_t *b,
                compare_f comp)
{
     if (a == NULL) return b;
     else if (b == NULL) return a;

     skew_heap_entry_t *l, *r;
     if (comp(a, b) == -1)
     {
          r = a->left;
          l = skew_heap_merge(a->right, b, comp);

          a->left = l;
          a->right = r;
          if (l) l->parent = a;

          return a;
     }
     else
     {
          r = b->left;
          l = skew_heap_merge(a, b->right, comp);

          b->left = l;
          b->right = r;
          if (l) l->parent = b;
```

```
          return b;
     }
}

static inline skew_heap_entry_t *
skew_heap_insert(skew_heap_entry_t *a, skew_heap_entry_t *b,
                 compare_f comp)
{
     skew_heap_init(b);
     return skew_heap_merge(a, b, comp);
}

static inline skew_heap_entry_t *
skew_heap_remove(skew_heap_entry_t *a, skew_heap_entry_t *b,
                 compare_f comp)
{
     skew_heap_entry_t *p   = b->parent;
     skew_heap_entry_t *rep = skew_heap_merge(b->left, b->right, comp);
     if (rep) rep->parent = p;

     if (p)
     {
          if (p->left == b)
               p->left = rep;
          else p->right = rep;
          return a;
     }
     else return rep;
}

#endif    /* !__LIBS_SKEW_HEAP_H__ */
[ucore/lab8_result/kern/driver/clock.c] ++++++++++++++++++++++++++++++
#include <x86.h>
#include <trap.h>
#include <stdio.h>
#include <picirq.h>

/* *
 * Support for time-related hardware gadgets - the 8253 timer,
 * which generates interruptes on IRQ-0.
 * */

#define IO_TIMER1           0x040               // 8253 Timer #1

/* *
 * Frequency of all three count-down timers; (TIMER_FREQ/freq)
 * is the appropriate count to generate a frequency of freq Hz.
 * */

#define TIMER_FREQ      1193182
#define TIMER_DIV(x)    ((TIMER_FREQ + (x) / 2) / (x))

#define TIMER_MODE      (IO_TIMER1 + 3)         // timer mode port
#define TIMER_SEL0      0x00                    // select counter 0
#define TIMER_RATEGEN   0x04                    // mode 2, rate generator
#define TIMER_16BIT     0x30                    // r/w counter 16 bits, LSB first

volatile size_t ticks;

long SYSTEM_READ_TIMER( void ){
    return ticks;
}

/* *
 * clock_init - initialize 8253 clock to interrupt 100 times per second,
```

```
 * and then enable IRQ_TIMER.
 * */
void
clock_init(void) {
    // set 8253 timer-chip
    outb(TIMER_MODE, TIMER_SEL0 | TIMER_RATEGEN | TIMER_16BIT);
    outb(IO_TIMER1, TIMER_DIV(100) % 256);
    outb(IO_TIMER1, TIMER_DIV(100) / 256);

    // initialize time counter 'ticks' to zero
    ticks = 0;

    cprintf("++ setup timer interrupts\n");
    pic_enable(IRQ_TIMER);
}
```

[ucore/lab8_result/kern/driver/console.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DRIVER_CONSOLE_H__
#define __KERN_DRIVER_CONSOLE_H__

void cons_init(void);
void cons_putc(int c);
int cons_getc(void);
void serial_intr(void);
void kbd_intr(void);

#endif /* !__KERN_DRIVER_CONSOLE_H__ */
```

[ucore/lab8_result/kern/driver/ide.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <stdio.h>
#include <trap.h>
#include <picirq.h>
#include <fs.h>
#include <ide.h>
#include <x86.h>
#include <assert.h>

#define ISA_DATA                0x00
#define ISA_ERROR               0x01
#define ISA_PRECOMP             0x01
#define ISA_CTRL                0x02
#define ISA_SECCNT              0x02
#define ISA_SECTOR              0x03
#define ISA_CYL_LO              0x04
#define ISA_CYL_HI              0x05
#define ISA_SDH                 0x06
#define ISA_COMMAND             0x07
#define ISA_STATUS              0x07

#define IDE_BSY                 0x80
#define IDE_DRDY                0x40
#define IDE_DF                  0x20
#define IDE_DRQ                 0x08
#define IDE_ERR                 0x01

#define IDE_CMD_READ            0x20
#define IDE_CMD_WRITE           0x30
#define IDE_CMD_IDENTIFY        0xEC

#define IDE_IDENT_SECTORS       20
#define IDE_IDENT_MODEL         54
#define IDE_IDENT_CAPABILITIES  98
#define IDE_IDENT_CMDSETS       164
#define IDE_IDENT_MAX_LBA       120
```

```
#define IDE_IDENT_MAX_LBA_EXT   200

#define IO_BASE0                0x1F0
#define IO_BASE1                0x170
#define IO_CTRL0                0x3F4
#define IO_CTRL1                0x374

#define MAX_IDE                 4
#define MAX_NSECS               128
#define MAX_DISK_NSECS          0x10000000U
#define VALID_IDE(ideno)        (((ideno) >= 0) && ((ideno) < MAX_IDE) && (ide_devices[ideno].
valid))

static const struct {
    unsigned short base;        // I/O Base
    unsigned short ctrl;        // Control Base
} channels[2] = {
    {IO_BASE0, IO_CTRL0},
    {IO_BASE1, IO_CTRL1},
};

#define IO_BASE(ideno)          (channels[(ideno) >> 1].base)
#define IO_CTRL(ideno)          (channels[(ideno) >> 1].ctrl)

static struct ide_device {
    unsigned char valid;        // 0 or 1 (If Device Really Exists)
    unsigned int sets;          // Commend Sets Supported
    unsigned int size;          // Size in Sectors
    unsigned char model[41];    // Model in String
} ide_devices[MAX_IDE];

static int
ide_wait_ready(unsigned short iobase, bool check_error) {
    int r;
    while ((r = inb(iobase + ISA_STATUS)) & IDE_BSY)
        /* nothing */;
    if (check_error && (r & (IDE_DF | IDE_ERR)) != 0) {
        return -1;
    }
    return 0;
}

void
ide_init(void) {
    static_assert((SECTSIZE % 4) == 0);
    unsigned short ideno, iobase;
    for (ideno = 0; ideno < MAX_IDE; ideno ++) {
        /* assume that no device here */
        ide_devices[ideno].valid = 0;

        iobase = IO_BASE(ideno);

        /* wait device ready */
        ide_wait_ready(iobase, 0);

        /* step1: select drive */
        outb(iobase + ISA_SDH, 0xE0 | ((ideno & 1) << 4));
        ide_wait_ready(iobase, 0);

        /* step2: send ATA identify command */
        outb(iobase + ISA_COMMAND, IDE_CMD_IDENTIFY);
        ide_wait_ready(iobase, 0);

        /* step3: polling */
        if (inb(iobase + ISA_STATUS) == 0 || ide_wait_ready(iobase, 1) != 0) {
```

```
            continue ;
        }

        /* device is ok */
        ide_devices[ideno].valid = 1;

        /* read identification space of the device */
        unsigned int buffer[128];
        insl(iobase + ISA_DATA, buffer, sizeof(buffer) / sizeof(unsigned int));

        unsigned char *ident = (unsigned char *)buffer;
        unsigned int sectors;
        unsigned int cmdsets = *(unsigned int *)(ident + IDE_IDENT_CMDSETS);
        /* device use 48-bits or 28-bits addressing */
        if (cmdsets & (1 << 26)) {
            sectors = *(unsigned int *)(ident + IDE_IDENT_MAX_LBA_EXT);
        }
        else {
            sectors = *(unsigned int *)(ident + IDE_IDENT_MAX_LBA);
        }
        ide_devices[ideno].sets = cmdsets;
        ide_devices[ideno].size = sectors;

        /* check if supports LBA */
        assert((*(unsigned short *)(ident + IDE_IDENT_CAPABILITIES) & 0x200) != 0);

        unsigned char *model = ide_devices[ideno].model, *data = ident + IDE_IDENT_MODEL;
        unsigned int i, length = 40;
        for (i = 0; i < length; i += 2) {
            model[i] = data[i + 1], model[i + 1] = data[i];
        }
        do {
            model[i] = '\0';
        } while (i -- > 0 && model[i] == ' ');

        cprintf("ide %d: %10u(sectors), '%s'.\n", ideno, ide_devices[ideno].size, ide_devices[
ideno].model);
    }

    // enable ide interrupt
    pic_enable(IRQ_IDE1);
    pic_enable(IRQ_IDE2);
}

bool
ide_device_valid(unsigned short ideno) {
    return VALID_IDE(ideno);
}

size_t
ide_device_size(unsigned short ideno) {
    if (ide_device_valid(ideno)) {
        return ide_devices[ideno].size;
    }
    return 0;
}

int
ide_read_secs(unsigned short ideno, uint32_t secno, void *dst, size_t nsecs) {
    assert(nsecs <= MAX_NSECS && VALID_IDE(ideno));
    assert(secno < MAX_DISK_NSECS && secno + nsecs <= MAX_DISK_NSECS);
    unsigned short iobase = IO_BASE(ideno), ioctrl = IO_CTRL(ideno);

    ide_wait_ready(iobase, 0);
```

```
    // generate interrupt
    outb(ioctrl + ISA_CTRL, 0);
    outb(iobase + ISA_SECCNT, nsecs);
    outb(iobase + ISA_SECTOR, secno & 0xFF);
    outb(iobase + ISA_CYL_LO, (secno >> 8) & 0xFF);
    outb(iobase + ISA_CYL_HI, (secno >> 16) & 0xFF);
    outb(iobase + ISA_SDH, 0xE0 | ((ideno & 1) << 4) | ((secno >> 24) & 0xF));
    outb(iobase + ISA_COMMAND, IDE_CMD_READ);

    int ret = 0;
    for (; nsecs > 0; nsecs --, dst += SECTSIZE) {
        if ((ret = ide_wait_ready(iobase, 1)) != 0) {
            goto out;
        }
        insl(iobase, dst, SECTSIZE / sizeof(uint32_t));
    }

out:
    return ret;
}

int
ide_write_secs(unsigned short ideno, uint32_t secno, const void *src, size_t nsecs) {
    assert(nsecs <= MAX_NSECS && VALID_IDE(ideno));
    assert(secno < MAX_DISK_NSECS && secno + nsecs <= MAX_DISK_NSECS);
    unsigned short iobase = IO_BASE(ideno), ioctrl = IO_CTRL(ideno);

    ide_wait_ready(iobase, 0);

    // generate interrupt
    outb(ioctrl + ISA_CTRL, 0);
    outb(iobase + ISA_SECCNT, nsecs);
    outb(iobase + ISA_SECTOR, secno & 0xFF);
    outb(iobase + ISA_CYL_LO, (secno >> 8) & 0xFF);
    outb(iobase + ISA_CYL_HI, (secno >> 16) & 0xFF);
    outb(iobase + ISA_SDH, 0xE0 | ((ideno & 1) << 4) | ((secno >> 24) & 0xF));
    outb(iobase + ISA_COMMAND, IDE_CMD_WRITE);

    int ret = 0;
    for (; nsecs > 0; nsecs --, src += SECTSIZE) {
        if ((ret = ide_wait_ready(iobase, 1)) != 0) {
            goto out;
        }
        outsl(iobase, src, SECTSIZE / sizeof(uint32_t));
    }

out:
    return ret;
}

[ucore/lab8_result/kern/driver/picirq.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <x86.h>
#include <picirq.h>

// I/O Addresses of the two programmable interrupt controllers
#define IO_PIC1             0x20    // Master (IRQs 0-7)
#define IO_PIC2             0xA0    // Slave (IRQs 8-15)

#define IRQ_SLAVE           2       // IRQ at which slave connects to master

// Current IRQ mask.
// Initial IRQ mask has interrupt 2 enabled (for slave 8259A).
static uint16_t irq_mask = 0xFFFF & ~(1 << IRQ_SLAVE);
static bool did_init = 0;
```

```
static void
pic_setmask(uint16_t mask) {
    irq_mask = mask;
    if (did_init) {
        outb(IO_PIC1 + 1, mask);
        outb(IO_PIC2 + 1, mask >> 8);
    }
}

void
pic_enable(unsigned int irq) {
    pic_setmask(irq_mask & ~(1 << irq));
}

/* pic_init - initialize the 8259A interrupt controllers */
void
pic_init(void) {
    did_init = 1;

    // mask all interrupts
    outb(IO_PIC1 + 1, 0xFF);
    outb(IO_PIC2 + 1, 0xFF);

    // Set up master (8259A-1)

    // ICW1:  0001g0hi
    //    g:  0 = edge triggering, 1 = level triggering
    //    h:  0 = cascaded PICs, 1 = master only
    //    i:  0 = no ICW4, 1 = ICW4 required
    outb(IO_PIC1, 0x11);

    // ICW2:  Vector offset
    outb(IO_PIC1 + 1, IRQ_OFFSET);

    // ICW3:  (master PIC) bit mask of IR lines connected to slaves
    //        (slave PIC) 3-bit # of slave's connection to master
    outb(IO_PIC1 + 1, 1 << IRQ_SLAVE);

    // ICW4:  000nbmap
    //    n:  1 = special fully nested mode
    //    b:  1 = buffered mode
    //    m:  0 = slave PIC, 1 = master PIC
    //        (ignored when b is 0, as the master/slave role
    //         can be hardwired).
    //    a:  1 = Automatic EOI mode
    //    p:  0 = MCS-80/85 mode, 1 = intel x86 mode
    outb(IO_PIC1 + 1, 0x3);

    // Set up slave (8259A-2)
    outb(IO_PIC2, 0x11);    // ICW1
    outb(IO_PIC2 + 1, IRQ_OFFSET + 8);  // ICW2
    outb(IO_PIC2 + 1, IRQ_SLAVE);       // ICW3
    // NB Automatic EOI mode doesn't tend to work on the slave.
    // Linux source code says it's "to be investigated".
    outb(IO_PIC2 + 1, 0x3);             // ICW4

    // OCW3:  0ef01prs
    //   ef:  0x = NOP, 10 = clear specific mask, 11 = set specific mask
    //    p:  0 = no polling, 1 = polling mode
    //   rs:  0x = NOP, 10 = read IRR, 11 = read ISR
    outb(IO_PIC1, 0x68);    // clear specific mask
    outb(IO_PIC1, 0x0a);    // read IRR by default

    outb(IO_PIC2, 0x68);    // OCW3
```

```
    outb(IO_PIC2, 0x0a);    // OCW3

    if (irq_mask != 0xFFFF) {
        pic_setmask(irq_mask);
    }
}
```

[ucore/lab8_result/kern/driver/picirq.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DRIVER_PICIRQ_H__
#define __KERN_DRIVER_PICIRQ_H__

void pic_init(void);
void pic_enable(unsigned int irq);

#define IRQ_OFFSET      32

#endif /* !__KERN_DRIVER_PICIRQ_H__ */
```

[ucore/lab8_result/kern/driver/kbdreg.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DRIVER_KBDREG_H__
#define __KERN_DRIVER_KBDREG_H__

// Special keycodes
#define KEY_HOME            0xE0
#define KEY_END             0xE1
#define KEY_UP              0xE2
#define KEY_DN              0xE3
#define KEY_LF              0xE4
#define KEY_RT              0xE5
#define KEY_PGUP            0xE6
#define KEY_PGDN            0xE7
#define KEY_INS             0xE8
#define KEY_DEL             0xE9


/* This is i8042reg.h + kbdreg.h from NetBSD. */

#define KBSTATP             0x64    // kbd controller status port(I)
#define KBS_DIB             0x01    // kbd data in buffer
#define KBS_IBF             0x02    // kbd input buffer low
#define KBS_WARM            0x04    // kbd input buffer low
#define BS_OCMD             0x08    // kbd output buffer has command
#define KBS_NOSEC           0x10    // kbd security lock not engaged
#define KBS_TERR            0x20    // kbd transmission error
#define KBS_RERR            0x40    // kbd receive error
#define KBS_PERR            0x80    // kbd parity error

#define KBCMDP              0x64    // kbd controller port(O)
#define KBC_RAMREAD         0x20    // read from RAM
#define KBC_RAMWRITE        0x60    // write to RAM
#define KBC_AUXDISABLE      0xa7    // disable auxiliary port
#define KBC_AUXENABLE       0xa8    // enable auxiliary port
#define KBC_AUXTEST         0xa9    // test auxiliary port
#define KBC_KBDECHO         0xd2    // echo to keyboard port
#define KBC_AUXECHO         0xd3    // echo to auxiliary port
#define KBC_AUXWRITE        0xd4    // write to auxiliary port
#define KBC_SELFTEST        0xaa    // start self-test
#define KBC_KBDTEST         0xab    // test keyboard port
#define KBC_KBDDISABLE      0xad    // disable keyboard port
#define KBC_KBDENABLE       0xae    // enable keyboard port
#define KBC_PULSE0          0xfe    // pulse output bit 0
#define KBC_PULSE1          0xfd    // pulse output bit 1
#define KBC_PULSE2          0xfb    // pulse output bit 2
#define KBC_PULSE3          0xf7    // pulse output bit 3
```

```
#define KBDATAP             0x60    // kbd data port(I)
#define KBOUTP              0x60    // kbd data port(O)

#define K_RDCMDBYTE         0x20
#define K_LDCMDBYTE         0x60

#define KC8_TRANS           0x40    // convert to old scan codes
#define KC8_MDISABLE        0x20    // disable mouse
#define KC8_KDISABLE        0x10    // disable keyboard
#define KC8_IGNSEC          0x08    // ignore security lock
#define KC8_CPU             0x04    // exit from protected mode reset
#define KC8_MENABLE         0x02    // enable mouse interrupt
#define KC8_KENABLE         0x01    // enable keyboard interrupt
#define CMDBYTE             (KC8_TRANS|KC8_CPU|KC8_MENABLE|KC8_KENABLE)

/* keyboard commands */
#define KBC_RESET           0xFF    // reset the keyboard
#define KBC_RESEND          0xFE    // request the keyboard resend the last byte
#define KBC_SETDEFAULT      0xF6    // resets keyboard to its power-on defaults
#define KBC_DISABLE         0xF5    // as per KBC_SETDEFAULT, but also disable key scanning
#define KBC_ENABLE          0xF4    // enable key scanning
#define KBC_TYPEMATIC       0xF3    // set typematic rate and delay
#define KBC_SETTABLE        0xF0    // set scancode translation table
#define KBC_MODEIND         0xED    // set mode indicators(i.e. LEDs)
#define KBC_ECHO            0xEE    // request an echo from the keyboard

/* keyboard responses */
#define KBR_EXTENDED        0xE0    // extended key sequence
#define KBR_RESEND          0xFE    // needs resend of command
#define KBR_ACK             0xFA    // received a valid command
#define KBR_OVERRUN         0x00    // flooded
#define KBR_FAILURE         0xFD    // diagnosic failure
#define KBR_BREAK           0xF0    // break code prefix - sent on key release
#define KBR_RSTDONE         0xAA    // reset complete
#define KBR_ECHO            0xEE    // echo response

#endif /* !__KERN_DRIVER_KBDREG_H__ */

[ucore/lab8_result/kern/driver/intr.c] ++++++++++++++++++++++++++++++
#include <x86.h>
#include <intr.h>

/* intr_enable - enable irq interrupt */
void
intr_enable(void) {
    sti();
}

/* intr_disable - disable irq interrupt */
void
intr_disable(void) {
    cli();
}

[ucore/lab8_result/kern/driver/clock.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_DRIVER_CLOCK_H__
#define __KERN_DRIVER_CLOCK_H__

#include <defs.h>

extern volatile size_t ticks;

void clock_init(void);

long SYSTEM_READ_TIMER( void );
```

```
#endif /* !__KERN_DRIVER_CLOCK_H__ */

[ucore/lab8_result/kern/driver/intr.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_DRIVER_INTR_H__
#define __KERN_DRIVER_INTR_H__

void intr_enable(void);
void intr_disable(void);

#endif /* !__KERN_DRIVER_INTR_H__ */

[ucore/lab8_result/kern/driver/console.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <x86.h>
#include <stdio.h>
#include <string.h>
#include <kbdreg.h>
#include <picirq.h>
#include <trap.h>
#include <memlayout.h>
#include <sync.h>

/* stupid I/O delay routine necessitated by historical PC design flaws */
static void
delay(void) {
    inb(0x84);
    inb(0x84);
    inb(0x84);
    inb(0x84);
}

/***** Serial I/O code *****/
#define COM1            0x3F8

#define COM_RX          0       // In:  Receive buffer (DLAB=0)
#define COM_TX          0       // Out: Transmit buffer (DLAB=0)
#define COM_DLL         0       // Out: Divisor Latch Low (DLAB=1)
#define COM_DLM         1       // Out: Divisor Latch High (DLAB=1)
#define COM_IER         1       // Out: Interrupt Enable Register
#define COM_IER_RDI     0x01    // Enable receiver data interrupt
#define COM_IIR         2       // In:  Interrupt ID Register
#define COM_FCR         2       // Out: FIFO Control Register
#define COM_LCR         3       // Out: Line Control Register
#define COM_LCR_DLAB    0x80    // Divisor latch access bit
#define COM_LCR_WLEN8   0x03    // Wordlength: 8 bits
#define COM_MCR         4       // Out: Modem Control Register
#define COM_MCR_RTS     0x02    // RTS complement
#define COM_MCR_DTR     0x01    // DTR complement
#define COM_MCR_OUT2    0x08    // Out2 complement
#define COM_LSR         5       // In:  Line Status Register
#define COM_LSR_DATA    0x01    // Data available
#define COM_LSR_TXRDY   0x20    // Transmit buffer avail
#define COM_LSR_TSRE    0x40    // Transmitter off

#define MONO_BASE       0x3B4
#define MONO_BUF        0xB0000
#define CGA_BASE        0x3D4
#define CGA_BUF         0xB8000
#define CRT_ROWS        25
#define CRT_COLS        80
#define CRT_SIZE        (CRT_ROWS * CRT_COLS)

#define LPTPORT         0x378
```

```
static uint16_t *crt_buf;
static uint16_t crt_pos;
static uint16_t addr_6845;

/* TEXT-mode CGA/VGA display output */

static void
cga_init(void) {
    volatile uint16_t *cp = (uint16_t *)(CGA_BUF + KERNBASE);
    uint16_t was = *cp;
    *cp = (uint16_t) 0xA55A;
    if (*cp != 0xA55A) {
        cp = (uint16_t*)(MONO_BUF + KERNBASE);
        addr_6845 = MONO_BASE;
    } else {
        *cp = was;
        addr_6845 = CGA_BASE;
    }

    // Extract cursor location
    uint32_t pos;
    outb(addr_6845, 14);
    pos = inb(addr_6845 + 1) << 8;
    outb(addr_6845, 15);
    pos |= inb(addr_6845 + 1);

    crt_buf = (uint16_t*) cp;
    crt_pos = pos;
}

static bool serial_exists = 0;

static void
serial_init(void) {
    // Turn off the FIFO
    outb(COM1 + COM_FCR, 0);

    // Set speed; requires DLAB latch
    outb(COM1 + COM_LCR, COM_LCR_DLAB);
    outb(COM1 + COM_DLL, (uint8_t) (115200 / 9600));
    outb(COM1 + COM_DLM, 0);

    // 8 data bits, 1 stop bit, parity off; turn off DLAB latch
    outb(COM1 + COM_LCR, COM_LCR_WLEN8 & ~COM_LCR_DLAB);

    // No modem controls
    outb(COM1 + COM_MCR, 0);
    // Enable rcv interrupts
    outb(COM1 + COM_IER, COM_IER_RDI);

    // Clear any preexisting overrun indications and interrupts
    // Serial port doesn't exist if COM_LSR returns 0xFF
    serial_exists = (inb(COM1 + COM_LSR) != 0xFF);
    (void) inb(COM1+COM_IIR);
    (void) inb(COM1+COM_RX);

    if (serial_exists) {
        pic_enable(IRQ_COM1);
    }
}

static void
lpt_putc_sub(int c) {
    int i;
```

```
    for (i = 0; !(inb(LPTPORT + 1) & 0x80) && i < 12800; i ++) {
        delay();
    }
    outb(LPTPORT + 0, c);
    outb(LPTPORT + 2, 0x08 | 0x04 | 0x01);
    outb(LPTPORT + 2, 0x08);
}

/* lpt_putc - copy console output to parallel port */
static void
lpt_putc(int c) {
    if (c != '\b') {
        lpt_putc_sub(c);
    }
    else {
        lpt_putc_sub('\b');
        lpt_putc_sub(' ');
        lpt_putc_sub('\b');
    }
}

/* cga_putc - print character to console */
static void
cga_putc(int c) {
    // set black on white
    if (!(c & ~0xFF)) {
        c |= 0x0700;
    }

    switch (c & 0xff) {
    case '\b':
        if (crt_pos > 0) {
            crt_pos --;
            crt_buf[crt_pos] = (c & ~0xff) | ' ';
        }
        break;
    case '\n':
        crt_pos += CRT_COLS;
    case '\r':
        crt_pos -= (crt_pos % CRT_COLS);
        break;
    default:
        crt_buf[crt_pos ++] = c;     // write the character
        break;
    }

    // What is the purpose of this?
    if (crt_pos >= CRT_SIZE) {
        int i;
        memmove(crt_buf, crt_buf + CRT_COLS, (CRT_SIZE - CRT_COLS) * sizeof(uint16_t));
        for (i = CRT_SIZE - CRT_COLS; i < CRT_SIZE; i ++) {
            crt_buf[i] = 0x0700 | ' ';
        }
        crt_pos -= CRT_COLS;
    }

    // move that little blinky thing
    outb(addr_6845, 14);
    outb(addr_6845 + 1, crt_pos >> 8);
    outb(addr_6845, 15);
    outb(addr_6845 + 1, crt_pos);
}

static void
serial_putc_sub(int c) {
```

```
    int i;
    for (i = 0; !(inb(COM1 + COM_LSR) & COM_LSR_TXRDY) && i < 12800; i ++) {
        delay();
    }
    outb(COM1 + COM_TX, c);
}

/* serial_putc - print character to serial port */
static void
serial_putc(int c) {
    if (c != '\b') {
        serial_putc_sub(c);
    }
    else {
        serial_putc_sub('\b');
        serial_putc_sub(' ');
        serial_putc_sub('\b');
    }
}

/* *
 * Here we manage the console input buffer, where we stash characters
 * received from the keyboard or serial port whenever the corresponding
 * interrupt occurs.
 * */

#define CONSBUFSIZE 512

static struct {
    uint8_t buf[CONSBUFSIZE];
    uint32_t rpos;
    uint32_t wpos;
} cons;

/* *
 * cons_intr - called by device interrupt routines to feed input
 * characters into the circular console input buffer.
 * */
static void
cons_intr(int (*proc)(void)) {
    int c;
    while ((c = (*proc)()) != -1) {
        if (c != 0) {
            cons.buf[cons.wpos ++] = c;
            if (cons.wpos == CONSBUFSIZE) {
                cons.wpos = 0;
            }
        }
    }
}

/* serial_proc_data - get data from serial port */
static int
serial_proc_data(void) {
    if (!(inb(COM1 + COM_LSR) & COM_LSR_DATA)) {
        return -1;
    }
    int c = inb(COM1 + COM_RX);
    if (c == 127) {
        c = '\b';
    }
    return c;
}

/* serial_intr - try to feed input characters from serial port */
```

```
void
serial_intr(void) {
    if (serial_exists) {
        cons_intr(serial_proc_data);
    }
}

/***** Keyboard input code *****/

#define NO              0

#define SHIFT           (1<<0)
#define CTL             (1<<1)
#define ALT             (1<<2)

#define CAPSLOCK        (1<<3)
#define NUMLOCK         (1<<4)
#define SCROLLLOCK      (1<<5)

#define E0ESC           (1<<6)

static uint8_t shiftcode[256] = {
    [0x1D] CTL,
    [0x2A] SHIFT,
    [0x36] SHIFT,
    [0x38] ALT,
    [0x9D] CTL,
    [0xB8] ALT
};

static uint8_t togglecode[256] = {
    [0x3A] CAPSLOCK,
    [0x45] NUMLOCK,
    [0x46] SCROLLLOCK
};

static uint8_t normalmap[256] = {
    NO,   0x1B, '1',  '2',  '3',  '4',  '5',  '6',  // 0x00
    '7',  '8',  '9',  '0',  '-',  '=',  '\b', '\t',
    'q',  'w',  'e',  'r',  't',  'y',  'u',  'i',  // 0x10
    'o',  'p',  '[',  ']',  '\n', NO,   'a',  's',
    'd',  'f',  'g',  'h',  'j',  'k',  'l',  ';',  // 0x20
    '\'', '`',  NO,   '\\', 'z',  'x',  'c',  'v',
    'b',  'n',  'm',  ',',  '.',  '/',  NO,   '*',  // 0x30
    NO,   ' ',  NO,   NO,   NO,   NO,   NO,   NO,
    NO,   NO,   NO,   NO,   NO,   NO,   NO,   '7',  // 0x40
    '8',  '9',  '-',  '4',  '5',  '6',  '+',  '1',
    '2',  '3',  '0',  '.',  NO,   NO,   NO,   NO,   // 0x50
    [0xC7] KEY_HOME,    [0x9C] '\n' /*KP_Enter*/,
    [0xB5] '/' /*KP_Div*/,  [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,    [0xCB] KEY_LF,
    [0xCD] KEY_RT,      [0xCF] KEY_END,
    [0xD0] KEY_DN,      [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,     [0xD3] KEY_DEL
};

static uint8_t shiftmap[256] = {
    NO,   033,  '!',  '@',  '#',  '$',  '%',  '^',  // 0x00
    '&',  '*',  '(',  ')',  '_',  '+',  '\b', '\t',
    'Q',  'W',  'E',  'R',  'T',  'Y',  'U',  'I',  // 0x10
    'O',  'P',  '{',  '}',  '\n', NO,   'A',  'S',
    'D',  'F',  'G',  'H',  'J',  'K',  'L',  ':',  // 0x20
    '"',  '~',  NO,   '|',  'Z',  'X',  'C',  'V',
    'B',  'N',  'M',  '<',  '>',  '?',  NO,   '*',  // 0x30
    NO,   ' ',  NO,   NO,   NO,   NO,   NO,   NO,
```

```
    NO,   NO,   NO,   NO,   NO,   NO,   NO,   '7',  // 0x40
    '8',  '9',  '-',  '4',  '5',  '6',  '+',  '1',
    '2',  '3',  '0',  '.',  NO,   NO,   NO,   NO,   // 0x50
    [0xC7] KEY_HOME,    [0x9C] '\n' /*KP_Enter*/,
    [0xB5] '/' /*KP_Div*/,  [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,    [0xCB] KEY_LF,
    [0xCD] KEY_RT,      [0xCF] KEY_END,
    [0xD0] KEY_DN,      [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,     [0xD3] KEY_DEL
};

#define C(x) (x - '@')

static uint8_t ctlmap[256] = {
    NO,      NO,      NO,      NO,      NO,      NO,      NO,      NO,
    NO,      NO,      NO,      NO,      NO,      NO,      NO,      NO,
    C('Q'),  C('W'),  C('E'),  C('R'),  C('T'),  C('Y'),  C('U'),  C('I'),
    C('O'),  C('P'),  NO,      NO,      '\r',    NO,      C('A'),  C('S'),
    C('D'),  C('F'),  C('G'),  C('H'),  C('J'),  C('K'),  C('L'),  NO,
    NO,      NO,      NO,      C('\\'), C('Z'),  C('X'),  C('C'),  C('V'),
    C('B'),  C('N'),  C('M'),  NO,      NO,      C('/'),  NO,      NO,
    [0x97] KEY_HOME,
    [0xB5] C('/'),      [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,    [0xCB] KEY_LF,
    [0xCD] KEY_RT,      [0xCF] KEY_END,
    [0xD0] KEY_DN,      [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,     [0xD3] KEY_DEL
};

static uint8_t *charcode[4] = {
    normalmap,
    shiftmap,
    ctlmap,
    ctlmap
};

/* *
 * kbd_proc_data - get data from keyboard
 *
 * The kbd_proc_data() function gets data from the keyboard.
 * If we finish a character, return it, else 0. And return -1 if no data.
 * */
static int
kbd_proc_data(void) {
    int c;
    uint8_t data;
    static uint32_t shift;

    if ((inb(KBSTATP) & KBS_DIB) == 0) {
        return -1;
    }

    data = inb(KBDATAP);

    if (data == 0xE0) {
        // E0 escape character
        shift |= E0ESC;
        return 0;
    } else if (data & 0x80) {
        // Key released
        data = (shift & E0ESC ? data : data & 0x7F);
        shift &= ~(shiftcode[data] | E0ESC);
        return 0;
    } else if (shift & E0ESC) {
        // Last character was an E0 escape; or with 0x80
```

```
        data |= 0x80;
        shift &= ~E0ESC;
    }

    shift |= shiftcode[data];
    shift ^= togglecode[data];

    c = charcode[shift & (CTL | SHIFT)][data];
    if (shift & CAPSLOCK) {
        if ('a' <= c && c <= 'z')
            c += 'A' - 'a';
        else if ('A' <= c && c <= 'Z')
            c += 'a' - 'A';
    }

    // Process special keys
    // Ctrl-Alt-Del: reboot
    if (!(~shift & (CTL | ALT)) && c == KEY_DEL) {
        cprintf("Rebooting!\n");
        outb(0x92, 0x3); // courtesy of Chris Frost
    }
    return c;
}

/* kbd_intr - try to feed input characters from keyboard */
static void
kbd_intr(void) {
    cons_intr(kbd_proc_data);
}

static void
kbd_init(void) {
    // drain the kbd buffer
    kbd_intr();
    pic_enable(IRQ_KBD);
}

/* cons_init - initializes the console devices */
void
cons_init(void) {
    cga_init();
    serial_init();
    kbd_init();
    if (!serial_exists) {
        cprintf("serial port does not exist!!\n");
    }
}

/* cons_putc - print a single character @c to console devices */
void
cons_putc(int c) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        lpt_putc(c);
        cga_putc(c);
        serial_putc(c);
    }
    local_intr_restore(intr_flag);
}

/* *
 * cons_getc - return the next input character from console,
 * or 0 if none waiting.
 * */
```

```
int
cons_getc(void) {
    int c = 0;
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        // poll for any pending input characters,
        // so that this function works even when interrupts are disabled
        // (e.g., when called from the kernel monitor).
        serial_intr();
        kbd_intr();

        // grab the next character from the input buffer.
        if (cons.rpos != cons.wpos) {
            c = cons.buf[cons.rpos ++];
            if (cons.rpos == CONSBUFSIZE) {
                cons.rpos = 0;
            }
        }
    }
    local_intr_restore(intr_flag);
    return c;
}
```

[ucore/lab8_result/kern/driver/ide.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DRIVER_IDE_H__
#define __KERN_DRIVER_IDE_H__

#include <defs.h>

void ide_init(void);
bool ide_device_valid(unsigned short ideno);
size_t ide_device_size(unsigned short ideno);

int ide_read_secs(unsigned short ideno, uint32_t secno, void *dst, size_t nsecs);
int ide_write_secs(unsigned short ideno, uint32_t secno, const void *src, size_t nsecs);

#endif /* !__KERN_DRIVER_IDE_H__ */
```

[ucore/lab8_result/kern/process/entry.S] ++++++++++++++++++++++++++++++

```
.text
.globl kernel_thread_entry
kernel_thread_entry:        # void kernel_thread(void)

    pushl %edx              # push arg
    call *%ebx              # call fn

    pushl %eax              # save the return value of fn(arg)
    call do_exit            # call do_exit to terminate current thread
```

[ucore/lab8_result/kern/process/proc.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_PROCESS_PROC_H__
#define __KERN_PROCESS_PROC_H__

#include <defs.h>
#include <list.h>
#include <trap.h>
#include <memlayout.h>
#include <skew_heap.h>


// process's state in his life cycle
enum proc_state {
    PROC_UNINIT = 0,  // uninitialized
    PROC_SLEEPING,    // sleeping
```

```
    PROC_RUNNABLE,    // runnable(maybe running)
    PROC_ZOMBIE,      // almost dead, and wait parent proc to reclaim his resource
};

// Saved registers for kernel context switches.
// Don't need to save all the %fs etc. segment registers,
// because they are constant across kernel contexts.
// Save all the regular registers so we don't need to care
// which are caller save, but not the return register %eax.
// (Not saving %eax just simplifies the switching code.)
// The layout of context must match code in switch.S.
struct context {
    uint32_t eip;
    uint32_t esp;
    uint32_t ebx;
    uint32_t ecx;
    uint32_t edx;
    uint32_t esi;
    uint32_t edi;
    uint32_t ebp;
};

#define PROC_NAME_LEN               50
#define MAX_PROCESS                 4096
#define MAX_PID                     (MAX_PROCESS * 2)

extern list_entry_t proc_list;

struct inode;

struct proc_struct {
    enum proc_state state;                      // Process state
    int pid;                                    // Process ID
    int runs;                                   // the running times of Proces
    uintptr_t kstack;                           // Process kernel stack
    volatile bool need_resched;                 // bool value: need to be rescheduled to relea
se CPU?
    struct proc_struct *parent;                 // the parent process
    struct mm_struct *mm;                       // Process's memory management field
    struct context context;                     // Switch here to run process
    struct trapframe *tf;                       // Trap frame for current interrupt
    uintptr_t cr3;                              // CR3 register: the base addr of Page Directr
oy Table(PDT)
    uint32_t flags;                             // Process flag
    char name[PROC_NAME_LEN + 1];               // Process name
    list_entry_t list_link;                     // Process link list
    list_entry_t hash_link;                     // Process hash list
    int exit_code;                              // exit code (be sent to parent proc)
    uint32_t wait_state;                        // waiting state
    struct proc_struct *cptr, *yptr, *optr;     // relations between processes
    struct run_queue *rq;                       // running queue contains Process
    list_entry_t run_link;                      // the entry linked in run queue
    int time_slice;                             // time slice for occupying the CPU
    skew_heap_entry_t lab6_run_pool;            // FOR LAB6 ONLY: the entry in the run pool
    uint32_t lab6_stride;                       // FOR LAB6 ONLY: the current stride of the pr
ocess
    uint32_t lab6_priority;                     // FOR LAB6 ONLY: the priority of process, set
 by lab6_set_priority(uint32_t)
    struct files_struct *filesp;                // the file related info(pwd, files_count, fil
es_array, fs_semaphore) of process
};

#define PF_EXITING                  0x00000001      // getting shutdown

#define WT_INTERRUPTED               0x80000000                    // the wait state could be
```

```
interrupted
#define WT_CHILD                    (0x00000001 | WT_INTERRUPTED)  // wait child process
#define WT_KSEM                      0x00000100                    // wait kernel semaphore
#define WT_TIMER                    (0x00000002 | WT_INTERRUPTED)  // wait timer
#define WT_KBD                      (0x00000004 | WT_INTERRUPTED)  // wait the input of keyboa
rd

#define le2proc(le, member)         \
    to_struct((le), struct proc_struct, member)

extern struct proc_struct *idleproc, *initproc, *current;

void proc_init(void);
void proc_run(struct proc_struct *proc);
int kernel_thread(int (*fn)(void *), void *arg, uint32_t clone_flags);

char *set_proc_name(struct proc_struct *proc, const char *name);
char *get_proc_name(struct proc_struct *proc);
void cpu_idle(void) __attribute__((noreturn));

struct proc_struct *find_proc(int pid);
int do_fork(uint32_t clone_flags, uintptr_t stack, struct trapframe *tf);
int do_exit(int error_code);
int do_yield(void);
int do_execve(const char *name, int argc, const char **argv);
int do_wait(int pid, int *code_store);
int do_kill(int pid);
//FOR LAB6, set the process's priority (bigger value will get more CPU time)
void lab6_set_priority(uint32_t priority);
int do_sleep(unsigned int time);
#endif /* !__KERN_PROCESS_PROC_H__ */

[ucore/lab8_result/kern/process/proc.c] ++++++++++++++++++++++++++++++
#include <proc.h>
#include <kmalloc.h>
#include <string.h>
#include <sync.h>
#include <pmm.h>
#include <error.h>
#include <sched.h>
#include <elf.h>
#include <vmm.h>
#include <trap.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
#include <fs.h>
#include <vfs.h>
#include <sysfile.h>

/* ------------- process/thread mechanism design&implementation -------------
(an simplified Linux process/thread mechanism )
introduction:
  ucore implements a simple process/thread mechanism. process contains the independent memory
sapce, at least one threads
for execution, the kernel data(for management), processor state (for context switch), files(in
 lab6), etc. ucore needs to
manage all these details efficiently. In ucore, a thread is just a special kind of process(sha
re process's memory).
------------------------------
process state       :     meaning               -- reason
    PROC_UNINIT     :   uninitialized           -- alloc_proc
    PROC_SLEEPING   :   sleeping                -- try_free_pages, do_wait, do_sleep
    PROC_RUNNABLE   :   runnable(maybe running) -- proc_init, wakeup_proc,
```

```
    PROC_ZOMBIE     :   almost dead             -- do_exit

-----------------------------
process state changing:

  alloc_proc                                  RUNNING
      +                                   +--<----<--+
      +                                   + proc_run +
      V                                   +-->---->--+
PROC_UNINIT -- proc_init/wakeup_proc --> PROC_RUNNABLE -- try_free_pages/do_wait/do_sleep -->
PROC_SLEEPING --
                                           A      +
                +
                                           |      +--- do_exit --> PROC_ZOMBIE
                +
                                           +
                +
                                           ------------------------wakeup_proc----------------
-----------------
-----------------------------
process relations
parent:           proc->parent  (proc is children)
children:         proc->cptr    (proc is parent)
older sibling:    proc->optr    (proc is younger sibling)
younger sibling:  proc->yptr    (proc is older sibling)
-----------------------------
related syscall for process:
SYS_exit        : process exit,                           -->do_exit
SYS_fork        : create child process, dup mm            -->do_fork-->wakeup_proc
SYS_wait        : wait process                            -->do_wait
SYS_exec        : after fork, process execute a program   -->load a program and refresh the mm
SYS_clone       : create child thread                     -->do_fork-->wakeup_proc
SYS_yield       : process flag itself need resecheduling, -- proc->need_sched=1, then schedule
r will rescheule this process
SYS_sleep       : process sleep                           -->do_sleep
SYS_kill        : kill process                            -->do_kill-->proc->flags |= PF_EXITI
NG
                                                                 -->wakeup_proc-->do_wait-->do
_exit
SYS_getpid      : get the process's pid

*/

// the process set's list
list_entry_t proc_list;

#define HASH_SHIFT          10
#define HASH_LIST_SIZE      (1 << HASH_SHIFT)
#define pid_hashfn(x)       (hash32(x, HASH_SHIFT))

// has list for process set based on pid
static list_entry_t hash_list[HASH_LIST_SIZE];

// idle proc
struct proc_struct *idleproc = NULL;
// init proc
struct proc_struct *initproc = NULL;
// current proc
struct proc_struct *current = NULL;

static int nr_process = 0;

void kernel_thread_entry(void);
void forkrets(struct trapframe *tf);
void switch_to(struct context *from, struct context *to);
```

```
// alloc_proc - alloc a proc_struct and init all fields of proc_struct
static struct proc_struct *
alloc_proc(void) {
    struct proc_struct *proc = kmalloc(sizeof(struct proc_struct));
    if (proc != NULL) {
    //LAB4:EXERCISE1 YOUR CODE
    /*
     * below fields in proc_struct need to be initialized
     *       enum proc_state state;                      // Process state
     *       int pid;                                    // Process ID
     *       int runs;                                   // the running times of Proces
     *       uintptr_t kstack;                           // Process kernel stack
     *       volatile bool need_resched;                 // bool value: need to be rescheduled
 to release CPU?
     *       struct proc_struct *parent;                 // the parent process
     *       struct mm_struct *mm;                       // Process's memory management field
     *       struct context context;                     // Switch here to run process
     *       struct trapframe *tf;                       // Trap frame for current interrupt
     *       uintptr_t cr3;                              // CR3 register: the base addr of Pag
e Directroy Table(PDT)
     *       uint32_t flags;                             // Process flag
     *       char name[PROC_NAME_LEN + 1];               // Process name
     */
     //LAB5 YOUR CODE : (update LAB4 steps)
    /*
     * below fields(add in LAB5) in proc_struct need to be initialized
     *       uint32_t wait_state;                        // waiting state
     *       struct proc_struct *cptr, *yptr, *optr;     // relations between processes
         */
    //LAB8:EXERCISE2 YOUR CODE HINT:need add some code to init fs in proc_struct, ...
        proc->state = PROC_UNINIT;
        proc->pid = -1;
        proc->runs = 0;
        proc->kstack = 0;
        proc->need_resched = 0;
        proc->parent = NULL;
        proc->mm = NULL;
        memset(&(proc->context), 0, sizeof(struct context));
        proc->tf = NULL;
        proc->cr3 = boot_cr3;
        proc->flags = 0;
        memset(proc->name, 0, PROC_NAME_LEN);
        proc->wait_state = 0;
        proc->cptr = proc->optr = proc->yptr = NULL;
        proc->rq = NULL;
        list_init(&(proc->run_link));
        proc->time_slice = 0;
        proc->lab6_run_pool.left = proc->lab6_run_pool.right = proc->lab6_run_pool.parent = NU
LL;
        proc->lab6_stride = 0;
        proc->lab6_priority = 0;
        proc->filesp = NULL;
    }
    return proc;
}

// set_proc_name - set the name of proc
char *
set_proc_name(struct proc_struct *proc, const char *name) {
    memset(proc->name, 0, sizeof(proc->name));
    return memcpy(proc->name, name, PROC_NAME_LEN);
}

// get_proc_name - get the name of proc
```

```
char *
get_proc_name(struct proc_struct *proc) {
    static char name[PROC_NAME_LEN + 1];
    memset(name, 0, sizeof(name));
    return memcpy(name, proc->name, PROC_NAME_LEN);
}

// set_links - set the relation links of process
static void
set_links(struct proc_struct *proc) {
    list_add(&proc_list, &(proc->list_link));
    proc->yptr = NULL;
    if ((proc->optr = proc->parent->cptr) != NULL) {
        proc->optr->yptr = proc;
    }
    proc->parent->cptr = proc;
    nr_process ++;
}

// remove_links - clean the relation links of process
static void
remove_links(struct proc_struct *proc) {
    list_del(&(proc->list_link));
    if (proc->optr != NULL) {
        proc->optr->yptr = proc->yptr;
    }
    if (proc->yptr != NULL) {
        proc->yptr->optr = proc->optr;
    }
    else {
       proc->parent->cptr = proc->optr;
    }
    nr_process --;
}

// get_pid - alloc a unique pid for process
static int
get_pid(void) {
    static_assert(MAX_PID > MAX_PROCESS);
    struct proc_struct *proc;
    list_entry_t *list = &proc_list, *le;
    static int next_safe = MAX_PID, last_pid = MAX_PID;
    if (++ last_pid >= MAX_PID) {
        last_pid = 1;
        goto inside;
    }
    if (last_pid >= next_safe) {
    inside:
        next_safe = MAX_PID;
    repeat:
        le = list;
        while ((le = list_next(le)) != list) {
            proc = le2proc(le, list_link);
            if (proc->pid == last_pid) {
                if (++ last_pid >= next_safe) {
                    if (last_pid >= MAX_PID) {
                        last_pid = 1;
                    }
                    next_safe = MAX_PID;
                    goto repeat;
                }
            }
            else if (proc->pid > last_pid && next_safe > proc->pid) {
                next_safe = proc->pid;
            }
```

```
        }
    }
    return last_pid;
}

// proc_run - make process "proc" running on cpu
// NOTE: before call switch_to, should load  base addr of "proc"'s new PDT
void
proc_run(struct proc_struct *proc) {
    if (proc != current) {
        bool intr_flag;
        struct proc_struct *prev = current, *next = proc;
        local_intr_save(intr_flag);
        {
            current = proc;
            load_esp0(next->kstack + KSTACKSIZE);
            lcr3(next->cr3);
            switch_to(&(prev->context), &(next->context));
        }
        local_intr_restore(intr_flag);
    }
}

// forkret -- the first kernel entry point of a new thread/process
// NOTE: the addr of forkret is setted in copy_thread function
//       after switch_to, the current proc will execute here.
static void
forkret(void) {
    forkrets(current->tf);
}

// hash_proc - add proc into proc hash_list
static void
hash_proc(struct proc_struct *proc) {
    list_add(hash_list + pid_hashfn(proc->pid), &(proc->hash_link));
}

// unhash_proc - delete proc from proc hash_list
static void
unhash_proc(struct proc_struct *proc) {
    list_del(&(proc->hash_link));
}

// find_proc - find proc frome proc hash_list according to pid
struct proc_struct *
find_proc(int pid) {
    if (0 < pid && pid < MAX_PID) {
        list_entry_t *list = hash_list + pid_hashfn(pid), *le = list;
        while ((le = list_next(le)) != list) {
            struct proc_struct *proc = le2proc(le, hash_link);
            if (proc->pid == pid) {
                return proc;
            }
        }
    }
    return NULL;
}

// kernel_thread - create a kernel thread using "fn" function
// NOTE: the contents of temp trapframe tf will be copied to
//       proc->tf in do_fork-->copy_thread function
int
kernel_thread(int (*fn)(void *), void *arg, uint32_t clone_flags) {
    struct trapframe tf;
    memset(&tf, 0, sizeof(struct trapframe));
```

```
    tf.tf_cs = KERNEL_CS;
    tf.tf_ds = tf.tf_es = tf.tf_ss = KERNEL_DS;
    tf.tf_regs.reg_ebx = (uint32_t)fn;
    tf.tf_regs.reg_edx = (uint32_t)arg;
    tf.tf_eip = (uint32_t)kernel_thread_entry;
    return do_fork(clone_flags | CLONE_VM, 0, &tf);
}

// setup_kstack - alloc pages with size KSTACKPAGE as process kernel stack
static int
setup_kstack(struct proc_struct *proc) {
    struct Page *page = alloc_pages(KSTACKPAGE);
    if (page != NULL) {
        proc->kstack = (uintptr_t)page2kva(page);
        return 0;
    }
    return -E_NO_MEM;
}

// put_kstack - free the memory space of process kernel stack
static void
put_kstack(struct proc_struct *proc) {
    free_pages(kva2page((void *)(proc->kstack)), KSTACKPAGE);
}

// setup_pgdir - alloc one page as PDT
static int
setup_pgdir(struct mm_struct *mm) {
    struct Page *page;
    if ((page = alloc_page()) == NULL) {
        return -E_NO_MEM;
    }
    pde_t *pgdir = page2kva(page);
    memcpy(pgdir, boot_pgdir, PGSIZE);
    pgdir[PDX(VPT)] = PADDR(pgdir) | PTE_P | PTE_W;
    mm->pgdir = pgdir;
    return 0;
}

// put_pgdir - free the memory space of PDT
static void
put_pgdir(struct mm_struct *mm) {
    free_page(kva2page(mm->pgdir));
}

// copy_mm - process "proc" duplicate OR share process "current"'s mm according clone_flags
//         - if clone_flags & CLONE_VM, then "share" ; else "duplicate"
static int
copy_mm(uint32_t clone_flags, struct proc_struct *proc) {
    struct mm_struct *mm, *oldmm = current->mm;

    /* current is a kernel thread */
    if (oldmm == NULL) {
        return 0;
    }
    if (clone_flags & CLONE_VM) {
        mm = oldmm;
        goto good_mm;
    }

    int ret = -E_NO_MEM;
    if ((mm = mm_create()) == NULL) {
        goto bad_mm;
    }
    if (setup_pgdir(mm) != 0) {
```

```
        goto bad_pgdir_cleanup_mm;
    }

    lock_mm(oldmm);
    {
        ret = dup_mmap(mm, oldmm);
    }
    unlock_mm(oldmm);

    if (ret != 0) {
        goto bad_dup_cleanup_mmap;
    }

good_mm:
    mm_count_inc(mm);
    proc->mm = mm;
    proc->cr3 = PADDR(mm->pgdir);
    return 0;
bad_dup_cleanup_mmap:
    exit_mmap(mm);
    put_pgdir(mm);
bad_pgdir_cleanup_mm:
    mm_destroy(mm);
bad_mm:
    return ret;
}

// copy_thread - setup the trapframe on the  process's kernel stack top and
//             - setup the kernel entry point and stack of process
static void
copy_thread(struct proc_struct *proc, uintptr_t esp, struct trapframe *tf) {
    proc->tf = (struct trapframe *)(proc->kstack + KSTACKSIZE) - 1;
    *(proc->tf) = *tf;
    proc->tf->tf_regs.reg_eax = 0;
    proc->tf->tf_esp = esp;
    proc->tf->tf_eflags |= FL_IF;

    proc->context.eip = (uintptr_t)forkret;
    proc->context.esp = (uintptr_t)(proc->tf);
}

//copy_fs&put_fs function used by do_fork in LAB8
static int
copy_fs(uint32_t clone_flags, struct proc_struct *proc) {
    struct files_struct *filesp, *old_filesp = current->filesp;
    assert(old_filesp != NULL);

    if (clone_flags & CLONE_FS) {
        filesp = old_filesp;
        goto good_files_struct;
    }

    int ret = -E_NO_MEM;
    if ((filesp = files_create()) == NULL) {
        goto bad_files_struct;
    }

    if ((ret = dup_files(filesp, old_filesp)) != 0) {
        goto bad_dup_cleanup_fs;
    }

good_files_struct:
    files_count_inc(filesp);
    proc->filesp = filesp;
    return 0;
```

```
bad_dup_cleanup_fs:
    files_destroy(filesp);
bad_files_struct:
    return ret;
}

static void
put_fs(struct proc_struct *proc) {
    struct files_struct *filesp = proc->filesp;
    if (filesp != NULL) {
        if (files_count_dec(filesp) == 0) {
            files_destroy(filesp);
        }
    }
}

/* do_fork -     parent process for a new child process
 * @clone_flags: used to guide how to clone the child process
 * @stack:       the parent's user stack pointer. if stack==0, It means to fork a kernel threa
d.
 * @tf:          the trapframe info, which will be copied to child process's proc->tf
 */
int
do_fork(uint32_t clone_flags, uintptr_t stack, struct trapframe *tf) {
    int ret = -E_NO_FREE_PROC;
    struct proc_struct *proc;
    if (nr_process >= MAX_PROCESS) {
        goto fork_out;
    }
    ret = -E_NO_MEM;
    //LAB4:EXERCISE2 YOUR CODE
    //LAB8:EXERCISE2 YOUR CODE  HINT:how to copy the fs in parent's proc_struct?
    /*
     * Some Useful MACROs, Functions and DEFINEs, you can use them in below implementation.
     * MACROs or Functions:
     *   alloc_proc:   create a proc struct and init fields (lab4:exercise1)
     *   setup_kstack: alloc pages with size KSTACKPAGE as process kernel stack
     *   copy_mm:      process "proc" duplicate OR share process "current"'s mm according clon
e_flags
     *                 if clone_flags & CLONE_VM, then "share" ; else "duplicate"
     *   copy_thread:  setup the trapframe on the  process's kernel stack top and
     *                 setup the kernel entry point and stack of process
     *   hash_proc:    add proc into proc hash_list
     *   get_pid:      alloc a unique pid for process
     *   wakeup_proc:  set proc->state = PROC_RUNNABLE
     * VARIABLES:
     *   proc_list:    the process set's list
     *   nr_process:   the number of process set
     */

    //    1. call alloc_proc to allocate a proc_struct
    //    2. call setup_kstack to allocate a kernel stack for child process
    //    3. call copy_mm to dup OR share mm according clone_flag
    //    4. call copy_thread to setup tf & context in proc_struct
    //    5. insert proc_struct into hash_list && proc_list
    //    6. call wakeup_proc to make the new child process RUNNABLE
    //    7. set ret vaule using child proc's pid

        //LAB5 YOUR CODE : (update LAB4 steps)
   /* Some Functions
    *    set_links:  set the relation links of process.  ALSO SEE: remove_links:  lean the rel
ation links of process
    *    ---------------------
        *    update step 1: set child proc's parent to current process, make sure current proc
```

```
ess's wait_state is 0
        *    update step 5: insert proc_struct into hash_list && proc_list, set the relation l
inks of process
    */
    if ((proc = alloc_proc()) == NULL) {
        goto fork_out;
    }

    proc->parent = current;
    assert(current->wait_state == 0);

    if (setup_kstack(proc) != 0) {
        goto bad_fork_cleanup_proc;
    }
    if (copy_fs(clone_flags, proc) != 0) { //for LAB8
        goto bad_fork_cleanup_kstack;
    }
    if (copy_mm(clone_flags, proc) != 0) {
        goto bad_fork_cleanup_fs;
    }
    copy_thread(proc, stack, tf);

    bool intr_flag;
    local_intr_save(intr_flag);
    {
        proc->pid = get_pid();
        hash_proc(proc);
        set_links(proc);

    }
    local_intr_restore(intr_flag);

    wakeup_proc(proc);

    ret = proc->pid;
fork_out:
    return ret;

bad_fork_cleanup_fs:  //for LAB8
    put_fs(proc);
bad_fork_cleanup_kstack:
    put_kstack(proc);
bad_fork_cleanup_proc:
    kfree(proc);
    goto fork_out;
}

// do_exit - called by sys_exit
//   1. call exit_mmap & put_pgdir & mm_destroy to free the almost all memory space of process
//   2. set process' state as PROC_ZOMBIE, then call wakeup_proc(parent) to ask parent reclaim
 itself.
//   3. call scheduler to switch to other process
int
do_exit(int error_code) {
    if (current == idleproc) {
        panic("idleproc exit.\n");
    }
    if (current == initproc) {
        panic("initproc exit.\n");
    }

    struct mm_struct *mm = current->mm;
    if (mm != NULL) {
        lcr3(boot_cr3);
        if (mm_count_dec(mm) == 0) {
```

```
            exit_mmap(mm);
            put_pgdir(mm);
            mm_destroy(mm);
        }
        current->mm = NULL;
    }
    put_fs(current); //for LAB8
    current->state = PROC_ZOMBIE;
    current->exit_code = error_code;

    bool intr_flag;
    struct proc_struct *proc;
    local_intr_save(intr_flag);
    {
        proc = current->parent;
        if (proc->wait_state == WT_CHILD) {
            wakeup_proc(proc);
        }
        while (current->cptr != NULL) {
            proc = current->cptr;
            current->cptr = proc->optr;

            proc->yptr = NULL;
            if ((proc->optr = initproc->cptr) != NULL) {
                initproc->cptr->yptr = proc;
            }
            proc->parent = initproc;
            initproc->cptr = proc;
            if (proc->state == PROC_ZOMBIE) {
                if (initproc->wait_state == WT_CHILD) {
                    wakeup_proc(initproc);
                }
            }
        }
    }
    local_intr_restore(intr_flag);

    schedule();
    panic("do_exit will not return!! %d.\n", current->pid);
}

//load_icode_read is used by load_icode in LAB8
static int
load_icode_read(int fd, void *buf, size_t len, off_t offset) {
    int ret;
    if ((ret = sysfile_seek(fd, offset, LSEEK_SET)) != 0) {
        return ret;
    }
    if ((ret = sysfile_read(fd, buf, len)) != len) {
        return (ret < 0) ? ret : -1;
    }
    return 0;
}

// load_icode -  called by sys_exec-->do_execve

static int
load_icode(int fd, int argc, char **kargv) {
    /* LAB8:EXERCISE2 YOUR CODE  HINT:how to load the file with handler fd  in to process's me
mory? how to setup argc/argv?
     * MACROs or Functions:
     *  mm_create        - create a mm
     *  setup_pgdir      - setup pgdir in mm
     *  load_icode_read  - read raw data content of program file
     *  mm_map           - build new vma
```

```
 *  pgdir_alloc_page - allocate new memory for  TEXT/DATA/BSS/stack parts
 *  lcr3             - update Page Directory Addr Register -- CR3
 */
    /* (1) create a new mm for current process
 * (2) create a new PDT, and mm->pgdir= kernel virtual addr of PDT
 * (3) copy TEXT/DATA/BSS parts in binary to memory space of process
 *    (3.1) read raw data content in file and resolve elfhdr
 *    (3.2) read raw data content in file and resolve proghdr based on info in elfhdr
 *    (3.3) call mm_map to build vma related to TEXT/DATA
 *    (3.4) callpgdir_alloc_page to allocate page for TEXT/DATA, read contents in file
 *          and copy them into the new allocated pages
 *    (3.5) callpgdir_alloc_page to allocate pages for BSS, memset zero in these pages
 * (4) call mm_map to setup user stack, and put parameters into user stack
 * (5) setup current process's mm, cr3, reset pgidr (using lcr3 MARCO)
 * (6) setup uargc and uargv in user stacks
 * (7) setup trapframe for user environment
 * (8) if up steps failed, you should cleanup the env.
 */
assert(argc >= 0 && argc <= EXEC_MAX_ARG_NUM);

if (current->mm != NULL) {
    panic("load_icode: current->mm must be empty.\n");
}

int ret = -E_NO_MEM;
struct mm_struct *mm;
if ((mm = mm_create()) == NULL) {
    goto bad_mm;
}
if (setup_pgdir(mm) != 0) {
    goto bad_pgdir_cleanup_mm;
}

struct Page *page;

struct elfhdr __elf, *elf = &__elf;
if ((ret = load_icode_read(fd, elf, sizeof(struct elfhdr), 0)) != 0) {
    goto bad_elf_cleanup_pgdir;
}

if (elf->e_magic != ELF_MAGIC) {
    ret = -E_INVAL_ELF;
    goto bad_elf_cleanup_pgdir;
}

struct proghdr __ph, *ph = &__ph;
uint32_t vm_flags, perm, phnum;
for (phnum = 0; phnum < elf->e_phnum; phnum ++) {
    off_t phoff = elf->e_phoff + sizeof(struct proghdr) * phnum;
    if ((ret = load_icode_read(fd, ph, sizeof(struct proghdr), phoff)) != 0) {
        goto bad_cleanup_mmap;
    }
    if (ph->p_type != ELF_PT_LOAD) {
        continue ;
    }
    if (ph->p_filesz > ph->p_memsz) {
        ret = -E_INVAL_ELF;
        goto bad_cleanup_mmap;
    }
    if (ph->p_filesz == 0) {
        continue ;
    }
    vm_flags = 0, perm = PTE_U;
    if (ph->p_flags & ELF_PF_X) vm_flags |= VM_EXEC;
    if (ph->p_flags & ELF_PF_W) vm_flags |= VM_WRITE;
```

```
    if (ph->p_flags & ELF_PF_R) vm_flags |= VM_READ;
    if (vm_flags & VM_WRITE) perm |= PTE_W;
    if ((ret = mm_map(mm, ph->p_va, ph->p_memsz, vm_flags, NULL)) != 0) {
        goto bad_cleanup_mmap;
    }
    off_t offset = ph->p_offset;
    size_t off, size;
    uintptr_t start = ph->p_va, end, la = ROUNDDOWN(start, PGSIZE);

    ret = -E_NO_MEM;

    end = ph->p_va + ph->p_filesz;
    while (start < end) {
        if ((page = pgdir_alloc_page(mm->pgdir, la, perm)) == NULL) {
            ret = -E_NO_MEM;
            goto bad_cleanup_mmap;
        }
        off = start - la, size = PGSIZE - off, la += PGSIZE;
        if (end < la) {
            size -= la - end;
        }
        if ((ret = load_icode_read(fd, page2kva(page) + off, size, offset)) != 0) {
            goto bad_cleanup_mmap;
        }
        start += size, offset += size;
    }
    end = ph->p_va + ph->p_memsz;

    if (start < la) {
        /* ph->p_memsz == ph->p_filesz */
        if (start == end) {
            continue ;
        }
        off = start + PGSIZE - la, size = PGSIZE - off;
        if (end < la) {
            size -= la - end;
        }
        memset(page2kva(page) + off, 0, size);
        start += size;
        assert((end < la && start == end) || (end >= la && start == la));
    }
    while (start < end) {
        if ((page = pgdir_alloc_page(mm->pgdir, la, perm)) == NULL) {
            ret = -E_NO_MEM;
            goto bad_cleanup_mmap;
        }
        off = start - la, size = PGSIZE - off, la += PGSIZE;
        if (end < la) {
            size -= la - end;
        }
        memset(page2kva(page) + off, 0, size);
        start += size;
    }
}
sysfile_close(fd);

vm_flags = VM_READ | VM_WRITE | VM_STACK;
if ((ret = mm_map(mm, USTACKTOP - USTACKSIZE, USTACKSIZE, vm_flags, NULL)) != 0) {
    goto bad_cleanup_mmap;
}
assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-PGSIZE , PTE_USER) != NULL);
assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-2*PGSIZE , PTE_USER) != NULL);
assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-3*PGSIZE , PTE_USER) != NULL);
assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-4*PGSIZE , PTE_USER) != NULL);
```

```
    mm_count_inc(mm);
    current->mm = mm;
    current->cr3 = PADDR(mm->pgdir);
    lcr3(PADDR(mm->pgdir));

    //setup argc, argv
    uint32_t argv_size=0, i;
    for (i = 0; i < argc; i ++) {
        argv_size += strnlen(kargv[i],EXEC_MAX_ARG_LEN + 1)+1;
    }

    uintptr_t stacktop = USTACKTOP - (argv_size/sizeof(long)+1)*sizeof(long);
    char** uargv=(char **)(stacktop  - argc * sizeof(char *));

    argv_size = 0;
    for (i = 0; i < argc; i ++) {
        uargv[i] = strcpy((char *)(stacktop + argv_size ), kargv[i]);
        argv_size +=  strnlen(kargv[i],EXEC_MAX_ARG_LEN + 1)+1;
    }

    stacktop = (uintptr_t)uargv - sizeof(int);
    *(int *)stacktop = argc;

    struct trapframe *tf = current->tf;
    memset(tf, 0, sizeof(struct trapframe));
    tf->tf_cs = USER_CS;
    tf->tf_ds = tf->tf_es = tf->tf_ss = USER_DS;
    tf->tf_esp = stacktop;
    tf->tf_eip = elf->e_entry;
    tf->tf_eflags = FL_IF;
    ret = 0;
out:
    return ret;
bad_cleanup_mmap:
    exit_mmap(mm);
bad_elf_cleanup_pgdir:
    put_pgdir(mm);
bad_pgdir_cleanup_mm:
    mm_destroy(mm);
bad_mm:
    goto out;
}

// this function isn't very correct in LAB8
static void
put_kargv(int argc, char **kargv) {
    while (argc > 0) {
        kfree(kargv[-- argc]);
    }
}

static int
copy_kargv(struct mm_struct *mm, int argc, char **kargv, const char **argv) {
    int i, ret = -E_INVAL;
    if (!user_mem_check(mm, (uintptr_t)argv, sizeof(const char *) * argc, 0)) {
        return ret;
    }
    for (i = 0; i < argc; i ++) {
        char *buffer;
        if ((buffer = kmalloc(EXEC_MAX_ARG_LEN + 1)) == NULL) {
            goto failed_nomem;
        }
        if (!copy_string(mm, buffer, argv[i], EXEC_MAX_ARG_LEN + 1)) {
            kfree(buffer);
            goto failed_cleanup;
```

```
        }
        kargv[i] = buffer;
    }
    return 0;

failed_nomem:
    ret = -E_NO_MEM;
failed_cleanup:
    put_kargv(i, kargv);
    return ret;
}

// do_execve - call exit_mmap(mm)&put_pgdir(mm) to reclaim memory space of current process
//           - call load_icode to setup new memory space accroding binary prog.
int
do_execve(const char *name, int argc, const char **argv) {
    static_assert(EXEC_MAX_ARG_LEN >= FS_MAX_FPATH_LEN);
    struct mm_struct *mm = current->mm;
    if (!(argc >= 1 && argc <= EXEC_MAX_ARG_NUM)) {
        return -E_INVAL;
    }

    char local_name[PROC_NAME_LEN + 1];
    memset(local_name, 0, sizeof(local_name));

    char *kargv[EXEC_MAX_ARG_NUM];
    const char *path;

    int ret = -E_INVAL;

    lock_mm(mm);
    if (name == NULL) {
        snprintf(local_name, sizeof(local_name), "<null> %d", current->pid);
    }
    else {
        if (!copy_string(mm, local_name, name, sizeof(local_name))) {
            unlock_mm(mm);
            return ret;
        }
    }
    if ((ret = copy_kargv(mm, argc, kargv, argv)) != 0) {
        unlock_mm(mm);
        return ret;
    }
    path = argv[0];
    unlock_mm(mm);
    files_closeall(current->filesp);

    /* sysfile_open will check the first argument path, thus we have to use a user-space point
er, and argv[0] may be incorrect */
    int fd;
    if ((ret = fd = sysfile_open(path, O_RDONLY)) < 0) {
        goto execve_exit;
    }
    if (mm != NULL) {
        lcr3(boot_cr3);
        if (mm_count_dec(mm) == 0) {
            exit_mmap(mm);
            put_pgdir(mm);
            mm_destroy(mm);
        }
        current->mm = NULL;
    }
    ret= -E_NO_MEM;;
    if ((ret = load_icode(fd, argc, kargv)) != 0) {
```

```
        goto execve_exit;
    }
    put_kargv(argc, kargv);
    set_proc_name(current, local_name);
    return 0;

execve_exit:
    put_kargv(argc, kargv);
    do_exit(ret);
    panic("already exit: %e.\n", ret);
}

// do_yield - ask the scheduler to reschedule
int
do_yield(void) {
    current->need_resched = 1;
    return 0;
}

// do_wait - wait one OR any children with PROC_ZOMBIE state, and free memory space of kernel
stack
//         - proc struct of this child.
// NOTE: only after do_wait function, all resources of the child proces are free.
int
do_wait(int pid, int *code_store) {
    struct mm_struct *mm = current->mm;
    if (code_store != NULL) {
        if (!user_mem_check(mm, (uintptr_t)code_store, sizeof(int), 1)) {
            return -E_INVAL;
        }
    }

    struct proc_struct *proc;
    bool intr_flag, haskid;
repeat:
    haskid = 0;
    if (pid != 0) {
        proc = find_proc(pid);
        if (proc != NULL && proc->parent == current) {
            haskid = 1;
            if (proc->state == PROC_ZOMBIE) {
                goto found;
            }
        }
    }
    else {
        proc = current->cptr;
        for (; proc != NULL; proc = proc->optr) {
            haskid = 1;
            if (proc->state == PROC_ZOMBIE) {
                goto found;
            }
        }
    }
    if (haskid) {
        current->state = PROC_SLEEPING;
        current->wait_state = WT_CHILD;
        schedule();
        if (current->flags & PF_EXITING) {
            do_exit(-E_KILLED);
        }
        goto repeat;
    }
    return -E_BAD_PROC;
```

```
found:
    if (proc == idleproc || proc == initproc) {
        panic("wait idleproc or initproc.\n");
    }
    if (code_store != NULL) {
        *code_store = proc->exit_code;
    }
    local_intr_save(intr_flag);
    {
        unhash_proc(proc);
        remove_links(proc);
    }
    local_intr_restore(intr_flag);
    put_kstack(proc);
    kfree(proc);
    return 0;
}

// do_kill - kill process with pid by set this process's flags with PF_EXITING
int
do_kill(int pid) {
    struct proc_struct *proc;
    if ((proc = find_proc(pid)) != NULL) {
        if (!(proc->flags & PF_EXITING)) {
            proc->flags |= PF_EXITING;
            if (proc->wait_state & WT_INTERRUPTED) {
                wakeup_proc(proc);
            }
            return 0;
        }
        return -E_KILLED;
    }
    return -E_INVAL;
}

// kernel_execve - do SYS_exec syscall to exec a user program called by user_main kernel_threa
d
static int
kernel_execve(const char *name, const char **argv) {
    int argc = 0, ret;
    while (argv[argc] != NULL) {
        argc ++;
    }
    asm volatile (
        "int %1;"
        : "=a" (ret)
        : "i" (T_SYSCALL), "0" (SYS_exec), "d" (name), "c" (argc), "b" (argv)
        : "memory");
    return ret;
}

#define __KERNEL_EXECVE(name, path, ...) ({                         \
const char *argv[] = {path, ##__VA_ARGS__, NULL};       \
                    cprintf("kernel_execve: pid = %d, name = \"%s\".\n",    \
                            current->pid, name);                    \
                    kernel_execve(name, argv);                      \
})

#define KERNEL_EXECVE(x, ...)                   __KERNEL_EXECVE(#x, #x, ##__VA_ARGS__)

#define KERNEL_EXECVE2(x, ...)                  KERNEL_EXECVE(x, ##__VA_ARGS__)

#define __KERNEL_EXECVE3(x, s, ...)             KERNEL_EXECVE(x, #s, ##__VA_ARGS__)

#define KERNEL_EXECVE3(x, s, ...)               __KERNEL_EXECVE3(x, s, ##__VA_ARGS__)
```

```
// user_main - kernel thread used to exec a user program
static int
user_main(void *arg) {
#ifdef TEST
#ifdef TESTSCRIPT
    KERNEL_EXECVE3(TEST, TESTSCRIPT);
#else
    KERNEL_EXECVE2(TEST);
#endif
#else
    KERNEL_EXECVE(sh);
#endif
    panic("user_main execve failed.\n");
}

// init_main - the second kernel thread used to create user_main kernel threads
static int
init_main(void *arg) {
    int ret;
    if ((ret = vfs_set_bootfs("disk0:")) != 0) {
        panic("set boot fs failed: %e.\n", ret);
    }

    size_t nr_free_pages_store = nr_free_pages();
    size_t kernel_allocated_store = kallocated();

    int pid = kernel_thread(user_main, NULL, 0);
    if (pid <= 0) {
        panic("create user_main failed.\n");
    }
 extern void check_sync(void);
    check_sync();                // check philosopher sync problem

    while (do_wait(0, NULL) == 0) {
        schedule();
    }

    fs_cleanup();

    cprintf("all user-mode processes have quit.\n");
    assert(initproc->cptr == NULL && initproc->yptr == NULL && initproc->optr == NULL);
    assert(nr_process == 2);
    assert(list_next(&proc_list) == &(initproc->list_link));
    assert(list_prev(&proc_list) == &(initproc->list_link));
    assert(nr_free_pages_store == nr_free_pages());
    assert(kernel_allocated_store == kallocated());
    cprintf("init check memory pass.\n");
    return 0;
}

// proc_init - set up the first kernel thread idleproc "idle" by itself and
//           - create the second kernel thread init_main
void
proc_init(void) {
    int i;

    list_init(&proc_list);
    for (i = 0; i < HASH_LIST_SIZE; i ++) {
        list_init(hash_list + i);
    }

    if ((idleproc = alloc_proc()) == NULL) {
        panic("cannot alloc idleproc.\n");
    }
```

```
    idleproc->pid = 0;
    idleproc->state = PROC_RUNNABLE;
    idleproc->kstack = (uintptr_t)bootstack;
    idleproc->need_resched = 1;

    if ((idleproc->filesp = files_create()) == NULL) {
        panic("create filesp (idleproc) failed.\n");
    }
    files_count_inc(idleproc->filesp);

    set_proc_name(idleproc, "idle");
    nr_process ++;

    current = idleproc;

    int pid = kernel_thread(init_main, NULL, 0);
    if (pid <= 0) {
        panic("create init_main failed.\n");
    }

    initproc = find_proc(pid);
    set_proc_name(initproc, "init");

    assert(idleproc != NULL && idleproc->pid == 0);
    assert(initproc != NULL && initproc->pid == 1);
}

// cpu_idle - at the end of kern_init, the first kernel thread idleproc will do below works
void
cpu_idle(void) {
    while (1) {
        if (current->need_resched) {
            schedule();
        }
    }
}

//FOR LAB6, set the process's priority (bigger value will get more CPU time)
void
lab6_set_priority(uint32_t priority)
{
    if (priority == 0)
        current->lab6_priority = 1;
    else current->lab6_priority = priority;
}

// do_sleep - set current process state to sleep and add timer with "time"
//          - then call scheduler. if process run again, delete timer first.
int
do_sleep(unsigned int time) {
    if (time == 0) {
        return 0;
    }
    bool intr_flag;
    local_intr_save(intr_flag);
    timer_t __timer, *timer = timer_init(&__timer, current, time);
    current->state = PROC_SLEEPING;
    current->wait_state = WT_TIMER;
    add_timer(timer);
    local_intr_restore(intr_flag);

    schedule();

    del_timer(timer);
```

```
    return 0;
}
[ucore/lab8_result/kern/process/switch.S] ++++++++++++++++++++++++++++++
.text
.globl switch_to
switch_to:                      # switch_to(from, to)

    # save from's registers
    movl 4(%esp), %eax          # eax points to from
    popl 0(%eax)                # save eip !popl
    movl %esp, 4(%eax)
    movl %ebx, 8(%eax)
    movl %ecx, 12(%eax)
    movl %edx, 16(%eax)
    movl %esi, 20(%eax)
    movl %edi, 24(%eax)
    movl %ebp, 28(%eax)

    # restore to's registers
    movl 4(%esp), %eax          # not 8(%esp): popped return address already
                                # eax now points to to
    movl 28(%eax), %ebp
    movl 24(%eax), %edi
    movl 20(%eax), %esi
    movl 16(%eax), %edx
    movl 12(%eax), %ecx
    movl 8(%eax), %ebx
    movl 4(%eax), %esp

    pushl 0(%eax)               # push eip

    ret

[ucore/lab8_result/kern/fs/devs/dev_disk0.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <mmu.h>
#include <sem.h>
#include <ide.h>
#include <inode.h>
#include <kmalloc.h>
#include <dev.h>
#include <vfs.h>
#include <iobuf.h>
#include <error.h>
#include <assert.h>

#define DISK0_BLKSIZE                   PGSIZE
#define DISK0_BUFSIZE                   (4 * DISK0_BLKSIZE)
#define DISK0_BLK_NSECT                 (DISK0_BLKSIZE / SECTSIZE)

static char *disk0_buffer;
static semaphore_t disk0_sem;

static void
lock_disk0(void) {
    down(&(disk0_sem));
}

static void
unlock_disk0(void) {
    up(&(disk0_sem));
}

static int
disk0_open(struct device *dev, uint32_t open_flags) {
```

```
    return 0;
}

static int
disk0_close(struct device *dev) {
    return 0;
}

static void
disk0_read_blks_nolock(uint32_t blkno, uint32_t nblks) {
    int ret;
    uint32_t sectno = blkno * DISK0_BLK_NSECT, nsecs = nblks * DISK0_BLK_NSECT;
    if ((ret = ide_read_secs(DISK0_DEV_NO, sectno, disk0_buffer, nsecs)) != 0) {
        panic("disk0: read blkno = %d (sectno = %d), nblks = %d (nsecs = %d): 0x%08x.\n",
                blkno, sectno, nblks, nsecs, ret);
    }
}

static void
disk0_write_blks_nolock(uint32_t blkno, uint32_t nblks) {
    int ret;
    uint32_t sectno = blkno * DISK0_BLK_NSECT, nsecs = nblks * DISK0_BLK_NSECT;
    if ((ret = ide_write_secs(DISK0_DEV_NO, sectno, disk0_buffer, nsecs)) != 0) {
        panic("disk0: write blkno = %d (sectno = %d), nblks = %d (nsecs = %d): 0x%08x.\n",
                blkno, sectno, nblks, nsecs, ret);
    }
}

static int
disk0_io(struct device *dev, struct iobuf *iob, bool write) {
    off_t offset = iob->io_offset;
    size_t resid = iob->io_resid;
    uint32_t blkno = offset / DISK0_BLKSIZE;
    uint32_t nblks = resid / DISK0_BLKSIZE;

    /* don't allow I/O that isn't block-aligned */
    if ((offset % DISK0_BLKSIZE) != 0 || (resid % DISK0_BLKSIZE) != 0) {
        return -E_INVAL;
    }

    /* don't allow I/O past the end of disk0 */
    if (blkno + nblks > dev->d_blocks) {
        return -E_INVAL;
    }

    /* read/write nothing ? */
    if (nblks == 0) {
        return 0;
    }

    lock_disk0();
    while (resid != 0) {
        size_t copied, alen = DISK0_BUFSIZE;
        if (write) {
            iobuf_move(iob, disk0_buffer, alen, 0, &copied);
            assert(copied != 0 && copied <= resid && copied % DISK0_BLKSIZE == 0);
            nblks = copied / DISK0_BLKSIZE;
            disk0_write_blks_nolock(blkno, nblks);
        }
        else {
            if (alen > resid) {
                alen = resid;
            }
            nblks = alen / DISK0_BLKSIZE;
            disk0_read_blks_nolock(blkno, nblks);
```

```
            iobuf_move(iob, disk0_buffer, alen, 1, &copied);
            assert(copied == alen && copied % DISK0_BLKSIZE == 0);
        }
        resid -= copied, blkno += nblks;
    }
    unlock_disk0();
    return 0;
}

static int
disk0_ioctl(struct device *dev, int op, void *data) {
    return -E_UNIMP;
}

static void
disk0_device_init(struct device *dev) {
    static_assert(DISK0_BLKSIZE % SECTSIZE == 0);
    if (!ide_device_valid(DISK0_DEV_NO)) {
        panic("disk0 device isn't available.\n");
    }
    dev->d_blocks = ide_device_size(DISK0_DEV_NO) / DISK0_BLK_NSECT;
    dev->d_blocksize = DISK0_BLKSIZE;
    dev->d_open = disk0_open;
    dev->d_close = disk0_close;
    dev->d_io = disk0_io;
    dev->d_ioctl = disk0_ioctl;
    sem_init(&(disk0_sem), 1);

    static_assert(DISK0_BUFSIZE % DISK0_BLKSIZE == 0);
    if ((disk0_buffer = kmalloc(DISK0_BUFSIZE)) == NULL) {
        panic("disk0 alloc buffer failed.\n");
    }
}

void
dev_init_disk0(void) {
    struct inode *node;
    if ((node = dev_create_inode()) == NULL) {
        panic("disk0: dev_create_node.\n");
    }
    disk0_device_init(vop_info(node, device));

    int ret;
    if ((ret = vfs_add_dev("disk0", node, 1)) != 0) {
        panic("disk0: vfs_add_dev: %e.\n", ret);
    }
}
```

[ucore/lab8_result/kern/fs/devs/dev_stdin.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <stdio.h>
#include <wait.h>
#include <sync.h>
#include <proc.h>
#include <sched.h>
#include <dev.h>
#include <vfs.h>
#include <iobuf.h>
#include <inode.h>
#include <unistd.h>
#include <error.h>
#include <assert.h>

#define STDIN_BUFSIZE               4096
```

```c
static char stdin_buffer[STDIN_BUFSIZE];
static off_t p_rpos, p_wpos;
static wait_queue_t __wait_queue, *wait_queue = &__wait_queue;

void
dev_stdin_write(char c) {
    bool intr_flag;
    if (c != '\0') {
        local_intr_save(intr_flag);
        {
            stdin_buffer[p_wpos % STDIN_BUFSIZE] = c;
            if (p_wpos - p_rpos < STDIN_BUFSIZE) {
                p_wpos ++;
            }
            if (!wait_queue_empty(wait_queue)) {
                wakeup_queue(wait_queue, WT_KBD, 1);
            }
        }
        local_intr_restore(intr_flag);
    }
}

static int
dev_stdin_read(char *buf, size_t len) {
    int ret = 0;
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        for (; ret < len; ret ++, p_rpos ++) {
        try_again:
            if (p_rpos < p_wpos) {
                *buf ++ = stdin_buffer[p_rpos % STDIN_BUFSIZE];
            }
            else {
                wait_t __wait, *wait = &__wait;
                wait_current_set(wait_queue, wait, WT_KBD);
                local_intr_restore(intr_flag);

                schedule();

                local_intr_save(intr_flag);
                wait_current_del(wait_queue, wait);
                if (wait->wakeup_flags == WT_KBD) {
                    goto try_again;
                }
                break;
            }
        }
    }
    local_intr_restore(intr_flag);
    return ret;
}

static int
stdin_open(struct device *dev, uint32_t open_flags) {
    if (open_flags != O_RDONLY) {
        return -E_INVAL;
    }
    return 0;
}

static int
stdin_close(struct device *dev) {
    return 0;
}
```

```
static int
stdin_io(struct device *dev, struct iobuf *iob, bool write) {
    if (!write) {
        int ret;
        if ((ret = dev_stdin_read(iob->io_base, iob->io_resid)) > 0) {
            iob->io_resid -= ret;
        }
        return ret;
    }
    return -E_INVAL;
}

static int
stdin_ioctl(struct device *dev, int op, void *data) {
    return -E_INVAL;
}

static void
stdin_device_init(struct device *dev) {
    dev->d_blocks = 0;
    dev->d_blocksize = 1;
    dev->d_open = stdin_open;
    dev->d_close = stdin_close;
    dev->d_io = stdin_io;
    dev->d_ioctl = stdin_ioctl;

    p_rpos = p_wpos = 0;
    wait_queue_init(wait_queue);
}

void
dev_init_stdin(void) {
    struct inode *node;
    if ((node = dev_create_inode()) == NULL) {
        panic("stdin: dev_create_node.\n");
    }
    stdin_device_init(vop_info(node, device));

    int ret;
    if ((ret = vfs_add_dev("stdin", node, 0)) != 0) {
        panic("stdin: vfs_add_dev: %e.\n", ret);
    }
}

[ucore/lab8_result/kern/fs/devs/dev_stdout.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <dev.h>
#include <vfs.h>
#include <iobuf.h>
#include <inode.h>
#include <unistd.h>
#include <error.h>
#include <assert.h>

static int
stdout_open(struct device *dev, uint32_t open_flags) {
    if (open_flags != O_WRONLY) {
        return -E_INVAL;
    }
    return 0;
}

static int
```

```
stdout_close(struct device *dev) {
    return 0;
}

static int
stdout_io(struct device *dev, struct iobuf *iob, bool write) {
    if (write) {
        char *data = iob->io_base;
        for (; iob->io_resid != 0; iob->io_resid --) {
            cputchar(*data ++);
        }
        return 0;
    }
    return -E_INVAL;
}

static int
stdout_ioctl(struct device *dev, int op, void *data) {
    return -E_INVAL;
}

static void
stdout_device_init(struct device *dev) {
    dev->d_blocks = 0;
    dev->d_blocksize = 1;
    dev->d_open = stdout_open;
    dev->d_close = stdout_close;
    dev->d_io = stdout_io;
    dev->d_ioctl = stdout_ioctl;
}

void
dev_init_stdout(void) {
    struct inode *node;
    if ((node = dev_create_inode()) == NULL) {
        panic("stdout: dev_create_node.\n");
    }
    stdout_device_init(vop_info(node, device));

    int ret;
    if ((ret = vfs_add_dev("stdout", node, 0)) != 0) {
        panic("stdout: vfs_add_dev: %e.\n", ret);
    }
}

[ucore/lab8_result/kern/fs/devs/dev.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_FS_DEVS_DEV_H__
#define __KERN_FS_DEVS_DEV_H__

#include <defs.h>

struct inode;
struct iobuf;

/*
 * Filesystem-namespace-accessible device.
 * d_io is for both reads and writes; the iobuf will indicates the direction.
 */
struct device {
    size_t d_blocks;
    size_t d_blocksize;
    int (*d_open)(struct device *dev, uint32_t open_flags);
    int (*d_close)(struct device *dev);
    int (*d_io)(struct device *dev, struct iobuf *iob, bool write);
    int (*d_ioctl)(struct device *dev, int op, void *data);
```

```
};

#define dop_open(dev, open_flags)           ((dev)->d_open(dev, open_flags))
#define dop_close(dev)                      ((dev)->d_close(dev))
#define dop_io(dev, iob, write)             ((dev)->d_io(dev, iob, write))
#define dop_ioctl(dev, op, data)            ((dev)->d_ioctl(dev, op, data))

void dev_init(void);
struct inode *dev_create_inode(void);

#endif /* !__KERN_FS_DEVS_DEV_H__ */

[ucore/lab8_result/kern/fs/devs/dev.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <stat.h>
#include <dev.h>
#include <inode.h>
#include <unistd.h>
#include <error.h>

/*
 * dev_open - Called for each open().
 */
static int
dev_open(struct inode *node, uint32_t open_flags) {
    if (open_flags & (O_CREAT | O_TRUNC | O_EXCL | O_APPEND)) {
        return -E_INVAL;
    }
    struct device *dev = vop_info(node, device);
    return dop_open(dev, open_flags);
}

/*
 * dev_close - Called on the last close(). Just pass through.
 */
static int
dev_close(struct inode *node) {
    struct device *dev = vop_info(node, device);
    return dop_close(dev);
}

/*
 * dev_read -Called for read. Hand off to iobuf.
 */
static int
dev_read(struct inode *node, struct iobuf *iob) {
    struct device *dev = vop_info(node, device);
    return dop_io(dev, iob, 0);
}

/*
 * dev_write -Called for write. Hand off to iobuf.
 */
static int
dev_write(struct inode *node, struct iobuf *iob) {
    struct device *dev = vop_info(node, device);
    return dop_io(dev, iob, 1);
}

/*
 * dev_ioctl - Called for ioctl(). Just pass through.
 */
static int
dev_ioctl(struct inode *node, int op, void *data) {
```

```
    struct device *dev = vop_info(node, device);
    return dop_ioctl(dev, op, data);
}

/*
 * dev_fstat - Called for stat().
 *             Set the type and the size (block devices only).
 *             The link count for a device is always 1.
 */
static int
dev_fstat(struct inode *node, struct stat *stat) {
    int ret;
    memset(stat, 0, sizeof(struct stat));
    if ((ret = vop_gettype(node, &(stat->st_mode))) != 0) {
        return ret;
    }
    struct device *dev = vop_info(node, device);
    stat->st_nlinks = 1;
    stat->st_blocks = dev->d_blocks;
    stat->st_size = stat->st_blocks * dev->d_blocksize;
    return 0;
}

/*
 * dev_gettype - Return the type. A device is a "block device" if it has a known
 *               length. A device that generates data in a stream is a "character
 *               device".
 */
static int
dev_gettype(struct inode *node, uint32_t *type_store) {
    struct device *dev = vop_info(node, device);
    *type_store = (dev->d_blocks > 0) ? S_IFBLK : S_IFCHR;
    return 0;
}

/*
 * dev_tryseek - Attempt a seek.
 *               For block devices, require block alignment.
 *               For character devices, prohibit seeking entirely.
 */
static int
dev_tryseek(struct inode *node, off_t pos) {
    struct device *dev = vop_info(node, device);
    if (dev->d_blocks > 0) {
        if ((pos % dev->d_blocksize) == 0) {
            if (pos >= 0 && pos < dev->d_blocks * dev->d_blocksize) {
                return 0;
            }
        }
    }
    return -E_INVAL;
}

/*
 * dev_lookup - Name lookup.
 *
 * One interesting feature of device:name pathname syntax is that you
 * can implement pathnames on arbitrary devices. For instance, if you
 * had a graphics device that supported multiple resolutions (which we
 * don't), you might arrange things so that you could open it with
 * pathnames like "video:800x600/24bpp" in order to select the operating
 * mode.
 *
 * However, we have no support for this in the base system.
 */
```

```
static int
dev_lookup(struct inode *node, char *path, struct inode **node_store) {
    if (*path != '\0') {
        return -E_NOENT;
    }
    vop_ref_inc(node);
    *node_store = node;
    return 0;
}

/*
 * Function table for device inodes.
 */
static const struct inode_ops dev_node_ops = {
    .vop_magic                      = VOP_MAGIC,
    .vop_open                       = dev_open,
    .vop_close                      = dev_close,
    .vop_read                       = dev_read,
    .vop_write                      = dev_write,
    .vop_fstat                      = dev_fstat,
    .vop_ioctl                      = dev_ioctl,
    .vop_gettype                    = dev_gettype,
    .vop_tryseek                    = dev_tryseek,
    .vop_lookup                     = dev_lookup,
};

#define init_device(x)                                          \
    do {                                                        \
        extern void dev_init_##x(void);                         \
        dev_init_##x();                                         \
    } while (0)

/* dev_init - Initialization functions for builtin vfs-level devices. */
void
dev_init(void) {
   // init_device(null);
    init_device(stdin);
    init_device(stdout);
    init_device(disk0);
}
/* dev_create_inode - Create inode for a vfs-level device. */
struct inode *
dev_create_inode(void) {
    struct inode *node;
    if ((node = alloc_inode(device)) != NULL) {
        vop_init(node, &dev_node_ops, NULL);
    }
    return node;
}

[ucore/lab8_result/kern/fs/sysfile.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <vmm.h>
#include <proc.h>
#include <kmalloc.h>
#include <vfs.h>
#include <file.h>
#include <iobuf.h>
#include <sysfile.h>
#include <stat.h>
#include <dirent.h>
#include <unistd.h>
#include <error.h>
#include <assert.h>
```

```
#define IOBUF_SIZE                          4096

/* copy_path - copy path name */
static int
copy_path(char **to, const char *from) {
    struct mm_struct *mm = current->mm;
    char *buffer;
    if ((buffer = kmalloc(FS_MAX_FPATH_LEN + 1)) == NULL) {
        return -E_NO_MEM;
    }
    lock_mm(mm);
    if (!copy_string(mm, buffer, from, FS_MAX_FPATH_LEN + 1)) {
        unlock_mm(mm);
        goto failed_cleanup;
    }
    unlock_mm(mm);
    *to = buffer;
    return 0;

failed_cleanup:
    kfree(buffer);
    return -E_INVAL;
}

/* sysfile_open - open file */
int
sysfile_open(const char *__path, uint32_t open_flags) {
    int ret;
    char *path;
    if ((ret = copy_path(&path, __path)) != 0) {
        return ret;
    }
    ret = file_open(path, open_flags);
    kfree(path);
    return ret;
}

/* sysfile_close - close file */
int
sysfile_close(int fd) {
    return file_close(fd);
}

/* sysfile_read - read file */
int
sysfile_read(int fd, void *base, size_t len) {
    struct mm_struct *mm = current->mm;
    if (len == 0) {
        return 0;
    }
    if (!file_testfd(fd, 1, 0)) {
        return -E_INVAL;
    }
    void *buffer;
    if ((buffer = kmalloc(IOBUF_SIZE)) == NULL) {
        return -E_NO_MEM;
    }

    int ret = 0;
    size_t copied = 0, alen;
    while (len != 0) {
        if ((alen = IOBUF_SIZE) > len) {
            alen = len;
        }
```

```
        ret = file_read(fd, buffer, alen, &alen);
        if (alen != 0) {
            lock_mm(mm);
            {
                if (copy_to_user(mm, base, buffer, alen)) {
                    assert(len >= alen);
                    base += alen, len -= alen, copied += alen;
                }
                else if (ret == 0) {
                    ret = -E_INVAL;
                }
            }
            unlock_mm(mm);
        }
        if (ret != 0 || alen == 0) {
            goto out;
        }
    }

out:
    kfree(buffer);
    if (copied != 0) {
        return copied;
    }
    return ret;
}

/* sysfile_write - write file */
int
sysfile_write(int fd, void *base, size_t len) {
    struct mm_struct *mm = current->mm;
    if (len == 0) {
        return 0;
    }
    if (!file_testfd(fd, 0, 1)) {
        return -E_INVAL;
    }
    void *buffer;
    if ((buffer = kmalloc(IOBUF_SIZE)) == NULL) {
        return -E_NO_MEM;
    }

    int ret = 0;
    size_t copied = 0, alen;
    while (len != 0) {
        if ((alen = IOBUF_SIZE) > len) {
            alen = len;
        }
        lock_mm(mm);
        {
            if (!copy_from_user(mm, buffer, base, alen, 0)) {
                ret = -E_INVAL;
            }
        }
        unlock_mm(mm);
        if (ret == 0) {
            ret = file_write(fd, buffer, alen, &alen);
            if (alen != 0) {
                assert(len >= alen);
                base += alen, len -= alen, copied += alen;
            }
        }
        if (ret != 0 || alen == 0) {
            goto out;
        }
```

```
    }

out:
    kfree(buffer);
    if (copied != 0) {
        return copied;
    }
    return ret;
}

/* sysfile_seek - seek file */
int
sysfile_seek(int fd, off_t pos, int whence) {
    return file_seek(fd, pos, whence);
}

/* sysfile_fstat - stat file */
int
sysfile_fstat(int fd, struct stat *__stat) {
    struct mm_struct *mm = current->mm;
    int ret;
    struct stat __local_stat, *stat = &__local_stat;
    if ((ret = file_fstat(fd, stat)) != 0) {
        return ret;
    }

    lock_mm(mm);
    {
        if (!copy_to_user(mm, __stat, stat, sizeof(struct stat))) {
            ret = -E_INVAL;
        }
    }
    unlock_mm(mm);
    return ret;
}

/* sysfile_fsync - sync file */
int
sysfile_fsync(int fd) {
    return file_fsync(fd);
}

/* sysfile_chdir - change dir */
int
sysfile_chdir(const char *__path) {
    int ret;
    char *path;
    if ((ret = copy_path(&path, __path)) != 0) {
        return ret;
    }
    ret = vfs_chdir(path);
    kfree(path);
    return ret;
}

/* sysfile_link - link file */
int
sysfile_link(const char *__path1, const char *__path2) {
    int ret;
    char *old_path, *new_path;
    if ((ret = copy_path(&old_path, __path1)) != 0) {
        return ret;
    }
    if ((ret = copy_path(&new_path, __path2)) != 0) {
        kfree(old_path);
```

```
        return ret;
    }
    ret = vfs_link(old_path, new_path);
    kfree(old_path), kfree(new_path);
    return ret;
}

/* sysfile_rename - rename file */
int
sysfile_rename(const char *__path1, const char *__path2) {
    int ret;
    char *old_path, *new_path;
    if ((ret = copy_path(&old_path, __path1)) != 0) {
        return ret;
    }
    if ((ret = copy_path(&new_path, __path2)) != 0) {
        kfree(old_path);
        return ret;
    }
    ret = vfs_rename(old_path, new_path);
    kfree(old_path), kfree(new_path);
    return ret;
}

/* sysfile_unlink - unlink file */
int
sysfile_unlink(const char *__path) {
    int ret;
    char *path;
    if ((ret = copy_path(&path, __path)) != 0) {
        return ret;
    }
    ret = vfs_unlink(path);
    kfree(path);
    return ret;
}

/* sysfile_get cwd - get current working directory */
int
sysfile_getcwd(char *buf, size_t len) {
    struct mm_struct *mm = current->mm;
    if (len == 0) {
        return -E_INVAL;
    }

    int ret = -E_INVAL;
    lock_mm(mm);
    {
        if (user_mem_check(mm, (uintptr_t)buf, len, 1)) {
            struct iobuf __iob, *iob = iobuf_init(&__iob, buf, len, 0);
            ret = vfs_getcwd(iob);
        }
    }
    unlock_mm(mm);
    return ret;
}

/* sysfile_getdirentry - get the file entry in DIR */
int
sysfile_getdirentry(int fd, struct dirent *__direntp) {
    struct mm_struct *mm = current->mm;
    struct dirent *direntp;
    if ((direntp = kmalloc(sizeof(struct dirent))) == NULL) {
        return -E_NO_MEM;
    }
```

```
    int ret = 0;
    lock_mm(mm);
    {
        if (!copy_from_user(mm, &(direntp->offset), &(__direntp->offset), sizeof(direntp->offs
et), 1)) {
            ret = -E_INVAL;
        }
    }
    unlock_mm(mm);

    if (ret != 0 || (ret = file_getdirentry(fd, direntp)) != 0) {
        goto out;
    }

    lock_mm(mm);
    {
        if (!copy_to_user(mm, __direntp, direntp, sizeof(struct dirent))) {
            ret = -E_INVAL;
        }
    }
    unlock_mm(mm);

out:
    kfree(direntp);
    return ret;
}

/* sysfile_dup -  duplicate fd1 to fd2 */
int
sysfile_dup(int fd1, int fd2) {
    return file_dup(fd1, fd2);
}

int
sysfile_pipe(int *fd_store) {
    return -E_UNIMP;
}

int
sysfile_mkfifo(const char *__name, uint32_t open_flags) {
    return -E_UNIMP;
}

[ucore/lab8_result/kern/fs/vfs/vfspath.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <vfs.h>
#include <inode.h>
#include <iobuf.h>
#include <stat.h>
#include <proc.h>
#include <error.h>
#include <assert.h>

/*
 * get_cwd_nolock - retrieve current process's working directory. without lock protect
 */
static struct inode *
get_cwd_nolock(void) {
    return current->filesp->pwd;
}
/*
 * set_cwd_nolock - set current working directory.
 */
```

```
static void
set_cwd_nolock(struct inode *pwd) {
    current->filesp->pwd = pwd;
}

/*
 * lock_cfs - lock the fs related process on current process
 */
static void
lock_cfs(void) {
    lock_files(current->filesp);
}
/*
 * unlock_cfs - unlock the fs related process on current process
 */
static void
unlock_cfs(void) {
    unlock_files(current->filesp);
}

/*
 *  vfs_get_curdir - Get current directory as a inode.
 */
int
vfs_get_curdir(struct inode **dir_store) {
    struct inode *node;
    if ((node = get_cwd_nolock()) != NULL) {
        vop_ref_inc(node);
        *dir_store = node;
        return 0;
    }
    return -E_NOENT;
}

/*
 * vfs_set_curdir - Set current directory as a inode.
 *                  The passed inode must in fact be a directory.
 */
int
vfs_set_curdir(struct inode *dir) {
    int ret = 0;
    lock_cfs();
    struct inode *old_dir;
    if ((old_dir = get_cwd_nolock()) != dir) {
        if (dir != NULL) {
            uint32_t type;
            if ((ret = vop_gettype(dir, &type)) != 0) {
                goto out;
            }
            if (!S_ISDIR(type)) {
                ret = -E_NOTDIR;
                goto out;
            }
            vop_ref_inc(dir);
        }
        set_cwd_nolock(dir);
        if (old_dir != NULL) {
            vop_ref_dec(old_dir);
        }
    }
out:
    unlock_cfs();
    return ret;
}
```

```
/*
 * vfs_chdir - Set current directory, as a pathname. Use vfs_lookup to translate
 *             it to a inode.
 */
int
vfs_chdir(char *path) {
    int ret;
    struct inode *node;
    if ((ret = vfs_lookup(path, &node)) == 0) {
        ret = vfs_set_curdir(node);
        vop_ref_dec(node);
    }
    return ret;
}
/*
 * vfs_getcwd - retrieve current working directory(cwd).
 */
int
vfs_getcwd(struct iobuf *iob) {
    int ret;
    struct inode *node;
    if ((ret = vfs_get_curdir(&node)) != 0) {
        return ret;
    }
    assert(node->in_fs != NULL);

    const char *devname = vfs_get_devname(node->in_fs);
    if ((ret = iobuf_move(iob, (char *)devname, strlen(devname), 1, NULL)) != 0) {
        goto out;
    }
    char colon = ':';
    if ((ret = iobuf_move(iob, &colon, sizeof(colon), 1, NULL)) != 0) {
        goto out;
    }
    ret = vop_namefile(node, iob);

out:
    vop_ref_dec(node);
    return ret;
}

[ucore/lab8_result/kern/fs/vfs/vfsfile.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <vfs.h>
#include <inode.h>
#include <unistd.h>
#include <error.h>
#include <assert.h>


// open file in vfs, get/create inode for file with filename path.
int
vfs_open(char *path, uint32_t open_flags, struct inode **node_store) {
    bool can_write = 0;
    switch (open_flags & O_ACCMODE) {
    case O_RDONLY:
        break;
    case O_WRONLY:
    case O_RDWR:
        can_write = 1;
        break;
    default:
        return -E_INVAL;
    }
```

```
    if (open_flags & O_TRUNC) {
        if (!can_write) {
            return -E_INVAL;
        }
    }

    int ret;
    struct inode *node;
    bool excl = (open_flags & O_EXCL) != 0;
    bool create = (open_flags & O_CREAT) != 0;
    ret = vfs_lookup(path, &node);

    if (ret != 0) {
        if (ret == -16 && (create)) {
            char *name;
            struct inode *dir;
            if ((ret = vfs_lookup_parent(path, &dir, &name)) != 0) {
                return ret;
            }
            ret = vop_create(dir, name, excl, &node);
        } else return ret;
    } else if (excl && create) {
        return -E_EXISTS;
    }
    assert(node != NULL);

    if ((ret = vop_open(node, open_flags)) != 0) {
        vop_ref_dec(node);
        return ret;
    }

    vop_open_inc(node);
    if (open_flags & O_TRUNC || create) {
        if ((ret = vop_truncate(node, 0)) != 0) {
            vop_open_dec(node);
            vop_ref_dec(node);
            return ret;
        }
    }
    *node_store = node;
    return 0;
}

// close file in vfs
int
vfs_close(struct inode *node) {
    vop_open_dec(node);
    vop_ref_dec(node);
    return 0;
}

// unimplement
int
vfs_unlink(char *path) {
    return -E_UNIMP;
}

// unimplement
int
vfs_rename(char *old_path, char *new_path) {
    return -E_UNIMP;
}

// unimplement
```

```
int
vfs_link(char *old_path, char *new_path) {
    return -E_UNIMP;
}

// unimplement
int
vfs_symlink(char *old_path, char *new_path) {
    return -E_UNIMP;
}

// unimplement
int
vfs_readlink(char *path, struct iobuf *iob) {
    return -E_UNIMP;
}

// unimplement
int
vfs_mkdir(char *path){
    return -E_UNIMP;
}
[ucore/lab8_result/kern/fs/vfs/vfsdev.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <string.h>
#include <vfs.h>
#include <dev.h>
#include <inode.h>
#include <sem.h>
#include <list.h>
#include <kmalloc.h>
#include <unistd.h>
#include <error.h>
#include <assert.h>

// device info entry in vdev_list
typedef struct {
    const char *devname;
    struct inode *devnode;
    struct fs *fs;
    bool mountable;
    list_entry_t vdev_link;
} vfs_dev_t;

#define le2vdev(le, member)                         \
    to_struct((le), vfs_dev_t, member)

static list_entry_t vdev_list;      // device info list in vfs layer
static semaphore_t vdev_list_sem;

static void
lock_vdev_list(void) {
    down(&vdev_list_sem);
}

static void
unlock_vdev_list(void) {
    up(&vdev_list_sem);
}

void
vfs_devlist_init(void) {
    list_init(&vdev_list);
    sem_init(&vdev_list_sem, 1);
```

```
}

// vfs_cleanup - finally clean (or sync) fs
void
vfs_cleanup(void) {
    if (!list_empty(&vdev_list)) {
        lock_vdev_list();
        {
            list_entry_t *list = &vdev_list, *le = list;
            while ((le = list_next(le)) != list) {
                vfs_dev_t *vdev = le2vdev(le, vdev_link);
                if (vdev->fs != NULL) {
                    fsop_cleanup(vdev->fs);
                }
            }
        }
        unlock_vdev_list();
    }
}

/*
 * vfs_get_root - Given a device name (stdin, stdout, etc.), hand
 *                back an appropriate inode.
 */
int
vfs_get_root(const char *devname, struct inode **node_store) {
    assert(devname != NULL);
    int ret = -E_NO_DEV;
    if (!list_empty(&vdev_list)) {
        lock_vdev_list();
        {
            list_entry_t *list = &vdev_list, *le = list;
            while ((le = list_next(le)) != list) {
                vfs_dev_t *vdev = le2vdev(le, vdev_link);
                if (strcmp(devname, vdev->devname) == 0) {
                    struct inode *found = NULL;
                    if (vdev->fs != NULL) {
                        found = fsop_get_root(vdev->fs);
                    }
                    else if (!vdev->mountable) {
                        vop_ref_inc(vdev->devnode);
                        found = vdev->devnode;
                    }
                    if (found != NULL) {
                        ret = 0, *node_store = found;
                    }
                    else {
                        ret = -E_NA_DEV;
                    }
                    break;
                }
            }
        }
        unlock_vdev_list();
    }
    return ret;
}

/*
 * vfs_get_devname - Given a filesystem, hand back the name of the device it's mounted on.
 */
const char *
vfs_get_devname(struct fs *fs) {
    assert(fs != NULL);
    list_entry_t *list = &vdev_list, *le = list;
```

```
    while ((le = list_next(le)) != list) {
        vfs_dev_t *vdev = le2vdev(le, vdev_link);
        if (vdev->fs == fs) {
            return vdev->devname;
        }
    }
    return NULL;
}

/*
 * check_devname_confilct - Is there alreadily device which has the same name?
 */
static bool
check_devname_conflict(const char *devname) {
    list_entry_t *list = &vdev_list, *le = list;
    while ((le = list_next(le)) != list) {
        vfs_dev_t *vdev = le2vdev(le, vdev_link);
        if (strcmp(vdev->devname, devname) == 0) {
            return 0;
        }
    }
    return 1;
}


/*
* vfs_do_add - Add a new device to the VFS layer's device table.
*
* If "mountable" is set, the device will be treated as one that expects
* to have a filesystem mounted on it, and a raw device will be created
* for direct access.
*/
static int
vfs_do_add(const char *devname, struct inode *devnode, struct fs *fs, bool mountable) {
    assert(devname != NULL);
    assert((devnode == NULL && !mountable) || (devnode != NULL && check_inode_type(devnode, de
vice)));
    if (strlen(devname) > FS_MAX_DNAME_LEN) {
        return -E_TOO_BIG;
    }

    int ret = -E_NO_MEM;
    char *s_devname;
    if ((s_devname = strdup(devname)) == NULL) {
        return ret;
    }

    vfs_dev_t *vdev;
    if ((vdev = kmalloc(sizeof(vfs_dev_t))) == NULL) {
        goto failed_cleanup_name;
    }

    ret = -E_EXISTS;
    lock_vdev_list();
    if (!check_devname_conflict(s_devname)) {
        unlock_vdev_list();
        goto failed_cleanup_vdev;
    }
    vdev->devname = s_devname;
    vdev->devnode = devnode;
    vdev->mountable = mountable;
    vdev->fs = fs;

    list_add(&vdev_list, &(vdev->vdev_link));
    unlock_vdev_list();
```

```
    return 0;

failed_cleanup_vdev:
    kfree(vdev);
failed_cleanup_name:
    kfree(s_devname);
    return ret;
}

/*
 * vfs_add_fs - Add a new fs,  by name. See  vfs_do_add information for the description of
 *              mountable.
 */
int
vfs_add_fs(const char *devname, struct fs *fs) {
    return vfs_do_add(devname, NULL, fs, 0);
}

/*
 * vfs_add_dev - Add a new device, by name. See  vfs_do_add information for the description of
 *               mountable.
 */
int
vfs_add_dev(const char *devname, struct inode *devnode, bool mountable) {
    return vfs_do_add(devname, devnode, NULL, mountable);
}

/*
 * find_mount - Look for a mountable device named DEVNAME.
 *              Should already hold vdev_list lock.
 */
static int
find_mount(const char *devname, vfs_dev_t **vdev_store) {
    assert(devname != NULL);
    list_entry_t *list = &vdev_list, *le = list;
    while ((le = list_next(le)) != list) {
        vfs_dev_t *vdev = le2vdev(le, vdev_link);
        if (vdev->mountable && strcmp(vdev->devname, devname) == 0) {
            *vdev_store = vdev;
            return 0;
        }
    }
    return -E_NO_DEV;
}

/*
 * vfs_mount - Mount a filesystem. Once we've found the device, call MOUNTFUNC to
 *             set up the filesystem and hand back a struct fs.
 *
 * The DATA argument is passed through unchanged to MOUNTFUNC.
 */
int
vfs_mount(const char *devname, int (*mountfunc)(struct device *dev, struct fs **fs_store)) {
    int ret;
    lock_vdev_list();
    vfs_dev_t *vdev;
    if ((ret = find_mount(devname, &vdev)) != 0) {
        goto out;
    }
    if (vdev->fs != NULL) {
        ret = -E_BUSY;
        goto out;
    }
    assert(vdev->devname != NULL && vdev->mountable);
```

```
    struct device *dev = vop_info(vdev->devnode, device);
    if ((ret = mountfunc(dev, &(vdev->fs))) == 0) {
        assert(vdev->fs != NULL);
        cprintf("vfs: mount %s.\n", vdev->devname);
    }

out:
    unlock_vdev_list();
    return ret;
}

/*
 * vfs_unmount - Unmount a filesystem/device by name.
 *               First calls FSOP_SYNC on the filesystem; then calls FSOP_UNMOUNT.
 */
int
vfs_unmount(const char *devname) {
    int ret;
    lock_vdev_list();
    vfs_dev_t *vdev;
    if ((ret = find_mount(devname, &vdev)) != 0) {
        goto out;
    }
    if (vdev->fs == NULL) {
        ret = -E_INVAL;
        goto out;
    }
    assert(vdev->devname != NULL && vdev->mountable);

    if ((ret = fsop_sync(vdev->fs)) != 0) {
        goto out;
    }
    if ((ret = fsop_unmount(vdev->fs)) == 0) {
        vdev->fs = NULL;
        cprintf("vfs: unmount %s.\n", vdev->devname);
    }

out:
    unlock_vdev_list();
    return ret;
}

/*
 * vfs_unmount_all - Global unmount function.
 */
int
vfs_unmount_all(void) {
    if (!list_empty(&vdev_list)) {
        lock_vdev_list();
        {
            list_entry_t *list = &vdev_list, *le = list;
            while ((le = list_next(le)) != list) {
                vfs_dev_t *vdev = le2vdev(le, vdev_link);
                if (vdev->mountable && vdev->fs != NULL) {
                    int ret;
                    if ((ret = fsop_sync(vdev->fs)) != 0) {
                        cprintf("vfs: warning: sync failed for %s: %e.\n", vdev->devname, ret)
;
                        continue ;
                    }
                    if ((ret = fsop_unmount(vdev->fs)) != 0) {
                        cprintf("vfs: warning: unmount failed for %s: %e.\n", vdev->devname, r
et);
                        continue ;
                    }
```

```
                    vdev->fs = NULL;
                    cprintf("vfs: unmount %s.\n", vdev->devname);
                }
            }
        }
        unlock_vdev_list();
    }
    return 0;
}

[ucore/lab8_result/kern/fs/vfs/vfslookup.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <vfs.h>
#include <inode.h>
#include <error.h>
#include <assert.h>

/*
 * get_device- Common code to pull the device name, if any, off the front of a
 *             path and choose the inode to begin the name lookup relative to.
 */

static int
get_device(char *path, char **subpath, struct inode **node_store) {
    int i, slash = -1, colon = -1;
    for (i = 0; path[i] != '\0'; i ++) {
        if (path[i] == ':') { colon = i; break; }
        if (path[i] == '/') { slash = i; break; }
    }
    if (colon < 0 && slash != 0) {
        /* *
         * No colon before a slash, so no device name specified, and the slash isn't leading
         * or is also absent, so this is a relative path or just a bare filename. Start from
         * the current directory, and use the whole thing as the subpath.
         * */
        *subpath = path;
        return vfs_get_curdir(node_store);
    }
    if (colon > 0) {
        /* device:path - get root of device's filesystem */
        path[colon] = '\0';

        /* device:/path - skip slash, treat as device:path */
        while (path[++ colon] == '/');
        *subpath = path + colon;
        return vfs_get_root(path, node_store);
    }

    /* *
     * we have either /path or :path
     * /path is a path relative to the root of the "boot filesystem"
     * :path is a path relative to the root of the current filesystem
     * */
    int ret;
    if (*path == '/') {
        if ((ret = vfs_get_bootfs(node_store)) != 0) {
            return ret;
        }
    }
    else {
        assert(*path == ':');
        struct inode *node;
        if ((ret = vfs_get_curdir(&node)) != 0) {
            return ret;
```

```
        }
        /* The current directory may not be a device, so it must have a fs. */
        assert(node->in_fs != NULL);
        *node_store = fsop_get_root(node->in_fs);
        vop_ref_dec(node);
    }

    /* ///... or :/... */
    while (*(++ path) == '/');
    *subpath = path;
    return 0;
}

/*
 * vfs_lookup - get the inode according to the path filename
 */
int
vfs_lookup(char *path, struct inode **node_store) {
    int ret;
    struct inode *node;
    if ((ret = get_device(path, &path, &node)) != 0) {
        return ret;
    }
    if (*path != '\0') {
        ret = vop_lookup(node, path, node_store);
        vop_ref_dec(node);
        return ret;
    }
    *node_store = node;
    return 0;
}

/*
 * vfs_lookup_parent - Name-to-vnode translation.
 *   (In BSD, both of these are subsumed by namei().)
 */
int
vfs_lookup_parent(char *path, struct inode **node_store, char **endp){
    int ret;
    struct inode *node;
    if ((ret = get_device(path, &path, &node)) != 0) {
        return ret;
    }
    *endp = path;
    *node_store = node;
    return 0;
}
[ucore/lab8_result/kern/fs/vfs/vfs.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <string.h>
#include <vfs.h>
#include <inode.h>
#include <sem.h>
#include <kmalloc.h>
#include <error.h>

static semaphore_t bootfs_sem;
static struct inode *bootfs_node = NULL;

extern void vfs_devlist_init(void);

// __alloc_fs - allocate memory for fs, and set fs type
struct fs *
__alloc_fs(int type) {
```

```
    struct fs *fs;
    if ((fs = kmalloc(sizeof(struct fs))) != NULL) {
        fs->fs_type = type;
    }
    return fs;
}

// vfs_init -  vfs initialize
void
vfs_init(void) {
    sem_init(&bootfs_sem, 1);
    vfs_devlist_init();
}

// lock_bootfs - lock  for bootfs
static void
lock_bootfs(void) {
    down(&bootfs_sem);
}
// ulock_bootfs - ulock for bootfs
static void
unlock_bootfs(void) {
    up(&bootfs_sem);
}

// change_bootfs - set the new fs inode
static void
change_bootfs(struct inode *node) {
    struct inode *old;
    lock_bootfs();
    {
        old = bootfs_node, bootfs_node = node;
    }
    unlock_bootfs();
    if (old != NULL) {
        vop_ref_dec(old);
    }
}

// vfs_set_bootfs - change the dir of file system
int
vfs_set_bootfs(char *fsname) {
    struct inode *node = NULL;
    if (fsname != NULL) {
        char *s;
        if ((s = strchr(fsname, ':')) == NULL || s[1] != '\0') {
            return -E_INVAL;
        }
        int ret;
        if ((ret = vfs_chdir(fsname)) != 0) {
            return ret;
        }
        if ((ret = vfs_get_curdir(&node)) != 0) {
            return ret;
        }
    }
    change_bootfs(node);
    return 0;
}

// vfs_get_bootfs - get the inode of bootfs
int
vfs_get_bootfs(struct inode **node_store) {
    struct inode *node = NULL;
    if (bootfs_node != NULL) {
```

```
        lock_bootfs();
        {
            if ((node = bootfs_node) != NULL) {
                vop_ref_inc(bootfs_node);
            }
        }
        unlock_bootfs();
    }
    if (node == NULL) {
        return -E_NOENT;
    }
    *node_store = node;
    return 0;
}
```

[ucore/lab8_result/kern/fs/vfs/inode.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_FS_VFS_INODE_H__
#define __KERN_FS_VFS_INODE_H__

#include <defs.h>
#include <dev.h>
#include <sfs.h>
#include <atomic.h>
#include <assert.h>

struct stat;
struct iobuf;

/*
 * A struct inode is an abstract representation of a file.
 *
 * It is an interface that allows the kernel's filesystem-independent
 * code to interact usefully with multiple sets of filesystem code.
 */

/*
 * Abstract low-level file.
 *
 * Note: in_info is Filesystem-specific data, in_type is the inode type
 *
 * open_count is managed using VOP_INCOPEN and VOP_DECOPEN by
 * vfs_open() and vfs_close(). Code above the VFS layer should not
 * need to worry about it.
 */
struct inode {
    union {
        struct device __device_info;
        struct sfs_inode __sfs_inode_info;
    } in_info;
    enum {
        inode_type_device_info = 0x1234,
        inode_type_sfs_inode_info,
    } in_type;
    int ref_count;
    int open_count;
    struct fs *in_fs;
    const struct inode_ops *in_ops;
};

#define __in_type(type)                                                 inode_type_##type##_info

#define check_inode_type(node, type)                                    ((node)->in_type == __in_t
ype(type))

#define __vop_info(node, type)                                          \
```

```
    ({                                                              \
        struct inode *__node = (node);                              \
        assert(__node != NULL && check_inode_type(__node, type));   \
        &(__node->in_info.__##type##_info);                         \
     })

#define vop_info(node, type)                                        __vop_info(node, type)

#define info2node(info, type)                                       \
    to_struct((info), struct inode, in_info.__##type##_info)

struct inode *__alloc_inode(int type);

#define alloc_inode(type)                                           __alloc_inode(__in_type(ty
pe))

#define MAX_INODE_COUNT                     0x10000

int inode_ref_inc(struct inode *node);
int inode_ref_dec(struct inode *node);
int inode_open_inc(struct inode *node);
int inode_open_dec(struct inode *node);

void inode_init(struct inode *node, const struct inode_ops *ops, struct fs *fs);
void inode_kill(struct inode *node);

#define VOP_MAGIC                           0x8c4ba476

/*
 * Abstract operations on a inode.
 *
 * These are used in the form VOP_FOO(inode, args), which are macros
 * that expands to inode->inode_ops->vop_foo(inode, args). The operations
 * "foo" are:
 *
 *    vop_open        - Called on open() of a file. Can be used to
 *                      reject illegal or undesired open modes. Note that
 *                      various operations can be performed without the
 *                      file actually being opened.
 *                      The inode need not look at O_CREAT, O_EXCL, or
 *                      O_TRUNC, as these are handled in the VFS layer.
 *
 *                      VOP_EACHOPEN should not be called directly from
 *                      above the VFS layer - use vfs_open() to open inodes.
 *                      This maintains the open count so VOP_LASTCLOSE can
 *                      be called at the right time.
 *
 *    vop_close       - To be called on *last* close() of a file.
 *
 *                      VOP_LASTCLOSE should not be called directly from
 *                      above the VFS layer - use vfs_close() to close
 *                      inodes opened with vfs_open().
 *
 *    vop_reclaim     - Called when inode is no longer in use. Note that
 *                      this may be substantially after vop_lastclose is
 *                      called.
 *
 ****************************************
 *
 *    vop_read        - Read data from file to uio, at offset specified
 *                      in the uio, updating uio_resid to reflect the
 *                      amount read, and updating uio_offset to match.
 *                      Not allowed on directories or symlinks.
 *
 *    vop_getdirentry - Read a single filename from a directory into a
```

```
 *                      uio, choosing what name based on the offset
 *                      field in the uio, and updating that field.
 *                      Unlike with I/O on regular files, the value of
 *                      the offset field is not interpreted outside
 *                      the filesystem and thus need not be a byte
 *                      count. However, the uio_resid field should be
 *                      handled in the normal fashion.
 *                      On non-directory objects, return ENOTDIR.
 *
 *    vop_write       - Write data from uio to file at offset specified
 *                      in the uio, updating uio_resid to reflect the
 *                      amount written, and updating uio_offset to match.
 *                      Not allowed on directories or symlinks.
 *
 *    vop_ioctl       - Perform ioctl operation OP on file using data
 *                      DATA. The interpretation of the data is specific
 *                      to each ioctl.
 *
 *    vop_fstat        -Return info about a file. The pointer is a
 *                      pointer to struct stat; see stat.h.
 *
 *    vop_gettype     - Return type of file. The values for file types
 *                      are in sfs.h.
 *
 *    vop_tryseek     - Check if seeking to the specified position within
 *                      the file is legal. (For instance, all seeks
 *                      are illegal on serial port devices, and seeks
 *                      past EOF on files whose sizes are fixed may be
 *                      as well.)
 *
 *    vop_fsync       - Force any dirty buffers associated with this file
 *                      to stable storage.
 *
 *    vop_truncate    - Forcibly set size of file to the length passed
 *                      in, discarding any excess blocks.
 *
 *    vop_namefile    - Compute pathname relative to filesystem root
 *                      of the file and copy to the specified io buffer.
 *                      Need not work on objects that are not
 *                      directories.
 *
 ***************************************
 *
 *    vop_creat       - Create a regular file named NAME in the passed
 *                      directory DIR. If boolean EXCL is true, fail if
 *                      the file already exists; otherwise, use the
 *                      existing file if there is one. Hand back the
 *                      inode for the file as per vop_lookup.
 *
 ***************************************
 *
 *    vop_lookup      - Parse PATHNAME relative to the passed directory
 *                      DIR, and hand back the inode for the file it
 *                      refers to. May destroy PATHNAME. Should increment
 *                      refcount on inode handed back.
 */
struct inode_ops {
    unsigned long vop_magic;
    int (*vop_open)(struct inode *node, uint32_t open_flags);
    int (*vop_close)(struct inode *node);
    int (*vop_read)(struct inode *node, struct iobuf *iob);
    int (*vop_write)(struct inode *node, struct iobuf *iob);
    int (*vop_fstat)(struct inode *node, struct stat *stat);
    int (*vop_fsync)(struct inode *node);
    int (*vop_namefile)(struct inode *node, struct iobuf *iob);
```

```
    int (*vop_getdirentry)(struct inode *node, struct iobuf *iob);
    int (*vop_reclaim)(struct inode *node);
    int (*vop_gettype)(struct inode *node, uint32_t *type_store);
    int (*vop_tryseek)(struct inode *node, off_t pos);
    int (*vop_truncate)(struct inode *node, off_t len);
    int (*vop_create)(struct inode *node, const char *name, bool excl, struct inode **node_sto
re);
    int (*vop_lookup)(struct inode *node, char *path, struct inode **node_store);
    int (*vop_ioctl)(struct inode *node, int op, void *data);
};

/*
 * Consistency check
 */
void inode_check(struct inode *node, const char *opstr);

#define __vop_op(node, sym)                                             \
    ({                                                                  \
        struct inode *__node = (node);                                  \
        assert(__node != NULL && __node->in_ops != NULL && __node->in_ops->vop_##sym != NULL);  \
        inode_check(__node, #sym);                                      \
        __node->in_ops->vop_##sym;                                      \
     })

#define vop_open(node, open_flags)                                      (__vop_op(node, open)(node
, open_flags))
#define vop_close(node)                                                 (__vop_op(node, close)(nod
e))
#define vop_read(node, iob)                                             (__vop_op(node, read)(node
, iob))
#define vop_write(node, iob)                                            (__vop_op(node, write)(nod
e, iob))
#define vop_fstat(node, stat)                                           (__vop_op(node, fstat)(nod
e, stat))
#define vop_fsync(node)                                                 (__vop_op(node, fsync)(nod
e))
#define vop_namefile(node, iob)                                         (__vop_op(node, namefile)(
node, iob))
#define vop_getdirentry(node, iob)                                      (__vop_op(node, getdirentr
y)(node, iob))
#define vop_reclaim(node)                                               (__vop_op(node, reclaim)(n
ode))
#define vop_ioctl(node, op, data)                                       (__vop_op(node, ioctl)(nod
e, op, data))
#define vop_gettype(node, type_store)                                   (__vop_op(node, gettype)(n
ode, type_store))
#define vop_tryseek(node, pos)                                          (__vop_op(node, tryseek)(n
ode, pos))
#define vop_truncate(node, len)                                         (__vop_op(node, truncate)(
node, len))
#define vop_create(node, name, excl, node_store)                        (__vop_op(node, create)(no
de, name, excl, node_store))
#define vop_lookup(node, path, node_store)                              (__vop_op(node, lookup)(no
de, path, node_store))


#define vop_fs(node)                                                    ((node)->in_fs)
#define vop_init(node, ops, fs)                                         inode_init(node, ops, fs)
#define vop_kill(node)                                                  inode_kill(node)
```

```
/*
 * Reference count manipulation (handled above filesystem level)
 */
#define vop_ref_inc(node)                                       inode_ref_inc(node)
#define vop_ref_dec(node)                                       inode_ref_dec(node)
/*
 * Open count manipulation (handled above filesystem level)
 *
 * VOP_INCOPEN is called by vfs_open. VOP_DECOPEN is called by vfs_close.
 * Neither of these should need to be called from above the vfs layer.
 */
#define vop_open_inc(node)                                      inode_open_inc(node)
#define vop_open_dec(node)                                      inode_open_dec(node)


static inline int
inode_ref_count(struct inode *node) {
    return node->ref_count;
}

static inline int
inode_open_count(struct inode *node) {
    return node->open_count;
}

#endif /* !__KERN_FS_VFS_INODE_H__ */

[ucore/lab8_result/kern/fs/vfs/inode.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <string.h>
#include <atomic.h>
#include <vfs.h>
#include <inode.h>
#include <error.h>
#include <assert.h>
#include <kmalloc.h>

/* *
 * __alloc_inode - alloc a inode structure and initialize in_type
 * */
struct inode *
__alloc_inode(int type) {
    struct inode *node;
    if ((node = kmalloc(sizeof(struct inode))) != NULL) {
        node->in_type = type;
    }
    return node;
}

/* *
 * inode_init - initialize a inode structure
 * invoked by vop_init
 * */
void
inode_init(struct inode *node, const struct inode_ops *ops, struct fs *fs) {
    node->ref_count = 0;
    node->open_count = 0;
    node->in_ops = ops, node->in_fs = fs;
    vop_ref_inc(node);
}

/* *
 * inode_kill - kill a inode structure
```

```
 * invoked by vop_kill
 * */
void
inode_kill(struct inode *node) {
    assert(inode_ref_count(node) == 0);
    assert(inode_open_count(node) == 0);
    kfree(node);
}

/* *
 * inode_ref_inc - increment ref_count
 * invoked by vop_ref_inc
 * */
int
inode_ref_inc(struct inode *node) {
    node->ref_count += 1;
    return node->ref_count;
}

/* *
 * inode_ref_dec - decrement ref_count
 * invoked by vop_ref_dec
 * calls vop_reclaim if the ref_count hits zero
 * */
int
inode_ref_dec(struct inode *node) {
    assert(inode_ref_count(node) > 0);
    int ref_count, ret;
    node->ref_count-= 1;
    ref_count = node->ref_count;
    if (ref_count == 0) {
        if ((ret = vop_reclaim(node)) != 0 && ret != -E_BUSY) {
            cprintf("vfs: warning: vop_reclaim: %e.\n", ret);
        }
    }
    return ref_count;
}

/* *
 * inode_open_inc - increment the open_count
 * invoked by vop_open_inc
 * */
int
inode_open_inc(struct inode *node) {
    node->open_count += 1;
    return node->open_count;
}

/* *
 * inode_open_dec - decrement the open_count
 * invoked by vop_open_dec
 * calls vop_close if the open_count hits zero
 * */
int
inode_open_dec(struct inode *node) {
    assert(inode_open_count(node) > 0);
    int open_count, ret;
    node->open_count -= 1;
    open_count = node->open_count;
    if (open_count == 0) {
        if ((ret = vop_close(node)) != 0) {
            cprintf("vfs: warning: vop_close: %e.\n", ret);
        }
    }
    return open_count;
```

```
}

/* *
 * inode_check - check the various things being valid
 * called before all vop_* calls
 * */
void
inode_check(struct inode *node, const char *opstr) {
    assert(node != NULL && node->in_ops != NULL);
    assert(node->in_ops->vop_magic == VOP_MAGIC);
    int ref_count = inode_ref_count(node), open_count = inode_open_count(node);
    assert(ref_count >= open_count && open_count >= 0);
    assert(ref_count < MAX_INODE_COUNT && open_count < MAX_INODE_COUNT);
}
```

[ucore/lab8_result/kern/fs/vfs/vfs.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_FS_VFS_VFS_H__
#define __KERN_FS_VFS_VFS_H__

#include <defs.h>
#include <fs.h>
#include <sfs.h>

struct inode;   // abstract structure for an on-disk file (inode.h)
struct device;  // abstract structure for a device (dev.h)
struct iobuf;   // kernel or userspace I/O buffer (iobuf.h)

/*
 * Abstract filesystem. (Or device accessible as a file.)
 *
 * Information:
 *      fs_info    : filesystem-specific data (sfs_fs)
 *      fs_type    : filesystem type
 * Operations:
 *
 *      fs_sync       - Flush all dirty buffers to disk.
 *      fs_get_root   - Return root inode of filesystem.
 *      fs_unmount    - Attempt unmount of filesystem.
 *      fs_cleanup    - Cleanup of filesystem.???
 *
 *
 * fs_get_root should increment the refcount of the inode returned.
 * It should not ever return NULL.
 *
 * If fs_unmount returns an error, the filesystem stays mounted, and
 * consequently the struct fs instance should remain valid. On success,
 * however, the filesystem object and all storage associated with the
 * filesystem should have been discarded/released.
 *
 */
struct fs {
    union {
        struct sfs_fs __sfs_info;
    } fs_info;                                      // filesystem-specific data
    enum {
        fs_type_sfs_info,
    } fs_type;                                      // filesystem type
    int (*fs_sync)(struct fs *fs);                  // Flush all dirty buffers to disk
    struct inode *(*fs_get_root)(struct fs *fs);    // Return root inode of filesystem.
    int (*fs_unmount)(struct fs *fs);               // Attempt unmount of filesystem.
    void (*fs_cleanup)(struct fs *fs);              // Cleanup of filesystem.???
};

#define __fs_type(type)                                         fs_type_##type##_info
```

```
#define check_fs_type(fs, type)                                         ((fs)->fs_type == __fs_typ
e(type))

#define __fsop_info(_fs, type) ({                                       \
            struct fs *__fs = (_fs);                                    \
            assert(__fs != NULL && check_fs_type(__fs, type));          \
            &(__fs->fs_info.__##type##_info);                           \
        })

#define fsop_info(fs, type)                 __fsop_info(fs, type)

#define info2fs(info, type)                                             \
    to_struct((info), struct fs, fs_info.__##type##_info)

struct fs *__alloc_fs(int type);

#define alloc_fs(type)                                                  __alloc_fs(__fs_type(type)
)

// Macros to shorten the calling sequences.
#define fsop_sync(fs)                       ((fs)->fs_sync(fs))
#define fsop_get_root(fs)                   ((fs)->fs_get_root(fs))
#define fsop_unmount(fs)                    ((fs)->fs_unmount(fs))
#define fsop_cleanup(fs)                    ((fs)->fs_cleanup(fs))

/*
 * Virtual File System layer functions.
 *
 * The VFS layer translates operations on abstract on-disk files or
 * pathnames to operations on specific files on specific filesystems.
 */
void vfs_init(void);
void vfs_cleanup(void);
void vfs_devlist_init(void);

/*
 * VFS layer low-level operations.
 * See inode.h for direct operations on inodes.
 * See fs.h for direct operations on filesystems/devices.
 *
 *    vfs_set_curdir   - change current directory of current thread by inode
 *    vfs_get_curdir   - retrieve inode of current directory of current thread
 *    vfs_get_root     - get root inode for the filesystem named DEVNAME
 *    vfs_get_devname  - get mounted device name for the filesystem passed in
 */
int vfs_set_curdir(struct inode *dir);
int vfs_get_curdir(struct inode **dir_store);
int vfs_get_root(const char *devname, struct inode **root_store);
const char *vfs_get_devname(struct fs *fs);


/*
 * VFS layer high-level operations on pathnames
 * Because namei may destroy pathnames, these all may too.
 *
 *    vfs_open         - Open or create a file. FLAGS/MODE per the syscall.
 *    vfs_close  - Close a inode opened with vfs_open. Does not fail.
 *                 (See vfspath.c for a discussion of why.)
 *    vfs_link         - Create a hard link to a file.
 *    vfs_symlink      - Create a symlink PATH containing contents CONTENTS.
 *    vfs_readlink     - Read contents of a symlink into a uio.
 *    vfs_mkdir        - Create a directory. MODE per the syscall.
 *    vfs_unlink       - Delete a file/directory.
 *    vfs_rename       - rename a file.
 *    vfs_chdir  - Change current directory of current thread by name.
```

```
 *    vfs_getcwd - Retrieve name of current directory of current thread.
 *
 */
int vfs_open(char *path, uint32_t open_flags, struct inode **inode_store);
int vfs_close(struct inode *node);
int vfs_link(char *old_path, char *new_path);
int vfs_symlink(char *old_path, char *new_path);
int vfs_readlink(char *path, struct iobuf *iob);
int vfs_mkdir(char *path);
int vfs_unlink(char *path);
int vfs_rename(char *old_path, char *new_path);
int vfs_chdir(char *path);
int vfs_getcwd(struct iobuf *iob);


/*
 * VFS layer mid-level operations.
 *
 *    vfs_lookup     - Like VOP_LOOKUP, but takes a full device:path name,
 *                     or a name relative to the current directory, and
 *                     goes to the correct filesystem.
 *    vfs_lookparent - Likewise, for VOP_LOOKPARENT.
 *
 * Both of these may destroy the path passed in.
 */
int vfs_lookup(char *path, struct inode **node_store);
int vfs_lookup_parent(char *path, struct inode **node_store, char **endp);

/*
 * Misc
 *
 *    vfs_set_bootfs - Set the filesystem that paths beginning with a
 *                     slash are sent to. If not set, these paths fail
 *                     with ENOENT. The argument should be the device
 *                     name or volume name for the filesystem (such as
 *                     "lhd0:") but need not have the trailing colon.
 *
 *    vfs_get_bootfs - return the inode of the bootfs filesystem.
 *
 *    vfs_add_fs     - Add a hardwired filesystem to the VFS named device
 *                     list. It will be accessible as "devname:". This is
 *                     intended for filesystem-devices like emufs, and
 *                     gizmos like Linux procfs or BSD kernfs, not for
 *                     mounting filesystems on disk devices.
 *
 *    vfs_add_dev    - Add a device to the VFS named device list. If
 *                     MOUNTABLE is zero, the device will be accessible
 *                     as "DEVNAME:". If the mountable flag is set, the
 *                     device will be accessible as "DEVNAMEraw:" and
 *                     mountable under the name "DEVNAME". Thus, the
 *                     console, added with MOUNTABLE not set, would be
 *                     accessed by pathname as "con:", and lhd0, added
 *                     with mountable set, would be accessed by
 *                     pathname as "lhd0raw:" and mounted by passing
 *                     "lhd0" to vfs_mount.
 *
 *    vfs_mount      - Attempt to mount a filesystem on a device. The
 *                     device named by DEVNAME will be looked up and
 *                     passed, along with DATA, to the supplied function
 *                     MOUNTFUNC, which should create a struct fs and
 *                     return it in RESULT.
 *
 *    vfs_unmount    - Unmount the filesystem presently mounted on the
 *                     specified device.
 *
```

```
 *    vfs_unmountall - Unmount all mounted filesystems.
 */
int vfs_set_bootfs(char *fsname);
int vfs_get_bootfs(struct inode **node_store);

int vfs_add_fs(const char *devname, struct fs *fs);
int vfs_add_dev(const char *devname, struct inode *devnode, bool mountable);

int vfs_mount(const char *devname, int (*mountfunc)(struct device *dev, struct fs **fs_store))
;
int vfs_unmount(const char *devname);
int vfs_unmount_all(void);

#endif /* !__KERN_FS_VFS_VFS_H__ */

[ucore/lab8_result/kern/fs/file.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_FS_FILE_H__
#define __KERN_FS_FILE_H__

//#include <types.h>
#include <fs.h>
#include <proc.h>
#include <atomic.h>
#include <assert.h>

struct inode;
struct stat;
struct dirent;

struct file {
    enum {
        FD_NONE, FD_INIT, FD_OPENED, FD_CLOSED,
    } status;
    bool readable;
    bool writable;
    int fd;
    off_t pos;
    struct inode *node;
    int open_count;
};

void fd_array_init(struct file *fd_array);
void fd_array_open(struct file *file);
void fd_array_close(struct file *file);
void fd_array_dup(struct file *to, struct file *from);
bool file_testfd(int fd, bool readable, bool writable);

int file_open(char *path, uint32_t open_flags);
int file_close(int fd);
int file_read(int fd, void *base, size_t len, size_t *copied_store);
int file_write(int fd, void *base, size_t len, size_t *copied_store);
int file_seek(int fd, off_t pos, int whence);
int file_fstat(int fd, struct stat *stat);
int file_fsync(int fd);
int file_getdirentry(int fd, struct dirent *dirent);
int file_dup(int fd1, int fd2);
int file_pipe(int fd[]);
int file_mkfifo(const char *name, uint32_t open_flags);

static inline int
fopen_count(struct file *file) {
    return file->open_count;
}

static inline int
```

```
fopen_count_inc(struct file *file) {
    file->open_count += 1;
    return file->open_count;
}

static inline int
fopen_count_dec(struct file *file) {
    file->open_count -= 1;
    return file->open_count;
}

#endif /* !__KERN_FS_FILE_H__ */

[ucore/lab8_result/kern/fs/swap/swapfs.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_FS_SWAP_SWAPFS_H__
#define __KERN_FS_SWAP_SWAPFS_H__

#include <memlayout.h>
#include <swap.h>

void swapfs_init(void);
int swapfs_read(swap_entry_t entry, struct Page *page);
int swapfs_write(swap_entry_t entry, struct Page *page);

#endif /* !__KERN_FS_SWAP_SWAPFS_H__ */

[ucore/lab8_result/kern/fs/swap/swapfs.c] ++++++++++++++++++++++++++++++
#include <swap.h>
#include <swapfs.h>
#include <mmu.h>
#include <fs.h>
#include <ide.h>
#include <pmm.h>
#include <assert.h>

void
swapfs_init(void) {
    static_assert((PGSIZE % SECTSIZE) == 0);
    if (!ide_device_valid(SWAP_DEV_NO)) {
        panic("swap fs isn't available.\n");
    }
    max_swap_offset = ide_device_size(SWAP_DEV_NO) / (PGSIZE / SECTSIZE);
}

int
swapfs_read(swap_entry_t entry, struct Page *page) {
    return ide_read_secs(SWAP_DEV_NO, swap_offset(entry) * PAGE_NSECT, page2kva(page), PAGE_NS
ECT);
}

int
swapfs_write(swap_entry_t entry, struct Page *page) {
    return ide_write_secs(SWAP_DEV_NO, swap_offset(entry) * PAGE_NSECT, page2kva(page), PAGE_N
SECT);
}

[ucore/lab8_result/kern/fs/fs.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_FS_FS_H__
#define __KERN_FS_FS_H__

#include <defs.h>
#include <mmu.h>
#include <sem.h>
#include <atomic.h>
```

```
#define SECTSIZE            512
#define PAGE_NSECT          (PGSIZE / SECTSIZE)

#define SWAP_DEV_NO         1
#define DISK0_DEV_NO        2
#define DISK1_DEV_NO        3

void fs_init(void);
void fs_cleanup(void);

struct inode;
struct file;

/*
 * process's file related information
 */
struct files_struct {
    struct inode *pwd;      // inode of present working directory
    struct file *fd_array;  // opened files array
    int files_count;        // the number of opened files
    semaphore_t files_sem;  // lock protect sem
};

#define FILES_STRUCT_BUFSIZE                        (PGSIZE - sizeof(struct files_struct))
#define FILES_STRUCT_NENTRY                         (FILES_STRUCT_BUFSIZE / sizeof(struct file)
)

void lock_files(struct files_struct *filesp);
void unlock_files(struct files_struct *filesp);

struct files_struct *files_create(void);
void files_destroy(struct files_struct *filesp);
void files_closeall(struct files_struct *filesp);
int dup_files(struct files_struct *to, struct files_struct *from);

static inline int
files_count(struct files_struct *filesp) {
    return filesp->files_count;
}

static inline int
files_count_inc(struct files_struct *filesp) {
    filesp->files_count += 1;
    return filesp->files_count;
}

static inline int
files_count_dec(struct files_struct *filesp) {
    filesp->files_count -= 1;
    return filesp->files_count;
}

#endif /* !__KERN_FS_FS_H__ */

[ucore/lab8_result/kern/fs/sfs/sfs.h] ++++++++++++++++++++++++++++++++
#ifndef __KERN_FS_SFS_SFS_H__
#define __KERN_FS_SFS_SFS_H__

#include <defs.h>
#include <mmu.h>
#include <list.h>
#include <sem.h>
#include <unistd.h>

/*
```

```
 * Simple FS (SFS) definitions visible to ucore. This covers the on-disk format
 * and is used by tools that work on SFS volumes, such as mksfs.
 */

#define SFS_MAGIC                                   0x2f8dbe2a              /* magic number fo
r sfs */
#define SFS_BLKSIZE                                 PGSIZE                  /* size of block *
/
#define SFS_NDIRECT                                 12                      /* # of direct blo
cks in inode */
#define SFS_MAX_INFO_LEN                            31                      /* max length of i
nfomation */
#define SFS_MAX_FNAME_LEN                           FS_MAX_FNAME_LEN        /* max length of f
ilename */
#define SFS_MAX_FILE_SIZE                           (1024UL * 1024 * 128)   /* max file size (
128M) */
#define SFS_BLKN_SUPER                              0                       /* block the super
block lives in */
#define SFS_BLKN_ROOT                               1                       /* location of the
 root dir inode */
#define SFS_BLKN_FREEMAP                            2                       /* 1st block of th
e freemap */

/* # of bits in a block */
#define SFS_BLKBITS                                 (SFS_BLKSIZE * CHAR_BIT)

/* # of entries in a block */
#define SFS_BLK_NENTRY                              (SFS_BLKSIZE / sizeof(uint32_t))

/* file types */
#define SFS_TYPE_INVAL                              0       /* Should not appear on disk */
#define SFS_TYPE_FILE                               1
#define SFS_TYPE_DIR                                2
#define SFS_TYPE_LINK                               3

/*
 * On-disk superblock
 */
struct sfs_super {
    uint32_t magic;                                 /* magic number, should be SFS_MAGIC */
    uint32_t blocks;                                /* # of blocks in fs */
    uint32_t unused_blocks;                         /* # of unused blocks in fs */
    char info[SFS_MAX_INFO_LEN + 1];                /* infomation for sfs  */
};

/* inode (on disk) */
struct sfs_disk_inode {
    uint32_t size;                                  /* size of the file (in bytes) */
    uint16_t type;                                  /* one of SYS_TYPE_* above */
    uint16_t nlinks;                                /* # of hard links to this file */
    uint32_t blocks;                                /* # of blocks */
    uint32_t direct[SFS_NDIRECT];                   /* direct blocks */
    uint32_t indirect;                              /* indirect blocks */
//    uint32_t db_indirect;                           /* double indirect blocks */
//  unused
};

/* file entry (on disk) */
struct sfs_disk_entry {
    uint32_t ino;                                   /* inode number */
    char name[SFS_MAX_FNAME_LEN + 1];               /* file name */
};

#define sfs_dentry_size                             \
    sizeof(((struct sfs_disk_entry *)0)->name)
```

```
/* inode for sfs */
struct sfs_inode {
    struct sfs_disk_inode *din;                     /* on-disk inode */
    uint32_t ino;                                   /* inode number */
    bool dirty;                                     /* true if inode modified */
    int reclaim_count;                              /* kill inode if it hits zero */
    semaphore_t sem;                                /* semaphore for din */
    list_entry_t inode_link;                        /* entry for linked-list in sfs_fs */
    list_entry_t hash_link;                         /* entry for hash linked-list in sfs_fs */
};

#define le2sin(le, member)                          \
    to_struct((le), struct sfs_inode, member)

/* filesystem for sfs */
struct sfs_fs {
    struct sfs_super super;                         /* on-disk superblock */
    struct device *dev;                             /* device mounted on */
    struct bitmap *freemap;                         /* blocks in use are mared 0 */
    bool super_dirty;                               /* true if super/freemap modified */
    void *sfs_buffer;                               /* buffer for non-block aligned io */
    semaphore_t fs_sem;                             /* semaphore for fs */
    semaphore_t io_sem;                             /* semaphore for io */
    semaphore_t mutex_sem;                          /* semaphore for link/unlink and rename */
    list_entry_t inode_list;                        /* inode linked-list */
    list_entry_t *hash_list;                        /* inode hash linked-list */
};

/* hash for sfs */
#define SFS_HLIST_SHIFT                             10
#define SFS_HLIST_SIZE                              (1 << SFS_HLIST_SHIFT)
#define sin_hashfn(x)                               (hash32(x, SFS_HLIST_SHIFT))

/* size of freemap (in bits) */
#define sfs_freemap_bits(super)                     ROUNDUP((super)->blocks, SFS_BLKBITS)

/* size of freemap (in blocks) */
#define sfs_freemap_blocks(super)                   ROUNDUP_DIV((super)->blocks, SFS_BLKBITS)

struct fs;
struct inode;

void sfs_init(void);
int sfs_mount(const char *devname);

void lock_sfs_fs(struct sfs_fs *sfs);
void lock_sfs_io(struct sfs_fs *sfs);
void unlock_sfs_fs(struct sfs_fs *sfs);
void unlock_sfs_io(struct sfs_fs *sfs);

int sfs_rblock(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks);
int sfs_wblock(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks);
int sfs_rbuf(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset);
int sfs_wbuf(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset);
int sfs_sync_super(struct sfs_fs *sfs);
int sfs_sync_freemap(struct sfs_fs *sfs);
int sfs_clear_block(struct sfs_fs *sfs, uint32_t blkno, uint32_t nblks);

int sfs_load_inode(struct sfs_fs *sfs, struct inode **node_store, uint32_t ino);

#endif /* !__KERN_FS_SFS_SFS_H__ */

[ucore/lab8_result/kern/fs/sfs/sfs_lock.c] ++++++++++++++++++++++++++++++
#include <defs.h>
```

```
#include <sem.h>
#include <sfs.h>


/*
 * lock_sfs_fs - lock the process of  SFS Filesystem Rd/Wr Disk Block
 *
 * called by: sfs_load_inode, sfs_sync, sfs_reclaim
 */
void
lock_sfs_fs(struct sfs_fs *sfs) {
    down(&(sfs->fs_sem));
}

/*
 * lock_sfs_io - lock the process of SFS File Rd/Wr Disk Block
 *
 * called by: sfs_rwblock, sfs_clear_block, sfs_sync_super
 */
void
lock_sfs_io(struct sfs_fs *sfs) {
    down(&(sfs->io_sem));
}

/*
 * unlock_sfs_fs - unlock the process of  SFS Filesystem Rd/Wr Disk Block
 *
 * called by: sfs_load_inode, sfs_sync, sfs_reclaim
 */
void
unlock_sfs_fs(struct sfs_fs *sfs) {
    up(&(sfs->fs_sem));
}

/*
 * unlock_sfs_io - unlock the process of sfs Rd/Wr Disk Block
 *
 * called by: sfs_rwblock sfs_clear_block sfs_sync_super
 */
void
unlock_sfs_io(struct sfs_fs *sfs) {
    up(&(sfs->io_sem));
}
[ucore/lab8_result/kern/fs/sfs/bitmap.c] ++++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <bitmap.h>
#include <kmalloc.h>
#include <error.h>
#include <assert.h>

#define WORD_TYPE           uint32_t
#define WORD_BITS           (sizeof(WORD_TYPE) * CHAR_BIT)

struct bitmap {
    uint32_t nbits;
    uint32_t nwords;
    WORD_TYPE *map;
};

// bitmap_create - allocate a new bitmap object.
struct bitmap *
bitmap_create(uint32_t nbits) {
    static_assert(WORD_BITS != 0);
    assert(nbits != 0 && nbits + WORD_BITS > nbits);
```

```
    struct bitmap *bitmap;
    if ((bitmap = kmalloc(sizeof(struct bitmap))) == NULL) {
        return NULL;
    }

    uint32_t nwords = ROUNDUP_DIV(nbits, WORD_BITS);
    WORD_TYPE *map;
    if ((map = kmalloc(sizeof(WORD_TYPE) * nwords)) == NULL) {
        kfree(bitmap);
        return NULL;
    }

    bitmap->nbits = nbits, bitmap->nwords = nwords;
    bitmap->map = memset(map, 0xFF, sizeof(WORD_TYPE) * nwords);

    /* mark any leftover bits at the end in use(0) */
    if (nbits != nwords * WORD_BITS) {
        uint32_t ix = nwords - 1, overbits = nbits - ix * WORD_BITS;

        assert(nbits / WORD_BITS == ix);
        assert(overbits > 0 && overbits < WORD_BITS);

        for (; overbits < WORD_BITS; overbits ++) {
            bitmap->map[ix] ^= (1 << overbits);
        }
    }
    return bitmap;
}

// bitmap_alloc - locate a cleared bit, set it, and return its index.
int
bitmap_alloc(struct bitmap *bitmap, uint32_t *index_store) {
    WORD_TYPE *map = bitmap->map;
    uint32_t ix, offset, nwords = bitmap->nwords;
    for (ix = 0; ix < nwords; ix ++) {
        if (map[ix] != 0) {
            for (offset = 0; offset < WORD_BITS; offset ++) {
                WORD_TYPE mask = (1 << offset);
                if (map[ix] & mask) {
                    map[ix] ^= mask;
                    *index_store = ix * WORD_BITS + offset;
                    return 0;
                }
            }
            assert(0);
        }
    }
    return -E_NO_MEM;
}

// bitmap_translate - according index, get the related word and mask
static void
bitmap_translate(struct bitmap *bitmap, uint32_t index, WORD_TYPE **word, WORD_TYPE *mask) {
    assert(index < bitmap->nbits);
    uint32_t ix = index / WORD_BITS, offset = index % WORD_BITS;
    *word = bitmap->map + ix;
    *mask = (1 << offset);
}

// bitmap_test - according index, get the related value (0 OR 1) in the bitmap
bool
bitmap_test(struct bitmap *bitmap, uint32_t index) {
    WORD_TYPE *word, mask;
    bitmap_translate(bitmap, index, &word, &mask);
```

```
    return (*word & mask);
}

// bitmap_free - according index, set related bit to 1
void
bitmap_free(struct bitmap *bitmap, uint32_t index) {
    WORD_TYPE *word, mask;
    bitmap_translate(bitmap, index, &word, &mask);
    assert(!(*word & mask));
    *word |= mask;
}

// bitmap_destroy - free memory contains bitmap
void
bitmap_destroy(struct bitmap *bitmap) {
    kfree(bitmap->map);
    kfree(bitmap);
}

// bitmap_getdata - return bitmap->map, return the length of bits to len_store
void *
bitmap_getdata(struct bitmap *bitmap, size_t *len_store) {
    if (len_store != NULL) {
        *len_store = sizeof(WORD_TYPE) * bitmap->nwords;
    }
    return bitmap->map;
}

[ucore/lab8_result/kern/fs/sfs/sfs_fs.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <string.h>
#include <kmalloc.h>
#include <list.h>
#include <fs.h>
#include <vfs.h>
#include <dev.h>
#include <sfs.h>
#include <inode.h>
#include <iobuf.h>
#include <bitmap.h>
#include <error.h>
#include <assert.h>

/*
 * sfs_sync - sync sfs's superblock and freemap in memroy into disk
 */
static int
sfs_sync(struct fs *fs) {
    struct sfs_fs *sfs = fsop_info(fs, sfs);
    lock_sfs_fs(sfs);
    {
        list_entry_t *list = &(sfs->inode_list), *le = list;
        while ((le = list_next(le)) != list) {
            struct sfs_inode *sin = le2sin(le, inode_link);
            vop_fsync(info2node(sin, sfs_inode));
        }
    }
    unlock_sfs_fs(sfs);

    int ret;
    if (sfs->super_dirty) {
        sfs->super_dirty = 0;
        if ((ret = sfs_sync_super(sfs)) != 0) {
            sfs->super_dirty = 1;
```

```
            return ret;
        }
        if ((ret = sfs_sync_freemap(sfs)) != 0) {
            sfs->super_dirty = 1;
            return ret;
        }
    }
    return 0;
}

/*
 * sfs_get_root - get the root directory inode  from disk (SFS_BLKN_ROOT,1)
 */
static struct inode *
sfs_get_root(struct fs *fs) {
    struct inode *node;
    int ret;
    if ((ret = sfs_load_inode(fsop_info(fs, sfs), &node, SFS_BLKN_ROOT)) != 0) {
        panic("load sfs root failed: %e", ret);
    }
    return node;
}

/*
 * sfs_unmount - unmount sfs, and free the memorys contain sfs->freemap/sfs_buffer/hash_liskt
and sfs itself.
 */
static int
sfs_unmount(struct fs *fs) {
    struct sfs_fs *sfs = fsop_info(fs, sfs);
    if (!list_empty(&(sfs->inode_list))) {
        return -E_BUSY;
    }
    assert(!sfs->super_dirty);
    bitmap_destroy(sfs->freemap);
    kfree(sfs->sfs_buffer);
    kfree(sfs->hash_list);
    kfree(sfs);
    return 0;
}

/*
 * sfs_cleanup - when sfs failed, then should call this function to sync sfs by calling sfs_sy
nc
 *
 * NOTICE: nouse now.
 */
static void
sfs_cleanup(struct fs *fs) {
    struct sfs_fs *sfs = fsop_info(fs, sfs);
    uint32_t blocks = sfs->super.blocks, unused_blocks = sfs->super.unused_blocks;
    cprintf("sfs: cleanup: '%s' (%d/%d/%d)\n", sfs->super.info,
            blocks - unused_blocks, unused_blocks, blocks);
    int i, ret;
    for (i = 0; i < 32; i ++) {
        if ((ret = fsop_sync(fs)) == 0) {
            break;
        }
    }
    if (ret != 0) {
        warn("sfs: sync error: '%s': %e.\n", sfs->super.info, ret);
    }
}

/*
```

```
 * sfs_init_read - used in sfs_do_mount to read disk block(blkno, 1) directly.
 *
 * @dev:        the block device
 * @blkno:      the NO. of disk block
 * @blk_buffer: the buffer used for read
 *
 *      (1) init iobuf
 *      (2) read dev into iobuf
 */
static int
sfs_init_read(struct device *dev, uint32_t blkno, void *blk_buffer) {
    struct iobuf __iob, *iob = iobuf_init(&__iob, blk_buffer, SFS_BLKSIZE, blkno * SFS_BLKSIZE
);
    return dop_io(dev, iob, 0);
}

/*
 * sfs_init_freemap - used in sfs_do_mount to read freemap data info in disk block(blkno, nblk
s) directly.
 *
 * @dev:        the block device
 * @bitmap:     the bitmap in memroy
 * @blkno:      the NO. of disk block
 * @nblks:      Rd number of disk block
 * @blk_buffer: the buffer used for read
 *
 *      (1) get data addr in bitmap
 *      (2) read dev into iobuf
 */
static int
sfs_init_freemap(struct device *dev, struct bitmap *freemap, uint32_t blkno, uint32_t nblks, v
oid *blk_buffer) {
    size_t len;
    void *data = bitmap_getdata(freemap, &len);
    assert(data != NULL && len == nblks * SFS_BLKSIZE);
    while (nblks != 0) {
        int ret;
        if ((ret = sfs_init_read(dev, blkno, data)) != 0) {
            return ret;
        }
        blkno ++, nblks --, data += SFS_BLKSIZE;
    }
    return 0;
}

/*
 * sfs_do_mount - mount sfs file system.
 *
 * @dev:        the block device contains sfs file system
 * @fs_store:   the fs struct in memroy
 */
static int
sfs_do_mount(struct device *dev, struct fs **fs_store) {
    static_assert(SFS_BLKSIZE >= sizeof(struct sfs_super));
    static_assert(SFS_BLKSIZE >= sizeof(struct sfs_disk_inode));
    static_assert(SFS_BLKSIZE >= sizeof(struct sfs_disk_entry));

    if (dev->d_blocksize != SFS_BLKSIZE) {
        return -E_NA_DEV;
    }

    /* allocate fs structure */
    struct fs *fs;
    if ((fs = alloc_fs(sfs)) == NULL) {
        return -E_NO_MEM;
```

```
    }
    struct sfs_fs *sfs = fsop_info(fs, sfs);
    sfs->dev = dev;

    int ret = -E_NO_MEM;

    void *sfs_buffer;
    if ((sfs->sfs_buffer = sfs_buffer = kmalloc(SFS_BLKSIZE)) == NULL) {
        goto failed_cleanup_fs;
    }

    /* load and check superblock */
    if ((ret = sfs_init_read(dev, SFS_BLKN_SUPER, sfs_buffer)) != 0) {
        goto failed_cleanup_sfs_buffer;
    }

    ret = -E_INVAL;

    struct sfs_super *super = sfs_buffer;
    if (super->magic != SFS_MAGIC) {
        cprintf("sfs: wrong magic in superblock. (%08x should be %08x).\n",
                super->magic, SFS_MAGIC);
        goto failed_cleanup_sfs_buffer;
    }
    if (super->blocks > dev->d_blocks) {
        cprintf("sfs: fs has %u blocks, device has %u blocks.\n",
                super->blocks, dev->d_blocks);
        goto failed_cleanup_sfs_buffer;
    }
    super->info[SFS_MAX_INFO_LEN] = '\0';
    sfs->super = *super;

    ret = -E_NO_MEM;

    uint32_t i;

    /* alloc and initialize hash list */
    list_entry_t *hash_list;
    if ((sfs->hash_list = hash_list = kmalloc(sizeof(list_entry_t) * SFS_HLIST_SIZE)) == NULL)
 {
        goto failed_cleanup_sfs_buffer;
    }
    for (i = 0; i < SFS_HLIST_SIZE; i ++) {
        list_init(hash_list + i);
    }

    /* load and check freemap */
    struct bitmap *freemap;
    uint32_t freemap_size_nbits = sfs_freemap_bits(super);
    if ((sfs->freemap = freemap = bitmap_create(freemap_size_nbits)) == NULL) {
        goto failed_cleanup_hash_list;
    }
    uint32_t freemap_size_nblks = sfs_freemap_blocks(super);
    if ((ret = sfs_init_freemap(dev, freemap, SFS_BLKN_FREEMAP, freemap_size_nblks, sfs_buffer
)) != 0) {
        goto failed_cleanup_freemap;
    }

    uint32_t blocks = sfs->super.blocks, unused_blocks = 0;
    for (i = 0; i < freemap_size_nbits; i ++) {
        if (bitmap_test(freemap, i)) {
            unused_blocks ++;
        }
    }
    assert(unused_blocks == sfs->super.unused_blocks);
```

```
    /* and other fields */
    sfs->super_dirty = 0;
    sem_init(&(sfs->fs_sem), 1);
    sem_init(&(sfs->io_sem), 1);
    sem_init(&(sfs->mutex_sem), 1);
    list_init(&(sfs->inode_list));
    cprintf("sfs: mount: '%s' (%d/%d/%d)\n", sfs->super.info,
            blocks - unused_blocks, unused_blocks, blocks);

    /* link addr of sync/get_root/unmount/cleanup funciton  fs's function pointers*/
    fs->fs_sync = sfs_sync;
    fs->fs_get_root = sfs_get_root;
    fs->fs_unmount = sfs_unmount;
    fs->fs_cleanup = sfs_cleanup;
    *fs_store = fs;
    return 0;

failed_cleanup_freemap:
    bitmap_destroy(freemap);
failed_cleanup_hash_list:
    kfree(hash_list);
failed_cleanup_sfs_buffer:
    kfree(sfs_buffer);
failed_cleanup_fs:
    kfree(fs);
    return ret;
}

int
sfs_mount(const char *devname) {
    return vfs_mount(devname, sfs_do_mount);
}

[ucore/lab8_result/kern/fs/sfs/sfs_io.c] ++++++++++++++++++++++++++++++++
#include <defs.h>
#include <string.h>
#include <dev.h>
#include <sfs.h>
#include <iobuf.h>
#include <bitmap.h>
#include <assert.h>

//Basic block-level I/O routines

/* sfs_rwblock_nolock - Basic block-level I/O routine for Rd/Wr one disk block,
 *                      without lock protect for mutex process on Rd/Wr disk block
 * @sfs:   sfs_fs which will be process
 * @buf:   the buffer uesed for Rd/Wr
 * @blkno: the NO. of disk block
 * @write: BOOL: Read or Write
 * @check: BOOL: if check (blono < sfs super.blocks)
 */
static int
sfs_rwblock_nolock(struct sfs_fs *sfs, void *buf, uint32_t blkno, bool write, bool check) {
    assert((blkno != 0 || !check) && blkno < sfs->super.blocks);
    struct iobuf __iob, *iob = iobuf_init(&__iob, buf, SFS_BLKSIZE, blkno * SFS_BLKSIZE);
    return dop_io(sfs->dev, iob, write);
}

/* sfs_rwblock - Basic block-level I/O routine for Rd/Wr N disk blocks ,
 *               with lock protect for mutex process on Rd/Wr disk block
 * @sfs:   sfs_fs which will be process
 * @buf:   the buffer uesed for Rd/Wr
 * @blkno: the NO. of disk block
```

```
 * @nblks: Rd/Wr number of disk block
 * @write: BOOL: Read - 0 or Write - 1
 */
static int
sfs_rwblock(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks, bool write) {
    int ret = 0;
    lock_sfs_io(sfs);
    {
        while (nblks != 0) {
            if ((ret = sfs_rwblock_nolock(sfs, buf, blkno, write, 1)) != 0) {
                break;
            }
            blkno ++, nblks --;
            buf += SFS_BLKSIZE;
        }
    }
    unlock_sfs_io(sfs);
    return ret;
}

/* sfs_rblock - The Wrap of sfs_rwblock function for Rd N disk blocks ,
 *
 * @sfs:   sfs_fs which will be process
 * @buf:   the buffer uesed for Rd/Wr
 * @blkno: the NO. of disk block
 * @nblks: Rd/Wr number of disk block
 */
int
sfs_rblock(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks) {
    return sfs_rwblock(sfs, buf, blkno, nblks, 0);
}

/* sfs_wblock - The Wrap of sfs_rwblock function for Wr N disk blocks ,
 *
 * @sfs:   sfs_fs which will be process
 * @buf:   the buffer uesed for Rd/Wr
 * @blkno: the NO. of disk block
 * @nblks: Rd/Wr number of disk block
 */
int
sfs_wblock(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks) {
    return sfs_rwblock(sfs, buf, blkno, nblks, 1);
}

/* sfs_rbuf - The Basic block-level I/O routine for  Rd( non-block & non-aligned io) one disk
block(using sfs->sfs_buffer)
 *            with lock protect for mutex process on Rd/Wr disk block
 * @sfs:    sfs_fs which will be process
 * @buf:    the buffer uesed for Rd
 * @len:    the length need to Rd
 * @blkno:  the NO. of disk block
 * @offset: the offset in the content of disk block
 */
int
sfs_rbuf(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset) {
    assert(offset >= 0 && offset < SFS_BLKSIZE && offset + len <= SFS_BLKSIZE);
    int ret;
    lock_sfs_io(sfs);
    {
        if ((ret = sfs_rwblock_nolock(sfs, sfs->sfs_buffer, blkno, 0, 1)) == 0) {
            memcpy(buf, sfs->sfs_buffer + offset, len);
        }
    }
    unlock_sfs_io(sfs);
    return ret;
```

```
}

/* sfs_wbuf - The Basic block-level I/O routine for  Wr( non-block & non-aligned io) one disk block(using sfs->sfs_buffer)
 *            with lock protect for mutex process on Rd/Wr disk block
 * @sfs:    sfs_fs which will be process
 * @buf:    the buffer uesed for Wr
 * @len:    the length need to Wr
 * @blkno:  the NO. of disk block
 * @offset: the offset in the content of disk block
 */
int
sfs_wbuf(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset) {
    assert(offset >= 0 && offset < SFS_BLKSIZE && offset + len <= SFS_BLKSIZE);
    int ret;
    lock_sfs_io(sfs);
    {
        if ((ret = sfs_rwblock_nolock(sfs, sfs->sfs_buffer, blkno, 0, 1)) == 0) {
            memcpy(sfs->sfs_buffer + offset, buf, len);
            ret = sfs_rwblock_nolock(sfs, sfs->sfs_buffer, blkno, 1, 1);
        }
    }
    unlock_sfs_io(sfs);
    return ret;
}

/*
 * sfs_sync_super - write sfs->super (in memory) into disk (SFS_BLKN_SUPER, 1) with lock protect.
 */
int
sfs_sync_super(struct sfs_fs *sfs) {
    int ret;
    lock_sfs_io(sfs);
    {
        memset(sfs->sfs_buffer, 0, SFS_BLKSIZE);
        memcpy(sfs->sfs_buffer, &(sfs->super), sizeof(sfs->super));
        ret = sfs_rwblock_nolock(sfs, sfs->sfs_buffer, SFS_BLKN_SUPER, 1, 0);
    }
    unlock_sfs_io(sfs);
    return ret;
}

/*
 * sfs_sync_freemap - write sfs bitmap into disk (SFS_BLKN_FREEMAP, nblks)  without lock protect.
 */
int
sfs_sync_freemap(struct sfs_fs *sfs) {
    uint32_t nblks = sfs_freemap_blocks(&(sfs->super));
    return sfs_wblock(sfs, bitmap_getdata(sfs->freemap, NULL), SFS_BLKN_FREEMAP, nblks);
}

/*
 * sfs_clear_block - write zero info into disk (blkno, nblks)  with lock protect.
 * @sfs:   sfs_fs which will be process
 * @blkno: the NO. of disk block
 * @nblks: Rd/Wr number of disk block
 */
int
sfs_clear_block(struct sfs_fs *sfs, uint32_t blkno, uint32_t nblks) {
    int ret;
    lock_sfs_io(sfs);
    {
        memset(sfs->sfs_buffer, 0, SFS_BLKSIZE);
```

```
        while (nblks != 0) {
            if ((ret = sfs_rwblock_nolock(sfs, sfs->sfs_buffer, blkno, 1, 1)) != 0) {
                break;
            }
            blkno ++, nblks --;
        }
    }
    unlock_sfs_io(sfs);
    return ret;
}
```

[ucore/lab8_result/kern/fs/sfs/sfs_inode.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <string.h>
#include <stdlib.h>
#include <list.h>
#include <stat.h>
#include <kmalloc.h>
#include <vfs.h>
#include <dev.h>
#include <sfs.h>
#include <inode.h>
#include <iobuf.h>
#include <bitmap.h>
#include <error.h>
#include <assert.h>

static const struct inode_ops sfs_node_dirops;  // dir operations
static const struct inode_ops sfs_node_fileops; // file operations

/*
 * lock_sin - lock the process of inode Rd/Wr
 */
static void
lock_sin(struct sfs_inode *sin) {
    down(&(sin->sem));
}

/*
 * unlock_sin - unlock the process of inode Rd/Wr
 */
static void
unlock_sin(struct sfs_inode *sin) {
    up(&(sin->sem));
}

/*
 * sfs_get_ops - return function addr of fs_node_dirops/sfs_node_fileops
 */
static const struct inode_ops *
sfs_get_ops(uint16_t type) {
    switch (type) {
    case SFS_TYPE_DIR:
        return &sfs_node_dirops;
    case SFS_TYPE_FILE:
        return &sfs_node_fileops;
    }
    panic("invalid file type %d.\n", type);
}

/*
 * sfs_hash_list - return inode entry in sfs->hash_list
 */
static list_entry_t *
sfs_hash_list(struct sfs_fs *sfs, uint32_t ino) {
```

```
    return sfs->hash_list + sin_hashfn(ino);
}

/*
 * sfs_set_links - link inode sin in sfs->linked-list AND sfs->hash_link
 */
static void
sfs_set_links(struct sfs_fs *sfs, struct sfs_inode *sin) {
    list_add(&(sfs->inode_list), &(sin->inode_link));
    list_add(sfs_hash_list(sfs, sin->ino), &(sin->hash_link));
}

/*
 * sfs_remove_links - unlink inode sin in sfs->linked-list AND sfs->hash_link
 */
static void
sfs_remove_links(struct sfs_inode *sin) {
    list_del(&(sin->inode_link));
    list_del(&(sin->hash_link));
}

/*
 * sfs_block_inuse - check the inode with NO. ino inuse info in bitmap
 */
static bool
sfs_block_inuse(struct sfs_fs *sfs, uint32_t ino) {
    if (ino != 0 && ino < sfs->super.blocks) {
        return !bitmap_test(sfs->freemap, ino);
    }
    panic("sfs_block_inuse: called out of range (0, %u) %u.\n", sfs->super.blocks, ino);
}

/*
 * sfs_block_alloc -  check and get a free disk block
 */
static int
sfs_block_alloc(struct sfs_fs *sfs, uint32_t *ino_store) {
    int ret;
    if ((ret = bitmap_alloc(sfs->freemap, ino_store)) != 0) {
        return ret;
    }
    assert(sfs->super.unused_blocks > 0);
    sfs->super.unused_blocks --, sfs->super_dirty = 1;
    assert(sfs_block_inuse(sfs, *ino_store));
    return sfs_clear_block(sfs, *ino_store, 1);
}

/*
 * sfs_block_free - set related bits for ino block to 1(means free) in bitmap, add sfs->super.
unused_blocks, set superblock dirty *
 */
static void
sfs_block_free(struct sfs_fs *sfs, uint32_t ino) {
    assert(sfs_block_inuse(sfs, ino));
    bitmap_free(sfs->freemap, ino);
    sfs->super.unused_blocks ++, sfs->super_dirty = 1;
}

/*
 * sfs_create_inode - alloc a inode in memroy, and init din/ino/dirty/reclian_count/sem fields
 in sfs_inode in inode
 */
static int
sfs_create_inode(struct sfs_fs *sfs, struct sfs_disk_inode *din, uint32_t ino, struct inode **
node_store) {
```

```
    struct inode *node;
    if ((node = alloc_inode(sfs_inode)) != NULL) {
        vop_init(node, sfs_get_ops(din->type), info2fs(sfs, sfs));
        struct sfs_inode *sin = vop_info(node, sfs_inode);
        sin->din = din, sin->ino = ino, sin->dirty = 0, sin->reclaim_count = 1;
        sem_init(&(sin->sem), 1);
        *node_store = node;
        return 0;
    }
    return -E_NO_MEM;
}

/*
 * lookup_sfs_nolock - according ino, find related inode
 *
 * NOTICE: le2sin, info2node MACRO
 */
static struct inode *
lookup_sfs_nolock(struct sfs_fs *sfs, uint32_t ino) {
    struct inode *node;
    list_entry_t *list = sfs_hash_list(sfs, ino), *le = list;
    while ((le = list_next(le)) != list) {
        struct sfs_inode *sin = le2sin(le, hash_link);
        if (sin->ino == ino) {
            node = info2node(sin, sfs_inode);
            if (vop_ref_inc(node) == 1) {
                sin->reclaim_count ++;
            }
            return node;
        }
    }
    return NULL;
}

/*
 * sfs_load_inode - If the inode isn't existed, load inode related ino disk block data into a
new created inode.
 *                  If the inode is in memory alreadily, then do nothing
 */
int
sfs_load_inode(struct sfs_fs *sfs, struct inode **node_store, uint32_t ino) {
    lock_sfs_fs(sfs);
    struct inode *node;
    if ((node = lookup_sfs_nolock(sfs, ino)) != NULL) {
        goto out_unlock;
    }

    int ret = -E_NO_MEM;
    struct sfs_disk_inode *din;
    if ((din = kmalloc(sizeof(struct sfs_disk_inode))) == NULL) {
        goto failed_unlock;
    }

    assert(sfs_block_inuse(sfs, ino));
    if ((ret = sfs_rbuf(sfs, din, sizeof(struct sfs_disk_inode), ino, 0)) != 0) {
        goto failed_cleanup_din;
    }

    assert(din->nlinks != 0);
    if ((ret = sfs_create_inode(sfs, din, ino, &node)) != 0) {
        goto failed_cleanup_din;
    }
    sfs_set_links(sfs, vop_info(node, sfs_inode));

out_unlock:
```

```
    unlock_sfs_fs(sfs);
    *node_store = node;
    return 0;

failed_cleanup_din:
    kfree(din);
failed_unlock:
    unlock_sfs_fs(sfs);
    return ret;
}

/*
 * sfs_bmap_get_sub_nolock - according entry pointer entp and index, find the index of indrect
 disk block
 *                           return the index of indrect disk block to ino_store. no lock prot
ect
 * @sfs:      sfs file system
 * @entp:     the pointer of index of entry disk block
 * @index:    the index of block in indrect block
 * @create:   BOOL, if the block isn't allocated, if create = 1 the alloc a block,  otherwise
just do nothing
 * @ino_store: 0 OR the index of already inused block or new allocated block.
 */
static int
sfs_bmap_get_sub_nolock(struct sfs_fs *sfs, uint32_t *entp, uint32_t index, bool create, uint3
2_t *ino_store) {
    assert(index < SFS_BLK_NENTRY);
    int ret;
    uint32_t ent, ino = 0;
    off_t offset = index * sizeof(uint32_t);  // the offset of entry in entry block
        // if entry block is existd, read the content of entry block into  sfs->sfs_buffer
    if ((ent = *entp) != 0) {
        if ((ret = sfs_rbuf(sfs, &ino, sizeof(uint32_t), ent, offset)) != 0) {
            return ret;
        }
        if (ino != 0 || !create) {
            goto out;
        }
    }
    else {
        if (!create) {
            goto out;
        }
                //if entry block isn't existd, allocated a entry block (for indrect block)
        if ((ret = sfs_block_alloc(sfs, &ent)) != 0) {
            return ret;
        }
    }

    if ((ret = sfs_block_alloc(sfs, &ino)) != 0) {
        goto failed_cleanup;
    }
    if ((ret = sfs_wbuf(sfs, &ino, sizeof(uint32_t), ent, offset)) != 0) {
        sfs_block_free(sfs, ino);
        goto failed_cleanup;
    }

out:
    if (ent != *entp) {
        *entp = ent;
    }
    *ino_store = ino;
    return 0;

failed_cleanup:
```

```
    if (ent != *entp) {
        sfs_block_free(sfs, ent);
    }
    return ret;
}

/*
 * sfs_bmap_get_nolock - according sfs_inode and index of block, find the NO. of disk block
 *                       no lock protect
 * @sfs:      sfs file system
 * @sin:      sfs inode in memory
 * @index:    the index of block in inode
 * @create:   BOOL, if the block isn't allocated, if create = 1 the alloc a block,  otherwise
just do nothing
 * @ino_store: 0 OR the index of already inused block or new allocated block.
 */
static int
sfs_bmap_get_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, uint32_t index, bool create, ui
nt32_t *ino_store) {
    struct sfs_disk_inode *din = sin->din;
    int ret;
    uint32_t ent, ino;
        // the index of disk block is in the fist SFS_NDIRECT  direct blocks
    if (index < SFS_NDIRECT) {
        if ((ino = din->direct[index]) == 0 && create) {
            if ((ret = sfs_block_alloc(sfs, &ino)) != 0) {
                return ret;
            }
            din->direct[index] = ino;
            sin->dirty = 1;
        }
        goto out;
    }
    // the index of disk block is in the indirect blocks.
    index -= SFS_NDIRECT;
    if (index < SFS_BLK_NENTRY) {
        ent = din->indirect;
        if ((ret = sfs_bmap_get_sub_nolock(sfs, &ent, index, create, &ino)) != 0) {
            return ret;
        }
        if (ent != din->indirect) {
            assert(din->indirect == 0);
            din->indirect = ent;
            sin->dirty = 1;
        }
        goto out;
    } else {
                panic ("sfs_bmap_get_nolock - index out of range");
        }
out:
    assert(ino == 0 || sfs_block_inuse(sfs, ino));
    *ino_store = ino;
    return 0;
}

/*
 * sfs_bmap_free_sub_nolock - set the entry item to 0 (free) in the indirect block
 */
static int
sfs_bmap_free_sub_nolock(struct sfs_fs *sfs, uint32_t ent, uint32_t index) {
    assert(sfs_block_inuse(sfs, ent) && index < SFS_BLK_NENTRY);
    int ret;
    uint32_t ino, zero = 0;
    off_t offset = index * sizeof(uint32_t);
    if ((ret = sfs_rbuf(sfs, &ino, sizeof(uint32_t), ent, offset)) != 0) {
```

```
        return ret;
    }
    if (ino != 0) {
        if ((ret = sfs_wbuf(sfs, &zero, sizeof(uint32_t), ent, offset)) != 0) {
            return ret;
        }
        sfs_block_free(sfs, ino);
    }
    return 0;
}

/*
 * sfs_bmap_free_nolock - free a block with logical index in inode and reset the inode's field
s
 */
static int
sfs_bmap_free_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, uint32_t index) {
    struct sfs_disk_inode *din = sin->din;
    int ret;
    uint32_t ent, ino;
    if (index < SFS_NDIRECT) {
        if ((ino = din->direct[index]) != 0) {
                        // free the block
            sfs_block_free(sfs, ino);
            din->direct[index] = 0;
            sin->dirty = 1;
        }
        return 0;
    }

    index -= SFS_NDIRECT;
    if (index < SFS_BLK_NENTRY) {
        if ((ent = din->indirect) != 0) {
                        // set the entry item to 0 in the indirect block
            if ((ret = sfs_bmap_free_sub_nolock(sfs, ent, index)) != 0) {
                return ret;
            }
        }
        return 0;
    }
    return 0;
}

/*
 * sfs_bmap_load_nolock - according to the DIR's inode and the logical index of block in inode
, find the NO. of disk block.
 * @sfs:      sfs file system
 * @sin:      sfs inode in memory
 * @index:    the logical index of disk block in inode
 * @ino_store:the NO. of disk block
 */
static int
sfs_bmap_load_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, uint32_t index, uint32_t *ino_
store) {
    struct sfs_disk_inode *din = sin->din;
    assert(index <= din->blocks);
    int ret;
    uint32_t ino;
    bool create = (index == din->blocks);
    if ((ret = sfs_bmap_get_nolock(sfs, sin, index, create, &ino)) != 0) {
        return ret;
    }
    assert(sfs_block_inuse(sfs, ino));
    if (create) {
        din->blocks ++;
```

```
    }
    if (ino_store != NULL) {
        *ino_store = ino;
    }
    return 0;
}

/*
 * sfs_bmap_truncate_nolock - free the disk block at the end of file
 */
static int
sfs_bmap_truncate_nolock(struct sfs_fs *sfs, struct sfs_inode *sin) {
    struct sfs_disk_inode *din = sin->din;
    assert(din->blocks != 0);
    int ret;
    if ((ret = sfs_bmap_free_nolock(sfs, sin, din->blocks - 1)) != 0) {
        return ret;
    }
    din->blocks --;
    sin->dirty = 1;
    return 0;
}

/*
 * sfs_dirent_read_nolock - read the file entry from disk block which contains this entry
 * @sfs:      sfs file system
 * @sin:      sfs inode in memory
 * @slot:     the index of file entry
 * @entry:    file entry
 */
static int
sfs_dirent_read_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, int slot, struct sfs_disk_en
try *entry) {
    assert(sin->din->type == SFS_TYPE_DIR && (slot >= 0 && slot < sin->din->blocks));
    int ret;
    uint32_t ino;
        // according to the DIR's inode and the slot of file entry, find the index of disk blo
ck which contains this file entry
    if ((ret = sfs_bmap_load_nolock(sfs, sin, slot, &ino)) != 0) {
        return ret;
    }
    assert(sfs_block_inuse(sfs, ino));
        // read the content of file entry in the disk block
    if ((ret = sfs_rbuf(sfs, entry, sizeof(struct sfs_disk_entry), ino, 0)) != 0) {
        return ret;
    }
    entry->name[SFS_MAX_FNAME_LEN] = '\0';
    return 0;
}

#define sfs_dirent_link_nolock_check(sfs, sin, slot, lnksin, name)                  \
    do {                                                                            \
        int err;                                                                    \
        if ((err = sfs_dirent_link_nolock(sfs, sin, slot, lnksin, name)) != 0) {    \
            warn("sfs_dirent_link error: %e.\n", err);                              \
        }                                                                           \
    } while (0)

#define sfs_dirent_unlink_nolock_check(sfs, sin, slot, lnksin)                      \
    do {                                                                            \
        int err;                                                                    \
        if ((err = sfs_dirent_unlink_nolock(sfs, sin, slot, lnksin)) != 0) {        \
            warn("sfs_dirent_unlink error: %e.\n", err);                            \
        }                                                                           \
    } while (0)
```

```
/*
 * sfs_dirent_search_nolock - read every file entry in the DIR, compare file name with each en
try->name
 *                            If equal, then return slot and NO. of disk of this file's inode
 * @sfs:        sfs file system
 * @sin:        sfs inode in memory
 * @name:       the filename
 * @ino_store:  NO. of disk of this file (with the filename)'s inode
 * @slot:       logical index of file entry (NOTICE: each file entry ocupied one  disk block)
 * @empty_slot: the empty logical index of file entry.
 */
static int
sfs_dirent_search_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, const char *name, uint32_t
 *ino_store, int *slot, int *empty_slot) {
    assert(strlen(name) <= SFS_MAX_FNAME_LEN);
    struct sfs_disk_entry *entry;
    if ((entry = kmalloc(sizeof(struct sfs_disk_entry))) == NULL) {
        return -E_NO_MEM;
    }

#define set_pvalue(x, v)            do { if ((x) != NULL) { *(x) = (v); } } while (0)
    int ret, i, nslots = sin->din->blocks;
    set_pvalue(empty_slot, nslots);
    for (i = 0; i < nslots; i ++) {
        if ((ret = sfs_dirent_read_nolock(sfs, sin, i, entry)) != 0) {
            goto out;
        }
        if (entry->ino == 0) {
            set_pvalue(empty_slot, i);
            continue ;
        }
        if (strcmp(name, entry->name) == 0) {
            set_pvalue(slot, i);
            set_pvalue(ino_store, entry->ino);
            goto out;
        }
    }
#undef set_pvalue
    ret = -E_NOENT;
out:
    kfree(entry);
    return ret;
}

/*
 * sfs_dirent_findino_nolock - read all file entries in DIR's inode and find a entry->ino == i
no
 */

static int
sfs_dirent_findino_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, uint32_t ino, struct sfs_
disk_entry *entry) {
    int ret, i, nslots = sin->din->blocks;
    for (i = 0; i < nslots; i ++) {
        if ((ret = sfs_dirent_read_nolock(sfs, sin, i, entry)) != 0) {
            return ret;
        }
        if (entry->ino == ino) {
            return 0;
        }
    }
    return -E_NOENT;
}
```

```c
/*
 * sfs_lookup_once - find inode corresponding the file name in DIR's sin inode
 * @sfs:        sfs file system
 * @sin:        DIR sfs inode in memory
 * @name:       the file name in DIR
 * @node_store: the inode corresponding the file name in DIR
 * @slot:       the logical index of file entry
 */
static int
sfs_lookup_once(struct sfs_fs *sfs, struct sfs_inode *sin, const char *name, struct inode **no
de_store, int *slot) {
    int ret;
    uint32_t ino;
    lock_sin(sin);
    {   // find the NO. of disk block and logical index of file entry
        ret = sfs_dirent_search_nolock(sfs, sin, name, &ino, slot, NULL);
    }
    unlock_sin(sin);
    if (ret == 0) {
                // load the content of inode with the the NO. of disk block
        ret = sfs_load_inode(sfs, node_store, ino);
    }
    return ret;
}

// sfs_opendir - just check the opne_flags, now support readonly
static int
sfs_opendir(struct inode *node, uint32_t open_flags) {
    switch (open_flags & O_ACCMODE) {
    case O_RDONLY:
        break;
    case O_WRONLY:
    case O_RDWR:
    default:
        return -E_ISDIR;
    }
    if (open_flags & O_APPEND) {
        return -E_ISDIR;
    }
    return 0;
}

// sfs_openfile - open file (no use)
static int
sfs_openfile(struct inode *node, uint32_t open_flags) {
    return 0;
}

// sfs_close - close file
static int
sfs_close(struct inode *node) {
    return vop_fsync(node);
}

/*
 * sfs_io_nolock - Rd/Wr a file contentfrom offset position to offset+ length  disk blocks<-->
buffer (in memroy)
 * @sfs:      sfs file system
 * @sin:      sfs inode in memory
 * @buf:      the buffer Rd/Wr
 * @offset:   the offset of file
 * @alenp:    the length need to read (is a pointer). and will RETURN the really Rd/Wr lenght
 * @write:    BOOL, 0 read, 1 write
 */
static int
```

```
sfs_io_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, void *buf, off_t offset, size_t *alen
p, bool write) {
    struct sfs_disk_inode *din = sin->din;
    assert(din->type != SFS_TYPE_DIR);
    off_t endpos = offset + *alenp, blkoff;
    *alenp = 0;
        // calculate the Rd/Wr end position
    if (offset < 0 || offset >= SFS_MAX_FILE_SIZE || offset > endpos) {
        return -E_INVAL;
    }
    if (offset == endpos) {
        return 0;
    }
    if (endpos > SFS_MAX_FILE_SIZE) {
        endpos = SFS_MAX_FILE_SIZE;
    }
    if (!write) {
        if (offset >= din->size) {
            return 0;
        }
        if (endpos > din->size) {
            endpos = din->size;
        }
    }

    int (*sfs_buf_op)(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset)
;
    int (*sfs_block_op)(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks);
    if (write) {
        sfs_buf_op = sfs_wbuf, sfs_block_op = sfs_wblock;
    }
    else {
        sfs_buf_op = sfs_rbuf, sfs_block_op = sfs_rblock;
    }

    int ret = 0;
    size_t size, alen = 0;
    uint32_t ino;
    uint32_t blkno = offset / SFS_BLKSIZE;          // The NO. of Rd/Wr begin block
    uint32_t nblks = endpos / SFS_BLKSIZE - blkno;  // The size of Rd/Wr blocks

  //LAB8:EXERCISE1 YOUR CODE HINT: call sfs_bmap_load_nolock, sfs_rbuf, sfs_rblock,etc. read d
ifferent kind of blocks in file
        /*
         * (1) If offset isn't aligned with the first block, Rd/Wr some content from offset to
 the end of the first block
         *       NOTICE: useful function: sfs_bmap_load_nolock, sfs_buf_op
         *               Rd/Wr size = (nblks != 0) ? (SFS_BLKSIZE - blkoff) : (endpos - offset
)
         * (2) Rd/Wr aligned blocks
         *       NOTICE: useful function: sfs_bmap_load_nolock, sfs_block_op
     * (3) If end position isn't aligned with the last block, Rd/Wr some content from begin to
 the (endpos % SFS_BLKSIZE) of the last block
         *       NOTICE: useful function: sfs_bmap_load_nolock, sfs_buf_op
        */
    if ((blkoff = offset % SFS_BLKSIZE) != 0) {
        size = (nblks != 0) ? (SFS_BLKSIZE - blkoff) : (endpos - offset);
        if ((ret = sfs_bmap_load_nolock(sfs, sin, blkno, &ino)) != 0) {
            goto out;
        }
        if ((ret = sfs_buf_op(sfs, buf, size, ino, blkoff)) != 0) {
            goto out;
        }
        alen += size;
        if (nblks == 0) {
```

```
            goto out;
        }
        buf += size, blkno ++, nblks --;
    }

    size = SFS_BLKSIZE;
    while (nblks != 0) {
        if ((ret = sfs_bmap_load_nolock(sfs, sin, blkno, &ino)) != 0) {
            goto out;
        }
        if ((ret = sfs_block_op(sfs, buf, ino, 1)) != 0) {
            goto out;
        }
        alen += size, buf += size, blkno ++, nblks --;
    }

    if ((size = endpos % SFS_BLKSIZE) != 0) {
        if ((ret = sfs_bmap_load_nolock(sfs, sin, blkno, &ino)) != 0) {
            goto out;
        }
        if ((ret = sfs_buf_op(sfs, buf, size, ino, 0)) != 0) {
            goto out;
        }
        alen += size;
    }
out:
    *alenp = alen;
    if (offset + alen > sin->din->size) {
        sin->din->size = offset + alen;
        sin->dirty = 1;
    }
    return ret;
}

/*
 * sfs_io - Rd/Wr file. the wrapper of sfs_io_nolock
            with lock protect
 */
static inline int
sfs_io(struct inode *node, struct iobuf *iob, bool write) {
    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    struct sfs_inode *sin = vop_info(node, sfs_inode);
    int ret;
    lock_sin(sin);
    {
        size_t alen = iob->io_resid;
        ret = sfs_io_nolock(sfs, sin, iob->io_base, iob->io_offset, &alen, write);
        if (alen != 0) {
            iobuf_skip(iob, alen);
        }
    }
    unlock_sin(sin);
    return ret;
}

// sfs_read - read file
static int
sfs_read(struct inode *node, struct iobuf *iob) {
    return sfs_io(node, iob, 0);
}

// sfs_write - write file
static int
sfs_write(struct inode *node, struct iobuf *iob) {
    return sfs_io(node, iob, 1);
```

```
}

/*
 * sfs_fstat - Return nlinks/block/size, etc. info about a file. The pointer is a pointer to s
truct stat;
 */
static int
sfs_fstat(struct inode *node, struct stat *stat) {
    int ret;
    memset(stat, 0, sizeof(struct stat));
    if ((ret = vop_gettype(node, &(stat->st_mode))) != 0) {
        return ret;
    }
    struct sfs_disk_inode *din = vop_info(node, sfs_inode)->din;
    stat->st_nlinks = din->nlinks;
    stat->st_blocks = din->blocks;
    stat->st_size = din->size;
    return 0;
}

/*
 * sfs_fsync - Force any dirty inode info associated with this file to stable storage.
 */
static int
sfs_fsync(struct inode *node) {
    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    struct sfs_inode *sin = vop_info(node, sfs_inode);
    int ret = 0;
    if (sin->dirty) {
        lock_sin(sin);
        {
            if (sin->dirty) {
                sin->dirty = 0;
                if ((ret = sfs_wbuf(sfs, sin->din, sizeof(struct sfs_disk_inode), sin->ino, 0)
) != 0) {
                    sin->dirty = 1;
                }
            }
        }
        unlock_sin(sin);
    }
    return ret;
}

/*
 *sfs_namefile -Compute pathname relative to filesystem root of the file and copy to the speci
fied io buffer.
 *
 */
static int
sfs_namefile(struct inode *node, struct iobuf *iob) {
    struct sfs_disk_entry *entry;
    if (iob->io_resid <= 2 || (entry = kmalloc(sizeof(struct sfs_disk_entry))) == NULL) {
        return -E_NO_MEM;
    }

    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    struct sfs_inode *sin = vop_info(node, sfs_inode);

    int ret;
    char *ptr = iob->io_base + iob->io_resid;
    size_t alen, resid = iob->io_resid - 2;
    vop_ref_inc(node);
    while (1) {
        struct inode *parent;
```

```
        if ((ret = sfs_lookup_once(sfs, sin, "..", &parent, NULL)) != 0) {
            goto failed;
        }

        uint32_t ino = sin->ino;
        vop_ref_dec(node);
        if (node == parent) {
            vop_ref_dec(node);
            break;
        }

        node = parent, sin = vop_info(node, sfs_inode);
        assert(ino != sin->ino && sin->din->type == SFS_TYPE_DIR);

        lock_sin(sin);
        {
            ret = sfs_dirent_findino_nolock(sfs, sin, ino, entry);
        }
        unlock_sin(sin);

        if (ret != 0) {
            goto failed;
        }

        if ((alen = strlen(entry->name) + 1) > resid) {
            goto failed_nomem;
        }
        resid -= alen, ptr -= alen;
        memcpy(ptr, entry->name, alen - 1);
        ptr[alen - 1] = '/';
    }
    alen = iob->io_resid - resid - 2;
    ptr = memmove(iob->io_base + 1, ptr, alen);
    ptr[-1] = '/', ptr[alen] = '\0';
    iobuf_skip(iob, alen);
    kfree(entry);
    return 0;

failed_nomem:
    ret = -E_NO_MEM;
failed:
    vop_ref_dec(node);
    kfree(entry);
    return ret;
}

/*
 * sfs_getdirentry_sub_noblock - get the content of file entry in DIR
 */
static int
sfs_getdirentry_sub_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, int slot, struct sfs_dis
k_entry *entry) {
    int ret, i, nslots = sin->din->blocks;
    for (i = 0; i < nslots; i ++) {
        if ((ret = sfs_dirent_read_nolock(sfs, sin, i, entry)) != 0) {
            return ret;
        }
        if (entry->ino != 0) {
            if (slot == 0) {
                return 0;
            }
            slot --;
        }
    }
    return -E_NOENT;
```

```
}

/*
 * sfs_getdirentry - according to the iob->io_offset, calculate the dir entry's slot in disk b
lock,
                     get dir entry content from the disk
 */
static int
sfs_getdirentry(struct inode *node, struct iobuf *iob) {
    struct sfs_disk_entry *entry;
    if ((entry = kmalloc(sizeof(struct sfs_disk_entry))) == NULL) {
        return -E_NO_MEM;
    }

    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    struct sfs_inode *sin = vop_info(node, sfs_inode);

    int ret, slot;
    off_t offset = iob->io_offset;
    if (offset < 0 || offset % sfs_dentry_size != 0) {
        kfree(entry);
        return -E_INVAL;
    }
    if ((slot = offset / sfs_dentry_size) > sin->din->blocks) {
        kfree(entry);
        return -E_NOENT;
    }
    lock_sin(sin);
    if ((ret = sfs_getdirentry_sub_nolock(sfs, sin, slot, entry)) != 0) {
        unlock_sin(sin);
        goto out;
    }
    unlock_sin(sin);
    ret = iobuf_move(iob, entry->name, sfs_dentry_size, 1, NULL);
out:
    kfree(entry);
    return ret;
}

/*
 * sfs_reclaim - Free all resources inode occupied . Called when inode is no longer in use.
 */
static int
sfs_reclaim(struct inode *node) {
    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    struct sfs_inode *sin = vop_info(node, sfs_inode);

    int  ret = -E_BUSY;
    uint32_t ent;
    lock_sfs_fs(sfs);
    assert(sin->reclaim_count > 0);
    if ((-- sin->reclaim_count) != 0 || inode_ref_count(node) != 0) {
        goto failed_unlock;
    }
    if (sin->din->nlinks == 0) {
        if ((ret = vop_truncate(node, 0)) != 0) {
            goto failed_unlock;
        }
    }
    if (sin->dirty) {
        if ((ret = vop_fsync(node)) != 0) {
            goto failed_unlock;
        }
    }
    sfs_remove_links(sin);
```

```
    unlock_sfs_fs(sfs);

    if (sin->din->nlinks == 0) {
        sfs_block_free(sfs, sin->ino);
        if ((ent = sin->din->indirect) != 0) {
            sfs_block_free(sfs, ent);
        }
    }
    kfree(sin->din);
    vop_kill(node);
    return 0;

failed_unlock:
    unlock_sfs_fs(sfs);
    return ret;
}

/*
 * sfs_gettype - Return type of file. The values for file types are in sfs.h.
 */
static int
sfs_gettype(struct inode *node, uint32_t *type_store) {
    struct sfs_disk_inode *din = vop_info(node, sfs_inode)->din;
    switch (din->type) {
    case SFS_TYPE_DIR:
        *type_store = S_IFDIR;
        return 0;
    case SFS_TYPE_FILE:
        *type_store = S_IFREG;
        return 0;
    case SFS_TYPE_LINK:
        *type_store = S_IFLNK;
        return 0;
    }
    panic("invalid file type %d.\n", din->type);
}

/*
 * sfs_tryseek - Check if seeking to the specified position within the file is legal.
 */
static int
sfs_tryseek(struct inode *node, off_t pos) {
    if (pos < 0 || pos >= SFS_MAX_FILE_SIZE) {
        return -E_INVAL;
    }
    struct sfs_inode *sin = vop_info(node, sfs_inode);
    if (pos > sin->din->size) {
        return vop_truncate(node, pos);
    }
    return 0;
}

/*
 * sfs_truncfile : reszie the file with new length
 */
static int
sfs_truncfile(struct inode *node, off_t len) {
    if (len < 0 || len > SFS_MAX_FILE_SIZE) {
        return -E_INVAL;
    }
    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    struct sfs_inode *sin = vop_info(node, sfs_inode);
    struct sfs_disk_inode *din = sin->din;

    int ret = 0;
```

```
        //new number of disk blocks of file
    uint32_t nblks, tblks = ROUNDUP_DIV(len, SFS_BLKSIZE);
    if (din->size == len) {
        assert(tblks == din->blocks);
        return 0;
    }

    lock_sin(sin);
        // old number of disk blocks of file
    nblks = din->blocks;
    if (nblks < tblks) {
                // try to enlarge the file size by add new disk block at the end of file
        while (nblks != tblks) {
            if ((ret = sfs_bmap_load_nolock(sfs, sin, nblks, NULL)) != 0) {
                goto out_unlock;
            }
            nblks ++;
        }
    }
    else if (tblks < nblks) {
                // try to reduce the file size
        while (tblks != nblks) {
            if ((ret = sfs_bmap_truncate_nolock(sfs, sin)) != 0) {
                goto out_unlock;
            }
            nblks --;
        }
    }
    assert(din->blocks == tblks);
    din->size = len;
    sin->dirty = 1;

out_unlock:
    unlock_sin(sin);
    return ret;
}

/*
 * sfs_lookup - Parse path relative to the passed directory
 *              DIR, and hand back the inode for the file it
 *              refers to.
 */
static int
sfs_lookup(struct inode *node, char *path, struct inode **node_store) {
    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    assert(*path != '\0' && *path != '/');
    vop_ref_inc(node);
    struct sfs_inode *sin = vop_info(node, sfs_inode);
    if (sin->din->type != SFS_TYPE_DIR) {
        vop_ref_dec(node);
        return -E_NOTDIR;
    }
    struct inode *subnode;
    int ret = sfs_lookup_once(sfs, sin, path, &subnode, NULL);

    vop_ref_dec(node);
    if (ret != 0) {
        return ret;
    }
    *node_store = subnode;
    return 0;
}

// The sfs specific DIR operations correspond to the abstract operations on a inode.
static const struct inode_ops sfs_node_dirops = {
```

```
    .vop_magic                      = VOP_MAGIC,
    .vop_open                       = sfs_opendir,
    .vop_close                      = sfs_close,
    .vop_fstat                      = sfs_fstat,
    .vop_fsync                      = sfs_fsync,
    .vop_namefile                   = sfs_namefile,
    .vop_getdirentry                = sfs_getdirentry,
    .vop_reclaim                    = sfs_reclaim,
    .vop_gettype                    = sfs_gettype,
    .vop_lookup                     = sfs_lookup,
};
/// The sfs specific FILE operations correspond to the abstract operations on a inode.
static const struct inode_ops sfs_node_fileops = {
    .vop_magic                      = VOP_MAGIC,
    .vop_open                       = sfs_openfile,
    .vop_close                      = sfs_close,
    .vop_read                       = sfs_read,
    .vop_write                      = sfs_write,
    .vop_fstat                      = sfs_fstat,
    .vop_fsync                      = sfs_fsync,
    .vop_reclaim                    = sfs_reclaim,
    .vop_gettype                    = sfs_gettype,
    .vop_tryseek                    = sfs_tryseek,
    .vop_truncate                   = sfs_truncfile,
};

[ucore/lab8_result/kern/fs/sfs/bitmap.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_FS_SFS_BITMAP_H__
#define __KERN_FS_SFS_BITMAP_H__

#include <defs.h>


/*
 * Fixed-size array of bits. (Intended for storage management.)
 *
 * Functions:
 *     bitmap_create  - allocate a new bitmap object.
 *                      Returns NULL on error.
 *     bitmap_getdata - return pointer to raw bit data (for I/O).
 *     bitmap_alloc   - locate a cleared bit, set it, and return its index.
 *     bitmap_mark    - set a clear bit by its index.
 *     bitmap_unmark  - clear a set bit by its index.
 *     bitmap_isset   - return whether a particular bit is set or not.
 *     bitmap_destroy - destroy bitmap.
 */


struct bitmap;

struct bitmap *bitmap_create(uint32_t nbits);                       // allocate a new bitmap obj
ect.
int bitmap_alloc(struct bitmap *bitmap, uint32_t *index_store);     // locate a cleared bit, set
 it, and return its index.
bool bitmap_test(struct bitmap *bitmap, uint32_t index);            // return whether a particul
ar bit is set or not.
void bitmap_free(struct bitmap *bitmap, uint32_t index);            // according index, set rela
ted bit to 1
void bitmap_destroy(struct bitmap *bitmap);                         // free memory contains bitm
ap
void *bitmap_getdata(struct bitmap *bitmap, size_t *len_store);     // return pointer to raw bit
 data (for I/O)

#endif /* !__KERN_FS_SFS_BITMAP_H__ */
```

[ucore/lab8_result/kern/fs/sfs/sfs.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <sfs.h>
#include <error.h>
#include <assert.h>

/*
 * sfs_init - mount sfs on disk0
 *
 * CALL GRAPH:
 *   kern_init-->fs_init-->sfs_init
 */
void
sfs_init(void) {
    int ret;
    if ((ret = sfs_mount("disk0")) != 0) {
        panic("failed: sfs: sfs_mount: %e.\n", ret);
    }
}
```

[ucore/lab8_result/kern/fs/sysfile.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_FS_SYSFILE_H__
#define __KERN_FS_SYSFILE_H__

#include <defs.h>

struct stat;
struct dirent;

int sysfile_open(const char *path, uint32_t open_flags);        // Open or create a file. FLAG
S/MODE per the syscall.
int sysfile_close(int fd);                                      // Close a vnode opened
int sysfile_read(int fd, void *base, size_t len);               // Read file
int sysfile_write(int fd, void *base, size_t len);              // Write file
int sysfile_seek(int fd, off_t pos, int whence);                // Seek file
int sysfile_fstat(int fd, struct stat *stat);                   // Stat file
int sysfile_fsync(int fd);                                      // Sync file
int sysfile_chdir(const char *path);                            // change DIR
int sysfile_mkdir(const char *path);                            // create DIR
int sysfile_link(const char *path1, const char *path2);         // set a path1's link as path2
int sysfile_rename(const char *path1, const char *path2);       // rename file
int sysfile_unlink(const char *path);                           // unlink a path
int sysfile_getcwd(char *buf, size_t len);                      // get current working directo
ry
int sysfile_getdirentry(int fd, struct dirent *direntp);        // get the file entry in DIR
int sysfile_dup(int fd1, int fd2);                              // duplicate file
int sysfile_pipe(int *fd_store);                                // build PIPE
int sysfile_mkfifo(const char *name, uint32_t open_flags);      // build named PIPE

#endif /* !__KERN_FS_SYSFILE_H__ */
```

[ucore/lab8_result/kern/fs/fs.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <kmalloc.h>
#include <sem.h>
#include <vfs.h>
#include <dev.h>
#include <file.h>
#include <sfs.h>
#include <inode.h>
#include <assert.h>
//called when init_main proc start
void
fs_init(void) {
    vfs_init();
```

```
    dev_init();
    sfs_init();
}

void
fs_cleanup(void) {
    vfs_cleanup();
}

void
lock_files(struct files_struct *filesp) {
    down(&(filesp->files_sem));
}

void
unlock_files(struct files_struct *filesp) {
    up(&(filesp->files_sem));
}
//Called when a new proc init
struct files_struct *
files_create(void) {
    //cprintf("[files_create]\n");
    static_assert((int)FILES_STRUCT_NENTRY > 128);
    struct files_struct *filesp;
    if ((filesp = kmalloc(sizeof(struct files_struct) + FILES_STRUCT_BUFSIZE)) != NULL) {
        filesp->pwd = NULL;
        filesp->fd_array = (void *)(filesp + 1);
        filesp->files_count = 0;
        sem_init(&(filesp->files_sem), 1);
        fd_array_init(filesp->fd_array);
    }
    return filesp;
}
//Called when a proc exit
void
files_destroy(struct files_struct *filesp) {
//    cprintf("[files_destroy]\n");
    assert(filesp != NULL && files_count(filesp) == 0);
    if (filesp->pwd != NULL) {
        vop_ref_dec(filesp->pwd);
    }
    int i;
    struct file *file = filesp->fd_array;
    for (i = 0; i < FILES_STRUCT_NENTRY; i ++, file ++) {
        if (file->status == FD_OPENED) {
            fd_array_close(file);
        }
        assert(file->status == FD_NONE);
    }
    kfree(filesp);
}

void
files_closeall(struct files_struct *filesp) {
//    cprintf("[files_closeall]\n");
    assert(filesp != NULL && files_count(filesp) > 0);
    int i;
    struct file *file = filesp->fd_array;
    //skip the stdin & stdout
    for (i = 2, file += 2; i < FILES_STRUCT_NENTRY; i ++, file ++) {
        if (file->status == FD_OPENED) {
            fd_array_close(file);
        }
    }
}
```

```
int
dup_files(struct files_struct *to, struct files_struct *from) {
//    cprintf("[dup_files]\n");
    assert(to != NULL && from != NULL);
    assert(files_count(to) == 0 && files_count(from) > 0);
    if ((to->pwd = from->pwd) != NULL) {
        vop_ref_inc(to->pwd);
    }
    int i;
    struct file *to_file = to->fd_array, *from_file = from->fd_array;
    for (i = 0; i < FILES_STRUCT_NENTRY; i ++, to_file ++, from_file ++) {
        if (from_file->status == FD_OPENED) {
            /* alloc_fd first */
            to_file->status = FD_INIT;
            fd_array_dup(to_file, from_file);
        }
    }
    return 0;
}
```

[ucore/lab8_result/kern/fs/iobuf.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_FS_IOBUF_H__
#define __KERN_FS_IOBUF_H__

#include <defs.h>

/*
 * iobuf is a buffer Rd/Wr status record
 */
struct iobuf {
    void *io_base;     // the base addr of buffer (used for Rd/Wr)
    off_t io_offset;   // current Rd/Wr position in buffer, will have been incremented by the
amount transferred
    size_t io_len;     // the length of buffer  (used for Rd/Wr)
    size_t io_resid;   // current resident length need to Rd/Wr, will have been decremented by
 the amount transferred.
};

#define iobuf_used(iob)                         ((size_t)((iob)->io_len - (iob)->io_resid))

struct iobuf *iobuf_init(struct iobuf *iob, void *base, size_t len, off_t offset);
int iobuf_move(struct iobuf *iob, void *data, size_t len, bool m2b, size_t *copiedp);
int iobuf_move_zeros(struct iobuf *iob, size_t len, size_t *copiedp);
void iobuf_skip(struct iobuf *iob, size_t n);

#endif /* !__KERN_FS_IOBUF_H__ */
```

[ucore/lab8_result/kern/fs/iobuf.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <string.h>
#include <iobuf.h>
#include <error.h>
#include <assert.h>

/*
 * iobuf_init - init io buffer struct.
 *                set up io_base to point to the buffer you want to transfer to, and set io_le
n to the length of buffer;
 *                initialize io_offset as desired;
 *                initialize io_resid to the total amount of data that can be transferred thro
ugh this io.
 */
struct iobuf *
iobuf_init(struct iobuf *iob, void *base, size_t len, off_t offset) {
```

```
    iob->io_base = base;
    iob->io_offset = offset;
    iob->io_len = iob->io_resid = len;
    return iob;
}

/* iobuf_move - move data  (iob->io_base ---> data OR  data --> iob->io.base) in memory
 * @copiedp:  the size of data memcopied
 *
 * iobuf_move may be called repeatedly on the same io to transfer
 * additional data until the available buffer space the io refers to
 * is exhausted.
 */
int
iobuf_move(struct iobuf *iob, void *data, size_t len, bool m2b, size_t *copiedp) {
    size_t alen;
    if ((alen = iob->io_resid) > len) {
        alen = len;
    }
    if (alen > 0) {
        void *src = iob->io_base, *dst = data;
        if (m2b) {
            void *tmp = src;
            src = dst, dst = tmp;
        }
        memmove(dst, src, alen);
        iobuf_skip(iob, alen), len -= alen;
    }
    if (copiedp != NULL) {
        *copiedp = alen;
    }
    return (len == 0) ? 0 : -E_NO_MEM;
}

/*
 * iobuf_move_zeros - set io buffer zero
 * @copiedp:  the size of data memcopied
 */
int
iobuf_move_zeros(struct iobuf *iob, size_t len, size_t *copiedp) {
    size_t alen;
    if ((alen = iob->io_resid) > len) {
        alen = len;
    }
    if (alen > 0) {
        memset(iob->io_base, 0, alen);
        iobuf_skip(iob, alen), len -= alen;
    }
    if (copiedp != NULL) {
        *copiedp = alen;
    }
    return (len == 0) ? 0 : -E_NO_MEM;
}

/*
 * iobuf_skip - change the current position of io buffer
 */
void
iobuf_skip(struct iobuf *iob, size_t n) {
    assert(iob->io_resid >= n);
    iob->io_base += n, iob->io_offset += n, iob->io_resid -= n;
}

[ucore/lab8_result/kern/fs/file.c] ++++++++++++++++++++++++++++++
#include <defs.h>
```

```c
#include <string.h>
#include <vfs.h>
#include <proc.h>
#include <file.h>
#include <unistd.h>
#include <iobuf.h>
#include <inode.h>
#include <stat.h>
#include <dirent.h>
#include <error.h>
#include <assert.h>

#define testfd(fd)                          ((fd) >= 0 && (fd) < FILES_STRUCT_NENTRY)

// get_fd_array - get current process's open files table
static struct file *
get_fd_array(void) {
    struct files_struct *filesp = current->filesp;
    assert(filesp != NULL && files_count(filesp) > 0);
    return filesp->fd_array;
}

// fd_array_init - initialize the open files table
void
fd_array_init(struct file *fd_array) {
    int fd;
    struct file *file = fd_array;
    for (fd = 0; fd < FILES_STRUCT_NENTRY; fd ++, file ++) {
        file->open_count = 0;
        file->status = FD_NONE, file->fd = fd;
    }
}

// fs_array_alloc - allocate a free file item (with FD_NONE status) in open files table
static int
fd_array_alloc(int fd, struct file **file_store) {
//    panic("debug");
    struct file *file = get_fd_array();
    if (fd == NO_FD) {
        for (fd = 0; fd < FILES_STRUCT_NENTRY; fd ++, file ++) {
            if (file->status == FD_NONE) {
                goto found;
            }
        }
        return -E_MAX_OPEN;
    }
    else {
        if (testfd(fd)) {
            file += fd;
            if (file->status == FD_NONE) {
                goto found;
            }
            return -E_BUSY;
        }
        return -E_INVAL;
    }
found:
    assert(fopen_count(file) == 0);
    file->status = FD_INIT, file->node = NULL;
    *file_store = file;
    return 0;
}

// fd_array_free - free a file item in open files table
static void
```

```
fd_array_free(struct file *file) {
    assert(file->status == FD_INIT || file->status == FD_CLOSED);
    assert(fopen_count(file) == 0);
    if (file->status == FD_CLOSED) {
        vfs_close(file->node);
    }
    file->status = FD_NONE;
}

static void
fd_array_acquire(struct file *file) {
    assert(file->status == FD_OPENED);
    fopen_count_inc(file);
}

// fd_array_release - file's open_count--; if file's open_count-- == 0 , then call fd_array_fr
ee to free this file item
static void
fd_array_release(struct file *file) {
    assert(file->status == FD_OPENED || file->status == FD_CLOSED);
    assert(fopen_count(file) > 0);
    if (fopen_count_dec(file) == 0) {
        fd_array_free(file);
    }
}

// fd_array_open - file's open_count++, set status to FD_OPENED
void
fd_array_open(struct file *file) {
    assert(file->status == FD_INIT && file->node != NULL);
    file->status = FD_OPENED;
    fopen_count_inc(file);
}

// fd_array_close - file's open_count--; if file's open_count-- == 0 , then call fd_array_free
 to free this file item
void
fd_array_close(struct file *file) {
    assert(file->status == FD_OPENED);
    assert(fopen_count(file) > 0);
    file->status = FD_CLOSED;
    if (fopen_count_dec(file) == 0) {
        fd_array_free(file);
    }
}

//fs_array_dup - duplicate file 'from'  to file 'to'
void
fd_array_dup(struct file *to, struct file *from) {
    //cprintf("[fd_array_dup]from fd=%d, to fd=%d\n",from->fd, to->fd);
    assert(to->status == FD_INIT && from->status == FD_OPENED);
    to->pos = from->pos;
    to->readable = from->readable;
    to->writable = from->writable;
    struct inode *node = from->node;
    vop_ref_inc(node), vop_open_inc(node);
    to->node = node;
    fd_array_open(to);
}

// fd2file - use fd as index of fd_array, return the array item (file)
static inline int
fd2file(int fd, struct file **file_store) {
    if (testfd(fd)) {
        struct file *file = get_fd_array() + fd;
```

```
        if (file->status == FD_OPENED && file->fd == fd) {
            *file_store = file;
            return 0;
        }
    }
    return -E_INVAL;
}

// file_testfd - test file is readble or writable?
bool
file_testfd(int fd, bool readable, bool writable) {
    int ret;
    struct file *file;
    if ((ret = fd2file(fd, &file)) != 0) {
        return 0;
    }
    if (readable && !file->readable) {
        return 0;
    }
    if (writable && !file->writable) {
        return 0;
    }
    return 1;
}

// open file
int
file_open(char *path, uint32_t open_flags) {
    bool readable = 0, writable = 0;
    switch (open_flags & O_ACCMODE) {
    case O_RDONLY: readable = 1; break;
    case O_WRONLY: writable = 1; break;
    case O_RDWR:
        readable = writable = 1;
        break;
    default:
        return -E_INVAL;
    }

    int ret;
    struct file *file;
    if ((ret = fd_array_alloc(NO_FD, &file)) != 0) {
        return ret;
    }

    struct inode *node;
    if ((ret = vfs_open(path, open_flags, &node)) != 0) {
        fd_array_free(file);
        return ret;
    }

    file->pos = 0;
    if (open_flags & O_APPEND) {
        struct stat __stat, *stat = &__stat;
        if ((ret = vop_fstat(node, stat)) != 0) {
            vfs_close(node);
            fd_array_free(file);
            return ret;
        }
        file->pos = stat->st_size;
    }

    file->node = node;
    file->readable = readable;
    file->writable = writable;
```

```
    fd_array_open(file);
    return file->fd;
}

// close file
int
file_close(int fd) {
    int ret;
    struct file *file;
    if ((ret = fd2file(fd, &file)) != 0) {
        return ret;
    }
    fd_array_close(file);
    return 0;
}

// read file
int
file_read(int fd, void *base, size_t len, size_t *copied_store) {
    int ret;
    struct file *file;
    *copied_store = 0;
    if ((ret = fd2file(fd, &file)) != 0) {
        return ret;
    }
    if (!file->readable) {
        return -E_INVAL;
    }
    fd_array_acquire(file);

    struct iobuf __iob, *iob = iobuf_init(&__iob, base, len, file->pos);
    ret = vop_read(file->node, iob);

    size_t copied = iobuf_used(iob);
    if (file->status == FD_OPENED) {
        file->pos += copied;
    }
    *copied_store = copied;
    fd_array_release(file);
    return ret;
}

// write file
int
file_write(int fd, void *base, size_t len, size_t *copied_store) {
    int ret;
    struct file *file;
    *copied_store = 0;
    if ((ret = fd2file(fd, &file)) != 0) {
        return ret;
    }
    if (!file->writable) {
        return -E_INVAL;
    }
    fd_array_acquire(file);

    struct iobuf __iob, *iob = iobuf_init(&__iob, base, len, file->pos);
    ret = vop_write(file->node, iob);

    size_t copied = iobuf_used(iob);
    if (file->status == FD_OPENED) {
        file->pos += copied;
    }
    *copied_store = copied;
    fd_array_release(file);
```

```
    return ret;
}

// seek file
int
file_seek(int fd, off_t pos, int whence) {
    struct stat __stat, *stat = &__stat;
    int ret;
    struct file *file;
    if ((ret = fd2file(fd, &file)) != 0) {
        return ret;
    }
    fd_array_acquire(file);

    switch (whence) {
    case LSEEK_SET: break;
    case LSEEK_CUR: pos += file->pos; break;
    case LSEEK_END:
        if ((ret = vop_fstat(file->node, stat)) == 0) {
            pos += stat->st_size;
        }
        break;
    default: ret = -E_INVAL;
    }

    if (ret == 0) {
        if ((ret = vop_tryseek(file->node, pos)) == 0) {
            file->pos = pos;
        }
//    cprintf("file_seek, pos=%d, whence=%d, ret=%d\n", pos, whence, ret);
    }
    fd_array_release(file);
    return ret;
}

// stat file
int
file_fstat(int fd, struct stat *stat) {
    int ret;
    struct file *file;
    if ((ret = fd2file(fd, &file)) != 0) {
        return ret;
    }
    fd_array_acquire(file);
    ret = vop_fstat(file->node, stat);
    fd_array_release(file);
    return ret;
}

// sync file
int
file_fsync(int fd) {
    int ret;
    struct file *file;
    if ((ret = fd2file(fd, &file)) != 0) {
        return ret;
    }
    fd_array_acquire(file);
    ret = vop_fsync(file->node);
    fd_array_release(file);
    return ret;
}

// get file entry in DIR
int
```

```
file_getdirentry(int fd, struct dirent *direntp) {
    int ret;
    struct file *file;
    if ((ret = fd2file(fd, &file)) != 0) {
        return ret;
    }
    fd_array_acquire(file);

    struct iobuf __iob, *iob = iobuf_init(&__iob, direntp->name, sizeof(direntp->name), dirent
p->offset);
    if ((ret = vop_getdirentry(file->node, iob)) == 0) {
        direntp->offset += iobuf_used(iob);
    }
    fd_array_release(file);
    return ret;
}

// duplicate file
int
file_dup(int fd1, int fd2) {
    int ret;
    struct file *file1, *file2;
    if ((ret = fd2file(fd1, &file1)) != 0) {
        return ret;
    }
    if ((ret = fd_array_alloc(fd2, &file2)) != 0) {
        return ret;
    }
    fd_array_dup(file2, file1);
    return file2->fd;
}


[ucore/lab8_result/kern/init/entry.S] ++++++++++++++++++++++++++++++
#include <mmu.h>
#include <memlayout.h>

#define REALLOC(x) (x - KERNBASE)

.text
.globl kern_entry
kern_entry:
    # load pa of boot pgdir
    movl $REALLOC(__boot_pgdir), %eax
    movl %eax, %cr3

    # enable paging
    movl %cr0, %eax
    orl $(CR0_PE | CR0_PG | CR0_AM | CR0_WP | CR0_NE | CR0_TS | CR0_EM | CR0_MP), %eax
    andl $~(CR0_TS | CR0_EM), %eax
    movl %eax, %cr0

    # update eip
    # now, eip = 0x1.....
    leal next, %eax
    # set eip = KERNBASE + 0x1.....
    jmp *%eax
next:

    # unmap va 0 ~ 4M, it's temporary mapping
    xorl %eax, %eax
    movl %eax, __boot_pgdir

    # set ebp, esp
    movl $0x0, %ebp
```

```
    # the kernel stack region is from bootstack -- bootstacktop,
    # the kernel stack size is KSTACKSIZE (8KB)defined in memlayout.h
    movl $bootstacktop, %esp
    # now kernel stack is ready , call the first C function
    call kern_init

# should never get here
spin:
    jmp spin

.data
.align PGSIZE
    .globl bootstack
bootstack:
    .space KSTACKSIZE
    .globl bootstacktop
bootstacktop:

# kernel builtin pgdir
# an initial page directory (Page Directory Table, PDT)
# These page directory table and page table can be reused!
.section .data.pgdir
.align PGSIZE
__boot_pgdir:
.globl __boot_pgdir
    # map va 0 ~ 4M to pa 0 ~ 4M (temporary)
    .long REALLOC(__boot_pt1) + (PTE_P | PTE_U | PTE_W)
    .space (KERNBASE >> PGSHIFT >> 10 << 2) - (. - __boot_pgdir) # pad to PDE of KERNBASE
    # map va KERNBASE + (0 ~ 4M) to pa 0 ~ 4M
    .long REALLOC(__boot_pt1) + (PTE_P | PTE_U | PTE_W)
    .space PGSIZE - (. - __boot_pgdir) # pad to PGSIZE

.set i, 0
__boot_pt1:
.rept 1024
    .long i * PGSIZE + (PTE_P | PTE_W)
    .set i, i + 1
.endr

[ucore/lab8_result/kern/init/init.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <string.h>
#include <console.h>
#include <kdebug.h>
#include <picirq.h>
#include <trap.h>
#include <clock.h>
#include <intr.h>
#include <pmm.h>
#include <vmm.h>
#include <ide.h>
#include <swap.h>
#include <proc.h>
#include <fs.h>

int kern_init(void) __attribute__((noreturn));

static void lab1_switch_test(void);

void t1()
{
        print_stackframe();
}
void t2()
```

```
{
        t1();
}
void t3()
{
        t2();
}
void t4()
{
        t2();
}

int
kern_init(void) {
        extern char edata[], end[];
        memset(edata, 0, end - edata);
        cons_init();
        const char *message = "(THU.CST) os is loading ...";
        cprintf("%s\n", message);
        cprintf("bootstack:0x%x, bootstacktop:0x%x\n", bootstack, bootstacktop);
        cprintf("edata:0x%x, end:0x%x\n", edata, end);

        print_kerninfo();
        print_stackframe();

        grade_backtrace();

        pmm_init();                 // init physical memory management

        pic_init();                 // init interrupt controller
        idt_init();                 // init interrupt descriptor table

        vmm_init();                 // init virtual memory management
        sched_init();               // init scheduler
        proc_init();                // init process table

        ide_init();                 // init ide devices
        swap_init();                // init swap
        fs_init();                  // init fs

        clock_init();               // init clock interrupt
        intr_enable();              // enable irq interrupt

        //LAB1: CAHLLENGE 1 If you try to do it, uncomment lab1_switch_test()
        // user/kernel mode switch test
        //lab1_switch_test();

        cpu_idle();                 // run idle process
}

void __attribute__((noinline))
        grade_backtrace2(int arg0, int arg1, int arg2, int arg3) {
                mon_backtrace(0, NULL, NULL);
        }

void __attribute__((noinline))
        grade_backtrace1(int arg0, int arg1) {
                grade_backtrace2(arg0, (int)&arg0, arg1, (int)&arg1);
        }

void __attribute__((noinline))
        grade_backtrace0(int arg0, int arg1, int arg2) {
                grade_backtrace1(arg0, arg2);
        }
```

```
void
grade_backtrace(void) {
        grade_backtrace0(0, (int)kern_init, 0xffff0000);
}

static void
lab1_print_cur_status(void) {
        static int round = 0;
        uint16_t reg1, reg2, reg3, reg4;
        asm volatile (
                        "mov %%cs, %0;"
                        "mov %%ds, %1;"
                        "mov %%es, %2;"
                        "mov %%ss, %3;"
                        : "=m"(reg1), "=m"(reg2), "=m"(reg3), "=m"(reg4));
        cprintf("%d: @ring %d\n", round, reg1 & 3);
        cprintf("%d:  cs = %x\n", round, reg1);
        cprintf("%d:  ds = %x\n", round, reg2);
        cprintf("%d:  es = %x\n", round, reg3);
        cprintf("%d:  ss = %x\n", round, reg4);
        round ++;
}

static void
lab1_switch_to_user(void) {
        //LAB1 CHALLENGE 1 : TODO
}

static void
lab1_switch_to_kernel(void) {
        //LAB1 CHALLENGE 1 :  TODO
}

static void
lab1_switch_test(void) {
        lab1_print_cur_status();
        cprintf("+++ switch to  user  mode +++\n");
        lab1_switch_to_user();
        lab1_print_cur_status();
        cprintf("+++ switch to kernel mode +++\n");
        lab1_switch_to_kernel();
        lab1_print_cur_status();
}

[ucore/lab8_result/kern/syscall/syscall.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <unistd.h>
#include <proc.h>
#include <syscall.h>
#include <trap.h>
#include <stdio.h>
#include <pmm.h>
#include <assert.h>
#include <clock.h>
#include <stat.h>
#include <dirent.h>
#include <sysfile.h>

static int
sys_exit(uint32_t arg[]) {
    int error_code = (int)arg[0];
    return do_exit(error_code);
}

static int
```

```
sys_fork(uint32_t arg[]) {
    struct trapframe *tf = current->tf;
    uintptr_t stack = tf->tf_esp;
    return do_fork(0, stack, tf);
}

static int
sys_wait(uint32_t arg[]) {
    int pid = (int)arg[0];
    int *store = (int *)arg[1];
    return do_wait(pid, store);
}

static int
sys_exec(uint32_t arg[]) {
    const char *name = (const char *)arg[0];
    int argc = (int)arg[1];
    const char **argv = (const char **)arg[2];
    return do_execve(name, argc, argv);
}

static int
sys_yield(uint32_t arg[]) {
    return do_yield();
}

static int
sys_kill(uint32_t arg[]) {
    int pid = (int)arg[0];
    return do_kill(pid);
}

static int
sys_getpid(uint32_t arg[]) {
    return current->pid;
}

static int
sys_putc(uint32_t arg[]) {
    int c = (int)arg[0];
    cputchar(c);
    return 0;
}

static int
sys_pgdir(uint32_t arg[]) {
    print_pgdir();
    return 0;
}

static uint32_t
sys_gettime(uint32_t arg[]) {
    return (int)ticks;
}
static uint32_t
sys_lab6_set_priority(uint32_t arg[])
{
    uint32_t priority = (uint32_t)arg[0];
    lab6_set_priority(priority);
    return 0;
}

static int
sys_sleep(uint32_t arg[]) {
    unsigned int time = (unsigned int)arg[0];
```

```
    return do_sleep(time);
}

static int
sys_open(uint32_t arg[]) {
    const char *path = (const char *)arg[0];
    uint32_t open_flags = (uint32_t)arg[1];
    return sysfile_open(path, open_flags);
}

static int
sys_close(uint32_t arg[]) {
    int fd = (int)arg[0];
    return sysfile_close(fd);
}

static int
sys_read(uint32_t arg[]) {
    int fd = (int)arg[0];
    void *base = (void *)arg[1];
    size_t len = (size_t)arg[2];
    return sysfile_read(fd, base, len);
}

static int
sys_write(uint32_t arg[]) {
    int fd = (int)arg[0];
    void *base = (void *)arg[1];
    size_t len = (size_t)arg[2];
    return sysfile_write(fd, base, len);
}

static int
sys_seek(uint32_t arg[]) {
    int fd = (int)arg[0];
    off_t pos = (off_t)arg[1];
    int whence = (int)arg[2];
    return sysfile_seek(fd, pos, whence);
}

static int
sys_fstat(uint32_t arg[]) {
    int fd = (int)arg[0];
    struct stat *stat = (struct stat *)arg[1];
    return sysfile_fstat(fd, stat);
}

static int
sys_fsync(uint32_t arg[]) {
    int fd = (int)arg[0];
    return sysfile_fsync(fd);
}

static int
sys_getcwd(uint32_t arg[]) {
    char *buf = (char *)arg[0];
    size_t len = (size_t)arg[1];
    return sysfile_getcwd(buf, len);
}

static int
sys_getdirentry(uint32_t arg[]) {
    int fd = (int)arg[0];
    struct dirent *direntp = (struct dirent *)arg[1];
    return sysfile_getdirentry(fd, direntp);
```

```
}

static int
sys_dup(uint32_t arg[]) {
    int fd1 = (int)arg[0];
    int fd2 = (int)arg[1];
    return sysfile_dup(fd1, fd2);
}

static int (*syscalls[])(uint32_t arg[]) = {
    [SYS_exit]              sys_exit,
    [SYS_fork]              sys_fork,
    [SYS_wait]              sys_wait,
    [SYS_exec]              sys_exec,
    [SYS_yield]             sys_yield,
    [SYS_kill]              sys_kill,
    [SYS_getpid]            sys_getpid,
    [SYS_putc]              sys_putc,
    [SYS_pgdir]             sys_pgdir,
    [SYS_gettime]           sys_gettime,
    [SYS_lab6_set_priority] sys_lab6_set_priority,
    [SYS_sleep]             sys_sleep,
    [SYS_open]              sys_open,
    [SYS_close]             sys_close,
    [SYS_read]              sys_read,
    [SYS_write]             sys_write,
    [SYS_seek]              sys_seek,
    [SYS_fstat]             sys_fstat,
    [SYS_fsync]             sys_fsync,
    [SYS_getcwd]            sys_getcwd,
    [SYS_getdirentry]       sys_getdirentry,
    [SYS_dup]               sys_dup,
};

#define NUM_SYSCALLS        ((sizeof(syscalls)) / (sizeof(syscalls[0])))

void
syscall(void) {
    struct trapframe *tf = current->tf;
    uint32_t arg[5];
    int num = tf->tf_regs.reg_eax;
    if (num >= 0 && num < NUM_SYSCALLS) {
        if (syscalls[num] != NULL) {
            arg[0] = tf->tf_regs.reg_edx;
            arg[1] = tf->tf_regs.reg_ecx;
            arg[2] = tf->tf_regs.reg_ebx;
            arg[3] = tf->tf_regs.reg_edi;
            arg[4] = tf->tf_regs.reg_esi;
            tf->tf_regs.reg_eax = syscalls[num](arg);
            return ;
        }
    }
    print_trapframe(tf);
    panic("undefined syscall %d, pid = %d, name = %s.\n",
            num, current->pid, current->name);
}

[ucore/lab8_result/kern/syscall/syscall.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SYSCALL_SYSCALL_H__
#define __KERN_SYSCALL_SYSCALL_H__

void syscall(void);

#endif /* !__KERN_SYSCALL_SYSCALL_H__ */
```

[ucore/lab8_result/kern/sync/monitor.c] ++++++++++++++++++++++++++++++

```c
#include <stdio.h>
#include <monitor.h>
#include <kmalloc.h>
#include <assert.h>


// Initialize monitor.
void
monitor_init (monitor_t * mtp, size_t num_cv) {
    int i;
    assert(num_cv>0);
    mtp->next_count = 0;
    mtp->cv = NULL;
    sem_init(&(mtp->mutex), 1); //unlocked
    sem_init(&(mtp->next), 0);
    mtp->cv =(condvar_t *) kmalloc(sizeof(condvar_t)*num_cv);
    assert(mtp->cv!=NULL);
    for(i=0; i<num_cv; i++){
        mtp->cv[i].count=0;
        sem_init(&(mtp->cv[i].sem),0);
        mtp->cv[i].owner=mtp;
    }
}

// Unlock one of threads waiting on the condition variable.
void
cond_signal (condvar_t *cvp) {
   //LAB7 EXERCISE1: YOUR CODE
   cprintf("cond_signal begin: cvp %x, cvp->count %d, cvp->owner->next_count %d\n", cvp, cvp->
count, cvp->owner->next_count);
  /*
   *      cond_signal(cv) {
   *          if(cv.count>0) {
   *             mt.next_count ++;
   *             signal(cv.sem);
   *             wait(mt.next);
   *             mt.next_count--;
   *          }
   *       }
   */
     if(cvp->count>0) {
        cvp->owner->next_count ++;
        up(&(cvp->sem));
        down(&(cvp->owner->next));
        cvp->owner->next_count --;
      }
   cprintf("cond_signal end: cvp %x, cvp->count %d, cvp->owner->next_count %d\n", cvp, cvp->co
unt, cvp->owner->next_count);
}

// Suspend calling thread on a condition variable waiting for condition Atomically unlocks
// mutex and suspends calling thread on conditional variable after waking up locks mutex. Noti
ce: mp is mutex semaphore for monitor's procedures
void
cond_wait (condvar_t *cvp) {
    //LAB7 EXERCISE1: YOUR CODE
    cprintf("cond_wait begin:  cvp %x, cvp->count %d, cvp->owner->next_count %d\n", cvp, cvp->
count, cvp->owner->next_count);
   /*
    *         cv.count ++;
    *         if(mt.next_count>0)
    *            signal(mt.next)
    *         else
    *            signal(mt.mutex);
```

```
     *         wait(cv.sem);
     *         cv.count --;
     */
       cvp->count++;
       if(cvp->owner->next_count > 0)
          up(&(cvp->owner->next));
       else
          up(&(cvp->owner->mutex));
       down(&(cvp->sem));
       cvp->count --;
     cprintf("cond_wait end:  cvp %x, cvp->count %d, cvp->owner->next_count %d\n", cvp, cvp->co
unt, cvp->owner->next_count);
}
[ucore/lab8_result/kern/sync/wait.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SYNC_WAIT_H__
#define __KERN_SYNC_WAIT_H__

#include <list.h>

typedef struct {
    list_entry_t wait_head;
} wait_queue_t;

struct proc_struct;

typedef struct {
    struct proc_struct *proc;
    uint32_t wakeup_flags;
    wait_queue_t *wait_queue;
    list_entry_t wait_link;
} wait_t;

#define le2wait(le, member)         \
    to_struct((le), wait_t, member)

void wait_init(wait_t *wait, struct proc_struct *proc);
void wait_queue_init(wait_queue_t *queue);
void wait_queue_add(wait_queue_t *queue, wait_t *wait);
void wait_queue_del(wait_queue_t *queue, wait_t *wait);

wait_t *wait_queue_next(wait_queue_t *queue, wait_t *wait);
wait_t *wait_queue_prev(wait_queue_t *queue, wait_t *wait);
wait_t *wait_queue_first(wait_queue_t *queue);
wait_t *wait_queue_last(wait_queue_t *queue);

bool wait_queue_empty(wait_queue_t *queue);
bool wait_in_queue(wait_t *wait);
void wakeup_wait(wait_queue_t *queue, wait_t *wait, uint32_t wakeup_flags, bool del);
void wakeup_first(wait_queue_t *queue, uint32_t wakeup_flags, bool del);
void wakeup_queue(wait_queue_t *queue, uint32_t wakeup_flags, bool del);

void wait_current_set(wait_queue_t *queue, wait_t *wait, uint32_t wait_state);

#define wait_current_del(queue, wait)                                       \
    do {                                                                    \
        if (wait_in_queue(wait)) {                                          \
            wait_queue_del(queue, wait);                                    \
        }                                                                   \
    } while (0)

#endif /* !__KERN_SYNC_WAIT_H__ */

[ucore/lab8_result/kern/sync/wait.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <list.h>
```

```c
#include <sync.h>
#include <wait.h>
#include <proc.h>

void
wait_init(wait_t *wait, struct proc_struct *proc) {
    wait->proc = proc;
    wait->wakeup_flags = WT_INTERRUPTED;
    list_init(&(wait->wait_link));
}

void
wait_queue_init(wait_queue_t *queue) {
    list_init(&(queue->wait_head));
}

void
wait_queue_add(wait_queue_t *queue, wait_t *wait) {
    assert(list_empty(&(wait->wait_link)) && wait->proc != NULL);
    wait->wait_queue = queue;
    list_add_before(&(queue->wait_head), &(wait->wait_link));
}

void
wait_queue_del(wait_queue_t *queue, wait_t *wait) {
    assert(!list_empty(&(wait->wait_link)) && wait->wait_queue == queue);
    list_del_init(&(wait->wait_link));
}

wait_t *
wait_queue_next(wait_queue_t *queue, wait_t *wait) {
    assert(!list_empty(&(wait->wait_link)) && wait->wait_queue == queue);
    list_entry_t *le = list_next(&(wait->wait_link));
    if (le != &(queue->wait_head)) {
        return le2wait(le, wait_link);
    }
    return NULL;
}

wait_t *
wait_queue_prev(wait_queue_t *queue, wait_t *wait) {
    assert(!list_empty(&(wait->wait_link)) && wait->wait_queue == queue);
    list_entry_t *le = list_prev(&(wait->wait_link));
    if (le != &(queue->wait_head)) {
        return le2wait(le, wait_link);
    }
    return NULL;
}

wait_t *
wait_queue_first(wait_queue_t *queue) {
    list_entry_t *le = list_next(&(queue->wait_head));
    if (le != &(queue->wait_head)) {
        return le2wait(le, wait_link);
    }
    return NULL;
}

wait_t *
wait_queue_last(wait_queue_t *queue) {
    list_entry_t *le = list_prev(&(queue->wait_head));
    if (le != &(queue->wait_head)) {
        return le2wait(le, wait_link);
    }
    return NULL;
```

```
}

bool
wait_queue_empty(wait_queue_t *queue) {
    return list_empty(&(queue->wait_head));
}

bool
wait_in_queue(wait_t *wait) {
    return !list_empty(&(wait->wait_link));
}

void
wakeup_wait(wait_queue_t *queue, wait_t *wait, uint32_t wakeup_flags, bool del) {
    if (del) {
        wait_queue_del(queue, wait);
    }
    wait->wakeup_flags = wakeup_flags;
    wakeup_proc(wait->proc);
}

void
wakeup_first(wait_queue_t *queue, uint32_t wakeup_flags, bool del) {
    wait_t *wait;
    if ((wait = wait_queue_first(queue)) != NULL) {
        wakeup_wait(queue, wait, wakeup_flags, del);
    }
}

void
wakeup_queue(wait_queue_t *queue, uint32_t wakeup_flags, bool del) {
    wait_t *wait;
    if ((wait = wait_queue_first(queue)) != NULL) {
        if (del) {
            do {
                wakeup_wait(queue, wait, wakeup_flags, 1);
            } while ((wait = wait_queue_first(queue)) != NULL);
        }
        else {
            do {
                wakeup_wait(queue, wait, wakeup_flags, 0);
            } while ((wait = wait_queue_next(queue, wait)) != NULL);
        }
    }
}

void
wait_current_set(wait_queue_t *queue, wait_t *wait, uint32_t wait_state) {
    assert(current != NULL);
    wait_init(wait, current);
    current->state = PROC_SLEEPING;
    current->wait_state = wait_state;
    wait_queue_add(queue, wait);
}

[ucore/lab8_result/kern/sync/sem.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <wait.h>
#include <atomic.h>
#include <kmalloc.h>
#include <sem.h>
#include <proc.h>
#include <sync.h>
#include <assert.h>
```

```
void
sem_init(semaphore_t *sem, int value) {
    sem->value = value;
    wait_queue_init(&(sem->wait_queue));
}

static __noinline void __up(semaphore_t *sem, uint32_t wait_state) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        wait_t *wait;
        if ((wait = wait_queue_first(&(sem->wait_queue))) == NULL) {
            sem->value ++;
        }
        else {
            assert(wait->proc->wait_state == wait_state);
            wakeup_wait(&(sem->wait_queue), wait, wait_state, 1);
        }
    }
    local_intr_restore(intr_flag);
}

static __noinline uint32_t __down(semaphore_t *sem, uint32_t wait_state) {
    bool intr_flag;
    local_intr_save(intr_flag);
    if (sem->value > 0) {
        sem->value --;
        local_intr_restore(intr_flag);
        return 0;
    }
    wait_t __wait, *wait = &__wait;
    wait_current_set(&(sem->wait_queue), wait, wait_state);
    local_intr_restore(intr_flag);

    schedule();

    local_intr_save(intr_flag);
    wait_current_del(&(sem->wait_queue), wait);
    local_intr_restore(intr_flag);

    if (wait->wakeup_flags != wait_state) {
        return wait->wakeup_flags;
    }
    return 0;
}

void
up(semaphore_t *sem) {
    __up(sem, WT_KSEM);
}

void
down(semaphore_t *sem) {
    uint32_t flags = __down(sem, WT_KSEM);
    assert(flags == 0);
}

bool
try_down(semaphore_t *sem) {
    bool intr_flag, ret = 0;
    local_intr_save(intr_flag);
    if (sem->value > 0) {
        sem->value --, ret = 1;
    }
    local_intr_restore(intr_flag);
```

```
    return ret;
}

[ucore/lab8_result/kern/sync/sync.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SYNC_SYNC_H__
#define __KERN_SYNC_SYNC_H__

#include <x86.h>
#include <intr.h>
#include <mmu.h>
#include <assert.h>
#include <atomic.h>
#include <sched.h>

static inline bool
__intr_save(void) {
    if (read_eflags() & FL_IF) {
        intr_disable();
        return 1;
    }
    return 0;
}

static inline void
__intr_restore(bool flag) {
    if (flag) {
        intr_enable();
    }
}

#define local_intr_save(x)      do { x = __intr_save(); } while (0)
#define local_intr_restore(x)   __intr_restore(x);

#endif /* !__KERN_SYNC_SYNC_H__ */

[ucore/lab8_result/kern/sync/sem.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SYNC_SEM_H__
#define __KERN_SYNC_SEM_H__

#include <defs.h>
#include <atomic.h>
#include <wait.h>

typedef struct {
    int value;
    wait_queue_t wait_queue;
} semaphore_t;

void sem_init(semaphore_t *sem, int value);
void up(semaphore_t *sem);
void down(semaphore_t *sem);
bool try_down(semaphore_t *sem);

#endif /* !__KERN_SYNC_SEM_H__ */

[ucore/lab8_result/kern/sync/check_sync.c] ++++++++++++++++++++++++++++++
#include <stdio.h>
#include <proc.h>
#include <sem.h>
#include <monitor.h>
#include <assert.h>

#define N 5 /* å\223²å-¦å®¶æ\225°ç\233® */
#define LEFT (i-1+N)%N /* iç\232\204å•¦é\202»å\217•ç \201 */
#define RIGHT (i+1)%N /* iç\232\204å\217³é\202»å\217•ç \201 */
```

```
#define THINKING 0 /* å\223²å-¦å®¶æ-£å\234¨æ\200\235è\200\203 */
#define HUNGRY 1 /* å\223²å-¦å®¶æ\203³å\217\226å¾\227å\217\211å-\220 */
#define EATING 2 /* å\223²å-¦å®¶æ-£å\234¨å\220\203é\235¢ */
#define TIMES  4 /* å\220\2034æ¬¡é¥- */
#define SLEEP_TIME 10

//-----------------philosopher problem using monitor ------------
/*PSEUDO CODE :philosopher problem using semaphore
system DINING_PHILOSOPHERS

VAR
me:    semaphore, initially 1;                    # for mutual exclusion
s[5]:  semaphore s[5], initially 0;               # for synchronization
pflag[5]: {THINK, HUNGRY, EAT}, initially THINK;  # philosopher flag

# As before, each philosopher is an endless cycle of thinking and eating.

procedure philosopher(i)
  {
    while TRUE do
     {
       THINKING;
       take_chopsticks(i);
       EATING;
       drop_chopsticks(i);
     }
  }

# The take_chopsticks procedure involves checking the status of neighboring
# philosophers and then declaring one's own intention to eat. This is a two-phase
# protocol; first declaring the status HUNGRY, then going on to EAT.

procedure take_chopsticks(i)
  {
    DOWN(me);                # critical section
    pflag[i] := HUNGRY;
    test[i];
    UP(me);                  # end critical section
    DOWN(s[i])               # Eat if enabled
   }

void test(i)                 # Let phil[i] eat, if waiting
  {
    if ( pflag[i] == HUNGRY
      && pflag[i-1] != EAT
      && pflag[i+1] != EAT)
       then
        {
          pflag[i] := EAT;
          UP(s[i])
        }
    }


# Once a philosopher finishes eating, all that remains is to relinquish the
# resources---its two chopsticks---and thereby release waiting neighbors.

void drop_chopsticks(int i)
  {
    DOWN(me);                 # critical section
    test(i-1);                # Let phil. on left eat if possible
    test(i+1);                # Let phil. on rght eat if possible
    UP(me);                   # up critical section
   }
```

```
*/


//---------- philosophers problem using semaphore ----------------------
int state_sema[N]; /* è®°å½\225æ¯\217ä¸ªäº°ç\212¶æ\200\201ç\232\204æ\225°ç»\204 */
/* ä¿¡å\217•é\207\217æ\230¯ä¸\200ä¸ªç\211¹æ®\212ç\232\204æ\225´å\236\213å\217\230é\207\217 */
semaphore_t mutex; /* ä¸´ç\225\214å\214°ä°\222æ\226¥ */
semaphore_t s[N]; /* æ¯\217ä¸ªå\223²å-¦å®¶ä¸\200ä¸ªä¿¡å\217•é\207\217 */

struct proc_struct *philosopher_proc_sema[N];

void phi_test_sema(i) /* iï¼\232å\223²å-¦å®¶å\217•ç \201ä»\2160å\210°N-1 */
{
    if(state_sema[i]==HUNGRY&&state_sema[LEFT]!=EATING
            &&state_sema[RIGHT]!=EATING)
    {
        state_sema[i]=EATING;
        up(&s[i]);
    }
}

void phi_take_forks_sema(int i) /* iï¼\232å\223²å-¦å®¶å\217•ç \201ä»\2160å\210°N-1 */
{
        down(&mutex); /* è¿\233å\205¥ä¸´ç\225\214å\214° */
        state_sema[i]=HUNGRY; /* è®°å½\225ä¸\213å\223²å-¦å®¶i饥¥é¥¿ç\232\204ä°\213å®\236 */
        phi_test_sema(i); /* è¯\225å\233¾å¾\227å\210°ä¸¤å\217ªå\217\211å-\220 */
        up(&mutex); /* ç¦»å¼\200ä¸´ç\225\214å\214° */
        down(&s[i]); /* å¦\202æ\236\234å¾\227ä¸\215å\210°å\217\211å-\220å°±é\230»å¡\236 */
}

void phi_put_forks_sema(int i) /* iï¼\232å\223²å-¦å®¶å\217•ç \201ä»\2160å\210°N-1 */
{
        down(&mutex); /* è¿\233å\205¥ä¸´ç\225\214å\214° */
        state_sema[i]=THINKING; /* å\223²å-¦å®¶è¿\233é¤\220ç»\223æ\235\237 */
        phi_test_sema(LEFT); /* ç\234\213ä¸\200ä¸\213å•¦é\202»å±\205ç\216°å\234¨æ\230¯å\220¦è
\203½è¿\233é¤\220 */
        phi_test_sema(RIGHT); /* ç\234\213ä¸\200ä¸\213å\217³é\202»å±\205ç\216°å\234¨æ\230¯å
\220¦è\203½è¿\233é¤\220 */
        up(&mutex); /* ç¦»å¼\200ä¸´ç\225\214å\214° */
}

int philosopher_using_semaphore(void * arg) /* iï¼\232å\223²å-¦å®¶å\217•ç \201ï¼\214ä»\2160å
\210°N-1 */
{
    int i, iter=0;
    i=(int)arg;
    cprintf("I am No.%d philosopher_sema\n",i);
    while(iter++<TIMES)
    { /* æ\227 é\231\220å¾ªç\216¯ */
        cprintf("Iter %d, No.%d philosopher_sema is thinking\n",iter,i); /* å\223²å-¦å®¶æ-£å
\234¨æ\200\235è\200\203 */
        do_sleep(SLEEP_TIME);
        phi_take_forks_sema(i);
        /* é\234\200è¦\201ä¸¤å\217ªå\217\211å-\220ï¼\214æ\210\226è\200\205é\230»å¡\236 */
        cprintf("Iter %d, No.%d philosopher_sema is eating\n",iter,i); /* è¿\233é¤\220 */
        do_sleep(SLEEP_TIME);
        phi_put_forks_sema(i);
        /* æ\212\212ä¸¤æ\212\212å\217\211å-\220å\220\214æ\227¶æ\224¾å\233\236æ¡\214å-\220 */
    }
    cprintf("No.%d philosopher_sema quit\n",i);
    return 0;
}

//-----------------philosopher problem using monitor ------------
/*PSEUDO CODE :philosopher problem using monitor
```

```
 * monitor dp
 * {
 *  enum {thinking, hungry, eating} state[5];
 *  condition self[5];
 *
 *  void pickup(int i) {
 *      state[i] = hungry;
 *      if ((state[(i+4)%5] != eating) && (state[(i+1)%5] != eating)) {
 *        state[i] = eating;
 *      else
 *         self[i].wait();
 *   }
 *
 *   void putdown(int i) {
 *      state[i] = thinking;
 *      if ((state[(i+4)%5] == hungry) && (state[(i+3)%5] != eating)) {
 *          state[(i+4)%5] = eating;
 *          self[(i+4)%5].signal();
 *      }
 *      if ((state[(i+1)%5] == hungry) && (state[(i+2)%5] != eating)) {
 *          state[(i+1)%5] = eating;
 *          self[(i+1)%5].signal();
 *      }
 *   }
 *
 *   void init() {
 *      for (int i = 0; i < 5; i++)
 *         state[i] = thinking;
 *   }
 * }
 */

struct proc_struct *philosopher_proc_condvar[N]; // N philosopher
int state_condvar[N];                            // the philosopher's state: EATING, HUNGARY,
THINKING
monitor_t mt, *mtp=&mt;                          // monitor

void phi_test_condvar (i) {
    if(state_condvar[i]==HUNGRY&&state_condvar[LEFT]!=EATING
            &&state_condvar[RIGHT]!=EATING) {
        cprintf("phi_test_condvar: state_condvar[%d] will eating\n",i);
        state_condvar[i] = EATING ;
        cprintf("phi_test_condvar: signal self_cv[%d] \n",i);
        cond_signal(&mtp->cv[i]) ;
    }
}


void phi_take_forks_condvar(int i) {
     down(&(mtp->mutex));
//--------into routine in monitor--------------
     // LAB7 EXERCISE1: YOUR CODE
     // I am hungry
     // try to get fork
      // I am hungry
      state_condvar[i]=HUNGRY;
      // try to get fork
      phi_test_condvar(i);
      if (state_condvar[i] != EATING) {
          cprintf("phi_take_forks_condvar: %d didn't get fork and will wait\n",i);
          cond_wait(&mtp->cv[i]);
      }
//--------leave routine in monitor--------------
      if(mtp->next_count>0)
         up(&(mtp->next));
```

```
      else
         up(&(mtp->mutex));
}

void phi_put_forks_condvar(int i) {
     down(&(mtp->mutex));

//--------into routine in monitor--------------
     // LAB7 EXERCISE1: YOUR CODE
     // I ate over
     // test left and right neighbors
      // I ate over
      state_condvar[i]=THINKING;
      // test left and right neighbors
      phi_test_condvar(LEFT);
      phi_test_condvar(RIGHT);
//--------leave routine in monitor--------------
     if(mtp->next_count>0)
        up(&(mtp->next));
     else
        up(&(mtp->mutex));
}

//---------- philosophers using monitor (condition variable) ----------------------
int philosopher_using_condvar(void * arg) { /* arg is the No. of philosopher 0~N-1*/

    int i, iter=0;
    i=(int)arg;
    cprintf("I am No.%d philosopher_condvar\n",i);
    while(iter++<TIMES)
    { /* iterate*/
        cprintf("Iter %d, No.%d philosopher_condvar is thinking\n",iter,i); /* thinking*/
        do_sleep(SLEEP_TIME);
        phi_take_forks_condvar(i);
        /* need two forks, maybe blocked */
        cprintf("Iter %d, No.%d philosopher_condvar is eating\n",iter,i); /* eating*/
        do_sleep(SLEEP_TIME);
        phi_put_forks_condvar(i);
        /* return two forks back*/
    }
    cprintf("No.%d philosopher_condvar quit\n",i);
    return 0;
}

void check_sync(void){

    int i;

    //check semaphore
    sem_init(&mutex, 1);
    for(i=0;i<N;i++){
        sem_init(&s[i], 0);
        int pid = kernel_thread(philosopher_using_semaphore, (void *)i, 0);
        if (pid <= 0) {
            panic("create No.%d philosopher_using_semaphore failed.\n");
        }
        philosopher_proc_sema[i] = find_proc(pid);
        set_proc_name(philosopher_proc_sema[i], "philosopher_sema_proc");
    }

    //check condition variable
    monitor_init(&mt, N);
    for(i=0;i<N;i++){
        state_condvar[i]=THINKING;
        int pid = kernel_thread(philosopher_using_condvar, (void *)i, 0);
```

```
        if (pid <= 0) {
            panic("create No.%d philosopher_using_condvar failed.\n");
        }
        philosopher_proc_condvar[i] = find_proc(pid);
        set_proc_name(philosopher_proc_condvar[i], "philosopher_condvar_proc");
    }
}
[ucore/lab8_result/kern/sync/monitor.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SYNC_MONITOR_CONDVAR_H__
#define __KERN_SYNC_MOINTOR_CONDVAR_H__

#include <sem.h>
/* In [OS CONCEPT] 7.7 section, the accurate define and approximate implementation of MONITOR
was introduced.
 * INTRODUCTION:
 *  Monitors were invented by C. A. R. Hoare and Per Brinch Hansen, and were first implemented
 in Brinch Hansen's
 *  Concurrent Pascal language. Generally, a monitor is a language construct and the compiler
usually enforces mutual exclusion. Compare this with semaphores, which are usually an OS const
ruct.
 * DEFNIE & CHARACTERISTIC:
 *  A monitor is a collection of procedures, variables, and data structures grouped together.
 *  Processes can call the monitor procedures but cannot access the internal data structures.
 *  Only one process at a time may be be active in a monitor.
 *  Condition variables allow for blocking and unblocking.
 *     cv.wait() blocks a process.
 *        The process is said to be waiting for (or waiting on) the condition variable cv.
 *     cv.signal() (also called cv.notify) unblocks a process waiting for the condition variab
le cv.
 *        When this occurs, we need to still require that only one process is active in the mo
nitor. This can be done in several ways:
 *           on some systems the old process (the one executing the signal) leaves the monito
r and the new one enters
 *           on some systems the signal must be the last statement executed inside the monito
r.
 *           on some systems the old process will block until the monitor is available again.
 *           on some systems the new process (the one unblocked by the signal) will remain bl
ocked until the monitor is available again.
 *   If a condition variable is signaled with nobody waiting, the signal is lost. Compare this
 with semaphores, in which a signal will allow a process that executes a wait in the future to
 no block.
 *   You should not think of a condition variable as a variable in the traditional sense.
 *     It does not have a value.
 *     Think of it as an object in the OOP sense.
 *     It has two methods, wait and signal that manipulate the calling process.
 * IMPLEMENTATION:
 *   monitor mt {
 *     ----------------variable-----------------
 *     semaphore mutex;
 *     semaphore next;
 *     int next_count;
 *     condvar {int count, sempahore sem}  cv[N];
 *     other variables in mt;
 *     --------condvar wait/signal--------------
 *     cond_wait (cv) {
 *         cv.count ++;
 *         if(mt.next_count>0)
 *            signal(mt.next)
 *         else
 *            signal(mt.mutex);
 *         wait(cv.sem);
 *         cv.count --;
 *      }
 *
 *      cond_signal(cv) {
```

```
 *          if(cv.count>0) {
 *             mt.next_count ++;
 *             signal(cv.sem);
 *             wait(mt.next);
 *             mt.next_count--;
 *          }
 *       }
 *     --------routines in monitor---------------
 *     routineA_in_mt () {
 *        wait(mt.mutex);
 *        ...
 *        real body of routineA
 *        ...
 *        if(next_count>0)
 *            signal(mt.next);
 *        else
 *            signal(mt.mutex);
 *     }
 */

typedef struct monitor monitor_t;

typedef struct condvar{
    semaphore_t sem;        // the sem semaphore  is used to down the waiting proc, and the si
gnaling proc should up the waiting proc
    int count;              // the number of waiters on condvar
    monitor_t * owner;      // the owner(monitor) of this condvar
} condvar_t;

typedef struct monitor{
    semaphore_t mutex;      // the mutex lock for going into the routines in monitor, should b
e initialized to 1
    semaphore_t next;       // the next semaphore is used to down the signaling proc itself, a
nd the other OR wakeuped waiting proc should wake up the sleeped signaling proc.
    int next_count;         // the number of of sleeped signaling proc
    condvar_t *cv;          // the condvars in monitor
} monitor_t;

// Initialize variables in monitor.
void     monitor_init (monitor_t *cvp, size_t num_cv);
// Unlock one of threads waiting on the condition variable.
void     cond_signal (condvar_t *cvp);
// Suspend calling thread on a condition variable waiting for condition atomically unlock mute
x in monitor,
// and suspends calling thread on conditional variable after waking up locks mutex.
void     cond_wait (condvar_t *cvp);

#endif /* !__KERN_SYNC_MONITOR_CONDVAR_H__ */
[ucore/lab8_result/kern/libs/readline.c] ++++++++++++++++++++++++++++++
#include <stdio.h>

#define BUFSIZE 1024
static char buf[BUFSIZE];

/* *
 * readline - get a line from stdin
 * @prompt:     the string to be written to stdout
 *
 * The readline() function will write the input string @prompt to
 * stdout first. If the @prompt is NULL or the empty string,
 * no prompt is issued.
 *
 * This function will keep on reading characters and saving them to buffer
 * 'buf' until '\n' or '\r' is encountered.
 *
```

```
 * Note that, if the length of string that will be read is longer than
 * buffer size, the end of string will be discarded.
 *
 * The readline() function returns the text of the line read. If some errors
 * are happened, NULL is returned. The return value is a global variable,
 * thus it should be copied before it is used.
 * */
char *
readline(const char *prompt) {
    if (prompt != NULL) {
        cprintf("%s", prompt);
    }
    int i = 0, c;
    while (1) {
        c = getchar();
        if (c < 0) {
            return NULL;
        }
        else if (c >= ' ' && i < BUFSIZE - 1) {
            cputchar(c);
            buf[i ++] = c;
        }
        else if (c == '\b' && i > 0) {
            cputchar(c);
            i --;
        }
        else if (c == '\n' || c == '\r') {
            cputchar(c);
            buf[i] = '\0';
            return buf;
        }
    }
}
```

[ucore/lab8_result/kern/libs/string.c] ++++++++++++++++++++++++++++++

```
#include <string.h>
#include <kmalloc.h>

char *
strdup(const char *src) {
    char *dst;
    size_t len = strlen(src);
    if ((dst = kmalloc(len + 1)) != NULL) {
        memcpy(dst, src, len);
        dst[len] = '\0';
    }
    return dst;
}

char *
stradd(const char *src1, const char *src2) {
    char *ret, *dst;
    size_t len1 = strlen(src1), len2 = strlen(src2);
    if ((ret = dst = kmalloc(len1 + len2 + 1)) != NULL) {
        memcpy(dst, src1, len1), dst += len1;
        memcpy(dst, src2, len2), dst += len2;
        *dst = '\0';
    }
    return ret;
}
```

[ucore/lab8_result/kern/libs/stdio.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <stdio.h>
#include <console.h>
```

```c
#include <unistd.h>
/* HIGH level console I/O */

/* *
 * cputch - writes a single character @c to stdout, and it will
 * increace the value of counter pointed by @cnt.
 * */
static void
cputch(int c, int *cnt) {
    cons_putc(c);
    (*cnt) ++;
}

/* *
 * vcprintf - format a string and writes it to stdout
 *
 * The return value is the number of characters which would be
 * written to stdout.
 *
 * Call this function if you are already dealing with a va_list.
 * Or you probably want cprintf() instead.
 * */
int
vcprintf(const char *fmt, va_list ap) {
    int cnt = 0;
    vprintfmt((void*)cputch, NO_FD, &cnt, fmt, ap);
    return cnt;
}

/* *
 * cprintf - formats a string and writes it to stdout
 *
 * The return value is the number of characters which would be
 * written to stdout.
 * */
int
cprintf(const char *fmt, ...) {
    va_list ap;
    int cnt;
    va_start(ap, fmt);
    cnt = vcprintf(fmt, ap);
    va_end(ap);
    return cnt;
}

/* cputchar - writes a single character to stdout */
void
cputchar(int c) {
    cons_putc(c);
}

/* *
 * cputs- writes the string pointed by @str to stdout and
 * appends a newline character.
 * */
int
cputs(const char *str) {
    int cnt = 0;
    char c;
    while ((c = *str ++) != '\0') {
        cputch(c, &cnt);
    }
    cputch('\n', &cnt);
    return cnt;
}
```

```
/* getchar - reads a single non-zero character from stdin */
int
getchar(void) {
    int c;
    while ((c = cons_getc()) == 0)
        /* do nothing */;
    return c;
}

[ucore/lab8_result/kern/trap/trapentry.S] ++++++++++++++++++++++++++++++
#include <memlayout.h>

# vectors.S sends all traps here.
.text
.globl __alltraps
__alltraps:
    # push registers to build a trap frame
    # therefore make the stack look like a struct trapframe
    pushl %ds
    pushl %es
    pushl %fs
    pushl %gs
    pushal

    # load GD_KDATA into %ds and %es to set up data segments for kernel
    movl $GD_KDATA, %eax
    movw %ax, %ds
    movw %ax, %es

    # push %esp to pass a pointer to the trapframe as an argument to trap()
    pushl %esp

    # call trap(tf), where tf=%esp
    call trap

    # pop the pushed stack pointer
    popl %esp

    # return falls through to trapret...
.globl __trapret
__trapret:
    # restore registers from stack
    popal

    # restore %ds, %es, %fs and %gs
    popl %gs
    popl %fs
    popl %es
    popl %ds

    # get rid of the trap number and error code
    addl $0x8, %esp
    iret

.globl forkrets
forkrets:
    # set stack to this new process's trapframe
    movl 4(%esp), %esp
    jmp __trapret
[ucore/lab8_result/kern/trap/trap.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_TRAP_TRAP_H__
#define __KERN_TRAP_TRAP_H__

#include <defs.h>
```

```
/* Trap Numbers */

/* Processor-defined: */
#define T_DIVIDE                0   // divide error
#define T_DEBUG                 1   // debug exception
#define T_NMI                   2   // non-maskable interrupt
#define T_BRKPT                 3   // breakpoint
#define T_OFLOW                 4   // overflow
#define T_BOUND                 5   // bounds check
#define T_ILLOP                 6   // illegal opcode
#define T_DEVICE                7   // device not available
#define T_DBLFLT                8   // double fault
// #define T_COPROC             9   // reserved (not used since 486)
#define T_TSS                   10  // invalid task switch segment
#define T_SEGNP                 11  // segment not present
#define T_STACK                 12  // stack exception
#define T_GPFLT                 13  // general protection fault
#define T_PGFLT                 14  // page fault
// #define T_RES                15  // reserved
#define T_FPERR                 16  // floating point error
#define T_ALIGN                 17  // aligment check
#define T_MCHK                  18  // machine check
#define T_SIMDERR               19  // SIMD floating point error

/* Hardware IRQ numbers. We receive these as (IRQ_OFFSET + IRQ_xx) */
#define IRQ_OFFSET              32  // IRQ 0 corresponds to int IRQ_OFFSET

#define IRQ_TIMER               0
#define IRQ_KBD                 1
#define IRQ_COM1                4
#define IRQ_IDE1                14
#define IRQ_IDE2                15
#define IRQ_ERROR               19
#define IRQ_SPURIOUS            31

/* *
 * These are arbitrarily chosen, but with care not to overlap
 * processor defined exceptions or interrupt vectors.
 * */
#define T_SWITCH_TOU                120    // user/kernel switch
#define T_SWITCH_TOK                121    // user/kernel switch

/* registers as pushed by pushal */
struct pushregs {
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp;          /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
};

struct trapframe {
    struct pushregs tf_regs;
    uint16_t tf_gs;
    uint16_t tf_padding0;
    uint16_t tf_fs;
    uint16_t tf_padding1;
    uint16_t tf_es;
    uint16_t tf_padding2;
    uint16_t tf_ds;
    uint16_t tf_padding3;
```

```
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding4;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding5;
} __attribute__((packed));

void idt_init(void);
void print_trapframe(struct trapframe *tf);
void print_regs(struct pushregs *regs);
bool trap_in_kernel(struct trapframe *tf);

#endif /* !__KERN_TRAP_TRAP_H__ */

[ucore/lab8_result/kern/trap/trap.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <mmu.h>
#include <memlayout.h>
#include <clock.h>
#include <trap.h>
#include <x86.h>
#include <stdio.h>
#include <assert.h>
#include <console.h>
#include <vmm.h>
#include <swap.h>
#include <kdebug.h>
#include <unistd.h>
#include <syscall.h>
#include <error.h>
#include <sched.h>
#include <sync.h>
#include <proc.h>

#define TICK_NUM 100

static void print_ticks() {
    cprintf("%d ticks\n",TICK_NUM);
#ifdef DEBUG_GRADE
    cprintf("End of Test.\n");
    panic("EOT: kernel seems ok.");
#endif
}

/* *
 * Interrupt descriptor table:
 *
 * Must be built at run time because shifted function addresses can't
 * be represented in relocation records.
 * */
static struct gatedesc idt[256] = {{0}};

static struct pseudodesc idt_pd = {
    sizeof(idt) - 1, (uintptr_t)idt
};

/* idt_init - initialize IDT to each of the entry points in kern/trap/vectors.S */
void
idt_init(void) {
```

```
     /* LAB1 YOUR CODE : STEP 2 */
     /* (1) Where are the entry addrs of each Interrupt Service Routine (ISR)?
      *     All ISR's entry addrs are stored in __vectors. where is uintptr_t __vectors[] ?
      *     __vectors[] is in kern/trap/vector.S which is produced by tools/vector.c
      *     (try "make" command in lab1, then you will find vector.S in kern/trap DIR)
      *     You can use  "extern uintptr_t __vectors[];" to define this extern variable which
will be used later.
      * (2) Now you should setup the entries of ISR in Interrupt Description Table (IDT).
      *     Can you see idt[256] in this file? Yes, it's IDT! you can use SETGATE macro to set
up each item of IDT
      * (3) After setup the contents of IDT, you will let CPU know where is the IDT by using '
lidt' instruction.
      *     You don't know the meaning of this instruction? just google it! and check the libs
/x86.h to know more.
      *     Notice: the argument of lidt is idt_pd. try to find it!
      */
     /* LAB5 YOUR CODE */
     //you should update your lab1 code (just add ONE or TWO lines of code), let user app to u
se syscall to get the service of ucore
     //so you should setup the syscall interrupt gate in here
    extern uintptr_t __vectors[];
    int i;
    for (i = 0; i < sizeof(idt) / sizeof(struct gatedesc); i ++) {
        SETGATE(idt[i], 0, GD_KTEXT, __vectors[i], DPL_KERNEL);
    }
    SETGATE(idt[T_SYSCALL], 1, GD_KTEXT, __vectors[T_SYSCALL], DPL_USER);
    lidt(&idt_pd);
}

static const char *
trapname(int trapno) {
    static const char * const excnames[] = {
        "Divide error",
        "Debug",
        "Non-Maskable Interrupt",
        "Breakpoint",
        "Overflow",
        "BOUND Range Exceeded",
        "Invalid Opcode",
        "Device Not Available",
        "Double Fault",
        "Coprocessor Segment Overrun",
        "Invalid TSS",
        "Segment Not Present",
        "Stack Fault",
        "General Protection",
        "Page Fault",
        "(unknown trap)",
        "x87 FPU Floating-Point Error",
        "Alignment Check",
        "Machine-Check",
        "SIMD Floating-Point Exception"
    };

    if (trapno < sizeof(excnames)/sizeof(const char * const)) {
        return excnames[trapno];
    }
    if (trapno >= IRQ_OFFSET && trapno < IRQ_OFFSET + 16) {
        return "Hardware Interrupt";
    }
    return "(unknown trap)";
}

/* trap_in_kernel - test if trap happened in kernel */
bool
```

```
trap_in_kernel(struct trapframe *tf) {
    return (tf->tf_cs == (uint16_t)KERNEL_CS);
}

static const char *IA32flags[] = {
    "CF", NULL, "PF", NULL, "AF", NULL, "ZF", "SF",
    "TF", "IF", "DF", "OF", NULL, NULL, "NT", NULL,
    "RF", "VM", "AC", "VIF", "VIP", "ID", NULL, NULL,
};

void
print_trapframe(struct trapframe *tf) {
    cprintf("trapframe at %p\n", tf);
    print_regs(&tf->tf_regs);
    cprintf("  ds   0x----%04x\n", tf->tf_ds);
    cprintf("  es   0x----%04x\n", tf->tf_es);
    cprintf("  fs   0x----%04x\n", tf->tf_fs);
    cprintf("  gs   0x----%04x\n", tf->tf_gs);
    cprintf("  trap 0x%08x %s\n", tf->tf_trapno, trapname(tf->tf_trapno));
    cprintf("  err  0x%08x\n", tf->tf_err);
    cprintf("  eip  0x%08x\n", tf->tf_eip);
    cprintf("  cs   0x----%04x\n", tf->tf_cs);
    cprintf("  flag 0x%08x ", tf->tf_eflags);

    int i, j;
    for (i = 0, j = 1; i < sizeof(IA32flags) / sizeof(IA32flags[0]); i ++, j <<= 1) {
        if ((tf->tf_eflags & j) && IA32flags[i] != NULL) {
            cprintf("%s,", IA32flags[i]);
        }
    }
    cprintf("IOPL=%d\n", (tf->tf_eflags & FL_IOPL_MASK) >> 12);

    if (!trap_in_kernel(tf)) {
        cprintf("  esp  0x%08x\n", tf->tf_esp);
        cprintf("  ss   0x----%04x\n", tf->tf_ss);
    }
}

void
print_regs(struct pushregs *regs) {
    cprintf("  edi  0x%08x\n", regs->reg_edi);
    cprintf("  esi  0x%08x\n", regs->reg_esi);
    cprintf("  ebp  0x%08x\n", regs->reg_ebp);
    cprintf("  oesp 0x%08x\n", regs->reg_oesp);
    cprintf("  ebx  0x%08x\n", regs->reg_ebx);
    cprintf("  edx  0x%08x\n", regs->reg_edx);
    cprintf("  ecx  0x%08x\n", regs->reg_ecx);
    cprintf("  eax  0x%08x\n", regs->reg_eax);
}

static inline void
print_pgfault(struct trapframe *tf) {
    /* error_code:
     * bit 0 == 0 means no page found, 1 means protection fault
     * bit 1 == 0 means read, 1 means write
     * bit 2 == 0 means kernel, 1 means user
     * */
    cprintf("page fault at 0x%08x: %c/%c [%s].\n", rcr2(),
            (tf->tf_err & 4) ? 'U' : 'K',
            (tf->tf_err & 2) ? 'W' : 'R',
            (tf->tf_err & 1) ? "protection fault" : "no page found");
}

static int
pgfault_handler(struct trapframe *tf) {
```

```
    extern struct mm_struct *check_mm_struct;
    if(check_mm_struct !=NULL) { //used for test check_swap
            print_pgfault(tf);
        }
    struct mm_struct *mm;
    if (check_mm_struct != NULL) {
        assert(current == idleproc);
        mm = check_mm_struct;
    }
    else {
        if (current == NULL) {
            print_trapframe(tf);
            print_pgfault(tf);
            panic("unhandled page fault.\n");
        }
        mm = current->mm;
    }
    return do_pgfault(mm, tf->tf_err, rcr2());
}

static volatile int in_swap_tick_event = 0;
extern struct mm_struct *check_mm_struct;

static void
trap_dispatch(struct trapframe *tf) {
    char c;

    int ret=0;

    switch (tf->tf_trapno) {
    case T_PGFLT:  //page fault
        if ((ret = pgfault_handler(tf)) != 0) {
            print_trapframe(tf);
            if (current == NULL) {
                panic("handle pgfault failed. ret=%d\n", ret);
            }
            else {
                if (trap_in_kernel(tf)) {
                    panic("handle pgfault failed in kernel mode. ret=%d\n", ret);
                }
                cprintf("killed by kernel.\n");
                panic("handle user mode pgfault failed. ret=%d\n", ret);
                do_exit(-E_KILLED);
            }
        }
        break;
    case T_SYSCALL:
        syscall();
        break;
    case IRQ_OFFSET + IRQ_TIMER:
#if 0
    LAB3 : If some page replacement algorithm(such as CLOCK PRA) need tick to change the prior
ity of pages,
    then you can add code here.
#endif
        /* LAB1 YOUR CODE : STEP 3 */
        /* handle the timer interrupt */
        /* (1) After a timer interrupt, you should record this event using a global variable (
increase it), such as ticks in kern/driver/clock.c
         * (2) Every TICK_NUM cycle, you can print some info using a funciton, such as print_t
icks().
         * (3) Too Simple? Yes, I think so!
         */
        /* LAB5 YOUR CODE */
        /* you should upate you lab1 code (just add ONE or TWO lines of code):
```

```
         *    Every TICK_NUM cycle, you should set current process's current->need_resched = 1
         */
        /* LAB6 YOUR CODE */
        /* IMPORTANT FUNCTIONS:
             * run_timer_list
             *----------------------
             * you should update your lab5 code (just add ONE or TWO lines of code):
         *    Every tick, you should update the system time, iterate the timers, and trigger t
he timers which are end to call scheduler.
         *    You can use one funcitons to finish all these things.
         */
        ticks ++;
        assert(current != NULL);
        run_timer_list();
        break;
    case IRQ_OFFSET + IRQ_COM1:
        //c = cons_getc();
        //cprintf("serial [%03d] %c\n", c, c);
        //break;
    case IRQ_OFFSET + IRQ_KBD:
        c = cons_getc();
        //cprintf("kbd [%03d] %c\n", c, c);
        {
          extern void dev_stdin_write(char c);
          dev_stdin_write(c);
        }
        break;
    //LAB1 CHALLENGE 1 : YOUR CODE you should modify below codes.
    case T_SWITCH_TOU:
    case T_SWITCH_TOK:
        panic("T_SWITCH_** ??\n");
        break;
    case IRQ_OFFSET + IRQ_IDE1:
    case IRQ_OFFSET + IRQ_IDE2:
        /* do nothing */
        break;
    default:
        print_trapframe(tf);
        if (current != NULL) {
            cprintf("unhandled trap.\n");
            do_exit(-E_KILLED);
        }
        // in kernel, it must be a mistake
        panic("unexpected trap in kernel.\n");

    }
}

/* *
 * trap - handles or dispatches an exception/interrupt. if and when trap() returns,
 * the code in kern/trap/trapentry.S restores the old CPU state saved in the
 * trapframe and then uses the iret instruction to return from the exception.
 * */
void
trap(struct trapframe *tf) {
    // dispatch based on what type of trap occurred
    // used for previous projects
    if (current == NULL) {
        trap_dispatch(tf);
    }
    else {
        // keep a trapframe chain in stack
        struct trapframe *otf = current->tf;
        current->tf = tf;
```

```
        bool in_kernel = trap_in_kernel(tf);

        trap_dispatch(tf);

        current->tf = otf;
        if (!in_kernel) {
            if (current->flags & PF_EXITING) {
                do_exit(-E_KILLED);
            }
            if (current->need_resched) {
                schedule();
            }
        }
    }
}
```

[ucore/lab8_result/kern/trap/vectors.S] ++++++++++++++++++++++++++++++

```
# handler
.text
.globl __alltraps
.globl vector0
vector0:
  pushl $0
  pushl $0
  jmp __alltraps
.globl vector1
vector1:
  pushl $0
  pushl $1
  jmp __alltraps
.globl vector2
vector2:
  pushl $0
  pushl $2
  jmp __alltraps
.globl vector3
vector3:
  pushl $0
  pushl $3
  jmp __alltraps
.globl vector4
vector4:
  pushl $0
  pushl $4
  jmp __alltraps
.globl vector5
vector5:
  pushl $0
  pushl $5
  jmp __alltraps
.globl vector6
vector6:
  pushl $0
  pushl $6
  jmp __alltraps
.globl vector7
vector7:
  pushl $0
  pushl $7
  jmp __alltraps
.globl vector8
vector8:
  pushl $8
  jmp __alltraps
.globl vector9
```

```
vector9:
  pushl $9
  jmp __alltraps
.globl vector10
vector10:
  pushl $10
  jmp __alltraps
.globl vector11
vector11:
  pushl $11
  jmp __alltraps
.globl vector12
vector12:
  pushl $12
  jmp __alltraps
.globl vector13
vector13:
  pushl $13
  jmp __alltraps
.globl vector14
vector14:
  pushl $14
  jmp __alltraps
.globl vector15
vector15:
  pushl $0
  pushl $15
  jmp __alltraps
.globl vector16
vector16:
  pushl $0
  pushl $16
  jmp __alltraps
.globl vector17
vector17:
  pushl $17
  jmp __alltraps
.globl vector18
vector18:
  pushl $0
  pushl $18
  jmp __alltraps
.globl vector19
vector19:
  pushl $0
  pushl $19
  jmp __alltraps
.globl vector20
vector20:
  pushl $0
  pushl $20
  jmp __alltraps
.globl vector21
vector21:
  pushl $0
  pushl $21
  jmp __alltraps
.globl vector22
vector22:
  pushl $0
  pushl $22
  jmp __alltraps
.globl vector23
vector23:
  pushl $0
```

```
  pushl $23
  jmp __alltraps
.globl vector24
vector24:
  pushl $0
  pushl $24
  jmp __alltraps
.globl vector25
vector25:
  pushl $0
  pushl $25
  jmp __alltraps
.globl vector26
vector26:
  pushl $0
  pushl $26
  jmp __alltraps
.globl vector27
vector27:
  pushl $0
  pushl $27
  jmp __alltraps
.globl vector28
vector28:
  pushl $0
  pushl $28
  jmp __alltraps
.globl vector29
vector29:
  pushl $0
  pushl $29
  jmp __alltraps
.globl vector30
vector30:
  pushl $0
  pushl $30
  jmp __alltraps
.globl vector31
vector31:
  pushl $0
  pushl $31
  jmp __alltraps
.globl vector32
vector32:
  pushl $0
  pushl $32
  jmp __alltraps
.globl vector33
vector33:
  pushl $0
  pushl $33
  jmp __alltraps
.globl vector34
vector34:
  pushl $0
  pushl $34
  jmp __alltraps
.globl vector35
vector35:
  pushl $0
  pushl $35
  jmp __alltraps
.globl vector36
vector36:
  pushl $0
```

```
  pushl $36
  jmp __alltraps
.globl vector37
vector37:
  pushl $0
  pushl $37
  jmp __alltraps
.globl vector38
vector38:
  pushl $0
  pushl $38
  jmp __alltraps
.globl vector39
vector39:
  pushl $0
  pushl $39
  jmp __alltraps
.globl vector40
vector40:
  pushl $0
  pushl $40
  jmp __alltraps
.globl vector41
vector41:
  pushl $0
  pushl $41
  jmp __alltraps
.globl vector42
vector42:
  pushl $0
  pushl $42
  jmp __alltraps
.globl vector43
vector43:
  pushl $0
  pushl $43
  jmp __alltraps
.globl vector44
vector44:
  pushl $0
  pushl $44
  jmp __alltraps
.globl vector45
vector45:
  pushl $0
  pushl $45
  jmp __alltraps
.globl vector46
vector46:
  pushl $0
  pushl $46
  jmp __alltraps
.globl vector47
vector47:
  pushl $0
  pushl $47
  jmp __alltraps
.globl vector48
vector48:
  pushl $0
  pushl $48
  jmp __alltraps
.globl vector49
vector49:
  pushl $0
```

```
  pushl $49
  jmp __alltraps
.globl vector50
vector50:
  pushl $0
  pushl $50
  jmp __alltraps
.globl vector51
vector51:
  pushl $0
  pushl $51
  jmp __alltraps
.globl vector52
vector52:
  pushl $0
  pushl $52
  jmp __alltraps
.globl vector53
vector53:
  pushl $0
  pushl $53
  jmp __alltraps
.globl vector54
vector54:
  pushl $0
  pushl $54
  jmp __alltraps
.globl vector55
vector55:
  pushl $0
  pushl $55
  jmp __alltraps
.globl vector56
vector56:
  pushl $0
  pushl $56
  jmp __alltraps
.globl vector57
vector57:
  pushl $0
  pushl $57
  jmp __alltraps
.globl vector58
vector58:
  pushl $0
  pushl $58
  jmp __alltraps
.globl vector59
vector59:
  pushl $0
  pushl $59
  jmp __alltraps
.globl vector60
vector60:
  pushl $0
  pushl $60
  jmp __alltraps
.globl vector61
vector61:
  pushl $0
  pushl $61
  jmp __alltraps
.globl vector62
vector62:
  pushl $0
```

```
  pushl $62
  jmp __alltraps
.globl vector63
vector63:
  pushl $0
  pushl $63
  jmp __alltraps
.globl vector64
vector64:
  pushl $0
  pushl $64
  jmp __alltraps
.globl vector65
vector65:
  pushl $0
  pushl $65
  jmp __alltraps
.globl vector66
vector66:
  pushl $0
  pushl $66
  jmp __alltraps
.globl vector67
vector67:
  pushl $0
  pushl $67
  jmp __alltraps
.globl vector68
vector68:
  pushl $0
  pushl $68
  jmp __alltraps
.globl vector69
vector69:
  pushl $0
  pushl $69
  jmp __alltraps
.globl vector70
vector70:
  pushl $0
  pushl $70
  jmp __alltraps
.globl vector71
vector71:
  pushl $0
  pushl $71
  jmp __alltraps
.globl vector72
vector72:
  pushl $0
  pushl $72
  jmp __alltraps
.globl vector73
vector73:
  pushl $0
  pushl $73
  jmp __alltraps
.globl vector74
vector74:
  pushl $0
  pushl $74
  jmp __alltraps
.globl vector75
vector75:
  pushl $0
```

```
  pushl $75
  jmp __alltraps
.globl vector76
vector76:
  pushl $0
  pushl $76
  jmp __alltraps
.globl vector77
vector77:
  pushl $0
  pushl $77
  jmp __alltraps
.globl vector78
vector78:
  pushl $0
  pushl $78
  jmp __alltraps
.globl vector79
vector79:
  pushl $0
  pushl $79
  jmp __alltraps
.globl vector80
vector80:
  pushl $0
  pushl $80
  jmp __alltraps
.globl vector81
vector81:
  pushl $0
  pushl $81
  jmp __alltraps
.globl vector82
vector82:
  pushl $0
  pushl $82
  jmp __alltraps
.globl vector83
vector83:
  pushl $0
  pushl $83
  jmp __alltraps
.globl vector84
vector84:
  pushl $0
  pushl $84
  jmp __alltraps
.globl vector85
vector85:
  pushl $0
  pushl $85
  jmp __alltraps
.globl vector86
vector86:
  pushl $0
  pushl $86
  jmp __alltraps
.globl vector87
vector87:
  pushl $0
  pushl $87
  jmp __alltraps
.globl vector88
vector88:
  pushl $0
```

```
  pushl $88
  jmp __alltraps
.globl vector89
vector89:
  pushl $0
  pushl $89
  jmp __alltraps
.globl vector90
vector90:
  pushl $0
  pushl $90
  jmp __alltraps
.globl vector91
vector91:
  pushl $0
  pushl $91
  jmp __alltraps
.globl vector92
vector92:
  pushl $0
  pushl $92
  jmp __alltraps
.globl vector93
vector93:
  pushl $0
  pushl $93
  jmp __alltraps
.globl vector94
vector94:
  pushl $0
  pushl $94
  jmp __alltraps
.globl vector95
vector95:
  pushl $0
  pushl $95
  jmp __alltraps
.globl vector96
vector96:
  pushl $0
  pushl $96
  jmp __alltraps
.globl vector97
vector97:
  pushl $0
  pushl $97
  jmp __alltraps
.globl vector98
vector98:
  pushl $0
  pushl $98
  jmp __alltraps
.globl vector99
vector99:
  pushl $0
  pushl $99
  jmp __alltraps
.globl vector100
vector100:
  pushl $0
  pushl $100
  jmp __alltraps
.globl vector101
vector101:
  pushl $0
```

```
  pushl $101
  jmp __alltraps
.globl vector102
vector102:
  pushl $0
  pushl $102
  jmp __alltraps
.globl vector103
vector103:
  pushl $0
  pushl $103
  jmp __alltraps
.globl vector104
vector104:
  pushl $0
  pushl $104
  jmp __alltraps
.globl vector105
vector105:
  pushl $0
  pushl $105
  jmp __alltraps
.globl vector106
vector106:
  pushl $0
  pushl $106
  jmp __alltraps
.globl vector107
vector107:
  pushl $0
  pushl $107
  jmp __alltraps
.globl vector108
vector108:
  pushl $0
  pushl $108
  jmp __alltraps
.globl vector109
vector109:
  pushl $0
  pushl $109
  jmp __alltraps
.globl vector110
vector110:
  pushl $0
  pushl $110
  jmp __alltraps
.globl vector111
vector111:
  pushl $0
  pushl $111
  jmp __alltraps
.globl vector112
vector112:
  pushl $0
  pushl $112
  jmp __alltraps
.globl vector113
vector113:
  pushl $0
  pushl $113
  jmp __alltraps
.globl vector114
vector114:
  pushl $0
```

```
  pushl $114
  jmp __alltraps
.globl vector115
vector115:
  pushl $0
  pushl $115
  jmp __alltraps
.globl vector116
vector116:
  pushl $0
  pushl $116
  jmp __alltraps
.globl vector117
vector117:
  pushl $0
  pushl $117
  jmp __alltraps
.globl vector118
vector118:
  pushl $0
  pushl $118
  jmp __alltraps
.globl vector119
vector119:
  pushl $0
  pushl $119
  jmp __alltraps
.globl vector120
vector120:
  pushl $0
  pushl $120
  jmp __alltraps
.globl vector121
vector121:
  pushl $0
  pushl $121
  jmp __alltraps
.globl vector122
vector122:
  pushl $0
  pushl $122
  jmp __alltraps
.globl vector123
vector123:
  pushl $0
  pushl $123
  jmp __alltraps
.globl vector124
vector124:
  pushl $0
  pushl $124
  jmp __alltraps
.globl vector125
vector125:
  pushl $0
  pushl $125
  jmp __alltraps
.globl vector126
vector126:
  pushl $0
  pushl $126
  jmp __alltraps
.globl vector127
vector127:
  pushl $0
```

```
  pushl $127
  jmp __alltraps
.globl vector128
vector128:
  pushl $0
  pushl $128
  jmp __alltraps
.globl vector129
vector129:
  pushl $0
  pushl $129
  jmp __alltraps
.globl vector130
vector130:
  pushl $0
  pushl $130
  jmp __alltraps
.globl vector131
vector131:
  pushl $0
  pushl $131
  jmp __alltraps
.globl vector132
vector132:
  pushl $0
  pushl $132
  jmp __alltraps
.globl vector133
vector133:
  pushl $0
  pushl $133
  jmp __alltraps
.globl vector134
vector134:
  pushl $0
  pushl $134
  jmp __alltraps
.globl vector135
vector135:
  pushl $0
  pushl $135
  jmp __alltraps
.globl vector136
vector136:
  pushl $0
  pushl $136
  jmp __alltraps
.globl vector137
vector137:
  pushl $0
  pushl $137
  jmp __alltraps
.globl vector138
vector138:
  pushl $0
  pushl $138
  jmp __alltraps
.globl vector139
vector139:
  pushl $0
  pushl $139
  jmp __alltraps
.globl vector140
vector140:
  pushl $0
```

```
  pushl $140
  jmp __alltraps
.globl vector141
vector141:
  pushl $0
  pushl $141
  jmp __alltraps
.globl vector142
vector142:
  pushl $0
  pushl $142
  jmp __alltraps
.globl vector143
vector143:
  pushl $0
  pushl $143
  jmp __alltraps
.globl vector144
vector144:
  pushl $0
  pushl $144
  jmp __alltraps
.globl vector145
vector145:
  pushl $0
  pushl $145
  jmp __alltraps
.globl vector146
vector146:
  pushl $0
  pushl $146
  jmp __alltraps
.globl vector147
vector147:
  pushl $0
  pushl $147
  jmp __alltraps
.globl vector148
vector148:
  pushl $0
  pushl $148
  jmp __alltraps
.globl vector149
vector149:
  pushl $0
  pushl $149
  jmp __alltraps
.globl vector150
vector150:
  pushl $0
  pushl $150
  jmp __alltraps
.globl vector151
vector151:
  pushl $0
  pushl $151
  jmp __alltraps
.globl vector152
vector152:
  pushl $0
  pushl $152
  jmp __alltraps
.globl vector153
vector153:
  pushl $0
```

```
  pushl $153
  jmp __alltraps
.globl vector154
vector154:
  pushl $0
  pushl $154
  jmp __alltraps
.globl vector155
vector155:
  pushl $0
  pushl $155
  jmp __alltraps
.globl vector156
vector156:
  pushl $0
  pushl $156
  jmp __alltraps
.globl vector157
vector157:
  pushl $0
  pushl $157
  jmp __alltraps
.globl vector158
vector158:
  pushl $0
  pushl $158
  jmp __alltraps
.globl vector159
vector159:
  pushl $0
  pushl $159
  jmp __alltraps
.globl vector160
vector160:
  pushl $0
  pushl $160
  jmp __alltraps
.globl vector161
vector161:
  pushl $0
  pushl $161
  jmp __alltraps
.globl vector162
vector162:
  pushl $0
  pushl $162
  jmp __alltraps
.globl vector163
vector163:
  pushl $0
  pushl $163
  jmp __alltraps
.globl vector164
vector164:
  pushl $0
  pushl $164
  jmp __alltraps
.globl vector165
vector165:
  pushl $0
  pushl $165
  jmp __alltraps
.globl vector166
vector166:
  pushl $0
```

```
  pushl $166
  jmp __alltraps
.globl vector167
vector167:
  pushl $0
  pushl $167
  jmp __alltraps
.globl vector168
vector168:
  pushl $0
  pushl $168
  jmp __alltraps
.globl vector169
vector169:
  pushl $0
  pushl $169
  jmp __alltraps
.globl vector170
vector170:
  pushl $0
  pushl $170
  jmp __alltraps
.globl vector171
vector171:
  pushl $0
  pushl $171
  jmp __alltraps
.globl vector172
vector172:
  pushl $0
  pushl $172
  jmp __alltraps
.globl vector173
vector173:
  pushl $0
  pushl $173
  jmp __alltraps
.globl vector174
vector174:
  pushl $0
  pushl $174
  jmp __alltraps
.globl vector175
vector175:
  pushl $0
  pushl $175
  jmp __alltraps
.globl vector176
vector176:
  pushl $0
  pushl $176
  jmp __alltraps
.globl vector177
vector177:
  pushl $0
  pushl $177
  jmp __alltraps
.globl vector178
vector178:
  pushl $0
  pushl $178
  jmp __alltraps
.globl vector179
vector179:
  pushl $0
```

```
  pushl $179
  jmp __alltraps
.globl vector180
vector180:
  pushl $0
  pushl $180
  jmp __alltraps
.globl vector181
vector181:
  pushl $0
  pushl $181
  jmp __alltraps
.globl vector182
vector182:
  pushl $0
  pushl $182
  jmp __alltraps
.globl vector183
vector183:
  pushl $0
  pushl $183
  jmp __alltraps
.globl vector184
vector184:
  pushl $0
  pushl $184
  jmp __alltraps
.globl vector185
vector185:
  pushl $0
  pushl $185
  jmp __alltraps
.globl vector186
vector186:
  pushl $0
  pushl $186
  jmp __alltraps
.globl vector187
vector187:
  pushl $0
  pushl $187
  jmp __alltraps
.globl vector188
vector188:
  pushl $0
  pushl $188
  jmp __alltraps
.globl vector189
vector189:
  pushl $0
  pushl $189
  jmp __alltraps
.globl vector190
vector190:
  pushl $0
  pushl $190
  jmp __alltraps
.globl vector191
vector191:
  pushl $0
  pushl $191
  jmp __alltraps
.globl vector192
vector192:
  pushl $0
```

```
  pushl $192
  jmp __alltraps
.globl vector193
vector193:
  pushl $0
  pushl $193
  jmp __alltraps
.globl vector194
vector194:
  pushl $0
  pushl $194
  jmp __alltraps
.globl vector195
vector195:
  pushl $0
  pushl $195
  jmp __alltraps
.globl vector196
vector196:
  pushl $0
  pushl $196
  jmp __alltraps
.globl vector197
vector197:
  pushl $0
  pushl $197
  jmp __alltraps
.globl vector198
vector198:
  pushl $0
  pushl $198
  jmp __alltraps
.globl vector199
vector199:
  pushl $0
  pushl $199
  jmp __alltraps
.globl vector200
vector200:
  pushl $0
  pushl $200
  jmp __alltraps
.globl vector201
vector201:
  pushl $0
  pushl $201
  jmp __alltraps
.globl vector202
vector202:
  pushl $0
  pushl $202
  jmp __alltraps
.globl vector203
vector203:
  pushl $0
  pushl $203
  jmp __alltraps
.globl vector204
vector204:
  pushl $0
  pushl $204
  jmp __alltraps
.globl vector205
vector205:
  pushl $0
```

```
  pushl $205
  jmp __alltraps
.globl vector206
vector206:
  pushl $0
  pushl $206
  jmp __alltraps
.globl vector207
vector207:
  pushl $0
  pushl $207
  jmp __alltraps
.globl vector208
vector208:
  pushl $0
  pushl $208
  jmp __alltraps
.globl vector209
vector209:
  pushl $0
  pushl $209
  jmp __alltraps
.globl vector210
vector210:
  pushl $0
  pushl $210
  jmp __alltraps
.globl vector211
vector211:
  pushl $0
  pushl $211
  jmp __alltraps
.globl vector212
vector212:
  pushl $0
  pushl $212
  jmp __alltraps
.globl vector213
vector213:
  pushl $0
  pushl $213
  jmp __alltraps
.globl vector214
vector214:
  pushl $0
  pushl $214
  jmp __alltraps
.globl vector215
vector215:
  pushl $0
  pushl $215
  jmp __alltraps
.globl vector216
vector216:
  pushl $0
  pushl $216
  jmp __alltraps
.globl vector217
vector217:
  pushl $0
  pushl $217
  jmp __alltraps
.globl vector218
vector218:
  pushl $0
```

```
  pushl $218
  jmp __alltraps
.globl vector219
vector219:
  pushl $0
  pushl $219
  jmp __alltraps
.globl vector220
vector220:
  pushl $0
  pushl $220
  jmp __alltraps
.globl vector221
vector221:
  pushl $0
  pushl $221
  jmp __alltraps
.globl vector222
vector222:
  pushl $0
  pushl $222
  jmp __alltraps
.globl vector223
vector223:
  pushl $0
  pushl $223
  jmp __alltraps
.globl vector224
vector224:
  pushl $0
  pushl $224
  jmp __alltraps
.globl vector225
vector225:
  pushl $0
  pushl $225
  jmp __alltraps
.globl vector226
vector226:
  pushl $0
  pushl $226
  jmp __alltraps
.globl vector227
vector227:
  pushl $0
  pushl $227
  jmp __alltraps
.globl vector228
vector228:
  pushl $0
  pushl $228
  jmp __alltraps
.globl vector229
vector229:
  pushl $0
  pushl $229
  jmp __alltraps
.globl vector230
vector230:
  pushl $0
  pushl $230
  jmp __alltraps
.globl vector231
vector231:
  pushl $0
```

```
  pushl $231
  jmp __alltraps
.globl vector232
vector232:
  pushl $0
  pushl $232
  jmp __alltraps
.globl vector233
vector233:
  pushl $0
  pushl $233
  jmp __alltraps
.globl vector234
vector234:
  pushl $0
  pushl $234
  jmp __alltraps
.globl vector235
vector235:
  pushl $0
  pushl $235
  jmp __alltraps
.globl vector236
vector236:
  pushl $0
  pushl $236
  jmp __alltraps
.globl vector237
vector237:
  pushl $0
  pushl $237
  jmp __alltraps
.globl vector238
vector238:
  pushl $0
  pushl $238
  jmp __alltraps
.globl vector239
vector239:
  pushl $0
  pushl $239
  jmp __alltraps
.globl vector240
vector240:
  pushl $0
  pushl $240
  jmp __alltraps
.globl vector241
vector241:
  pushl $0
  pushl $241
  jmp __alltraps
.globl vector242
vector242:
  pushl $0
  pushl $242
  jmp __alltraps
.globl vector243
vector243:
  pushl $0
  pushl $243
  jmp __alltraps
.globl vector244
vector244:
  pushl $0
```

```
  pushl $244
  jmp __alltraps
.globl vector245
vector245:
  pushl $0
  pushl $245
  jmp __alltraps
.globl vector246
vector246:
  pushl $0
  pushl $246
  jmp __alltraps
.globl vector247
vector247:
  pushl $0
  pushl $247
  jmp __alltraps
.globl vector248
vector248:
  pushl $0
  pushl $248
  jmp __alltraps
.globl vector249
vector249:
  pushl $0
  pushl $249
  jmp __alltraps
.globl vector250
vector250:
  pushl $0
  pushl $250
  jmp __alltraps
.globl vector251
vector251:
  pushl $0
  pushl $251
  jmp __alltraps
.globl vector252
vector252:
  pushl $0
  pushl $252
  jmp __alltraps
.globl vector253
vector253:
  pushl $0
  pushl $253
  jmp __alltraps
.globl vector254
vector254:
  pushl $0
  pushl $254
  jmp __alltraps
.globl vector255
vector255:
  pushl $0
  pushl $255
  jmp __alltraps

# vector table
.data
.globl __vectors
__vectors:
  .long vector0
  .long vector1
  .long vector2
```

```
.long vector3
.long vector4
.long vector5
.long vector6
.long vector7
.long vector8
.long vector9
.long vector10
.long vector11
.long vector12
.long vector13
.long vector14
.long vector15
.long vector16
.long vector17
.long vector18
.long vector19
.long vector20
.long vector21
.long vector22
.long vector23
.long vector24
.long vector25
.long vector26
.long vector27
.long vector28
.long vector29
.long vector30
.long vector31
.long vector32
.long vector33
.long vector34
.long vector35
.long vector36
.long vector37
.long vector38
.long vector39
.long vector40
.long vector41
.long vector42
.long vector43
.long vector44
.long vector45
.long vector46
.long vector47
.long vector48
.long vector49
.long vector50
.long vector51
.long vector52
.long vector53
.long vector54
.long vector55
.long vector56
.long vector57
.long vector58
.long vector59
.long vector60
.long vector61
.long vector62
.long vector63
.long vector64
.long vector65
.long vector66
.long vector67
```

```
.long vector68
.long vector69
.long vector70
.long vector71
.long vector72
.long vector73
.long vector74
.long vector75
.long vector76
.long vector77
.long vector78
.long vector79
.long vector80
.long vector81
.long vector82
.long vector83
.long vector84
.long vector85
.long vector86
.long vector87
.long vector88
.long vector89
.long vector90
.long vector91
.long vector92
.long vector93
.long vector94
.long vector95
.long vector96
.long vector97
.long vector98
.long vector99
.long vector100
.long vector101
.long vector102
.long vector103
.long vector104
.long vector105
.long vector106
.long vector107
.long vector108
.long vector109
.long vector110
.long vector111
.long vector112
.long vector113
.long vector114
.long vector115
.long vector116
.long vector117
.long vector118
.long vector119
.long vector120
.long vector121
.long vector122
.long vector123
.long vector124
.long vector125
.long vector126
.long vector127
.long vector128
.long vector129
.long vector130
.long vector131
.long vector132
```

```
.long vector133
.long vector134
.long vector135
.long vector136
.long vector137
.long vector138
.long vector139
.long vector140
.long vector141
.long vector142
.long vector143
.long vector144
.long vector145
.long vector146
.long vector147
.long vector148
.long vector149
.long vector150
.long vector151
.long vector152
.long vector153
.long vector154
.long vector155
.long vector156
.long vector157
.long vector158
.long vector159
.long vector160
.long vector161
.long vector162
.long vector163
.long vector164
.long vector165
.long vector166
.long vector167
.long vector168
.long vector169
.long vector170
.long vector171
.long vector172
.long vector173
.long vector174
.long vector175
.long vector176
.long vector177
.long vector178
.long vector179
.long vector180
.long vector181
.long vector182
.long vector183
.long vector184
.long vector185
.long vector186
.long vector187
.long vector188
.long vector189
.long vector190
.long vector191
.long vector192
.long vector193
.long vector194
.long vector195
.long vector196
.long vector197
```

```
  .long vector198
  .long vector199
  .long vector200
  .long vector201
  .long vector202
  .long vector203
  .long vector204
  .long vector205
  .long vector206
  .long vector207
  .long vector208
  .long vector209
  .long vector210
  .long vector211
  .long vector212
  .long vector213
  .long vector214
  .long vector215
  .long vector216
  .long vector217
  .long vector218
  .long vector219
  .long vector220
  .long vector221
  .long vector222
  .long vector223
  .long vector224
  .long vector225
  .long vector226
  .long vector227
  .long vector228
  .long vector229
  .long vector230
  .long vector231
  .long vector232
  .long vector233
  .long vector234
  .long vector235
  .long vector236
  .long vector237
  .long vector238
  .long vector239
  .long vector240
  .long vector241
  .long vector242
  .long vector243
  .long vector244
  .long vector245
  .long vector246
  .long vector247
  .long vector248
  .long vector249
  .long vector250
  .long vector251
  .long vector252
  .long vector253
  .long vector254
  .long vector255
[ucore/lab8_result/kern/debug/panic.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <stdio.h>
#include <intr.h>
#include <kmonitor.h>

static bool is_panic = 0;
```

```
/* *
 * __panic - __panic is called on unresolvable fatal errors. it prints
 * "panic: 'message'", and then enters the kernel monitor.
 * */
void
__panic(const char *file, int line, const char *fmt, ...) {
    if (is_panic) {
        goto panic_dead;
    }
    is_panic = 1;

    // print the 'message'
    va_list ap;
    va_start(ap, fmt);
    cprintf("kernel panic at %s:%d:\n    ", file, line);
    vcprintf(fmt, ap);
    cprintf("\n");

    cprintf("stack trackback:\n");
    print_stackframe();

    va_end(ap);

panic_dead:
    intr_disable();
    while (1) {
        kmonitor(NULL);
    }
}

/* __warn - like panic, but don't */
void
__warn(const char *file, int line, const char *fmt, ...) {
    va_list ap;
    va_start(ap, fmt);
    cprintf("kernel warning at %s:%d:\n    ", file, line);
    vcprintf(fmt, ap);
    cprintf("\n");
    va_end(ap);
}

bool
is_kernel_panic(void) {
    return is_panic;
}
```

[ucore/lab8_result/kern/debug/kdebug.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DEBUG_KDEBUG_H__
#define __KERN_DEBUG_KDEBUG_H__

#include <defs.h>
#include <trap.h>

void print_kerninfo(void);
void print_stackframe(void);
void print_debuginfo(uintptr_t eip);

#endif /* !__KERN_DEBUG_KDEBUG_H__ */
```

[ucore/lab8_result/kern/debug/assert.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DEBUG_ASSERT_H__
#define __KERN_DEBUG_ASSERT_H__

#include <defs.h>
```

```
void __warn(const char *file, int line, const char *fmt, ...);
void __noreturn __panic(const char *file, int line, const char *fmt, ...);

#define warn(...)                                               \
    __warn(__FILE__, __LINE__, __VA_ARGS__)

#define panic(...)                                              \
    __panic(__FILE__, __LINE__, __VA_ARGS__)

#define assert(x)                                               \
    do {                                                        \
        if (!(x)) {                                             \
            panic("assertion failed: %s", #x);                  \
        }                                                       \
    } while (0)

// static_assert(x) will generate a compile-time error if 'x' is false.
#define static_assert(x)                                        \
    switch (x) { case 0: case (x): ; }

#endif /* !__KERN_DEBUG_ASSERT_H__ */

[ucore/lab8_result/kern/debug/kdebug.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <x86.h>
#include <stab.h>
#include <stdio.h>
#include <string.h>
#include <memlayout.h>
#include <sync.h>
#include <vmm.h>
#include <proc.h>
#include <kdebug.h>
#include <kmonitor.h>
#include <assert.h>

#define STACKFRAME_DEPTH 20

extern const struct stab __STAB_BEGIN__[];  // beginning of stabs table
extern const struct stab __STAB_END__[];    // end of stabs table
extern const char __STABSTR_BEGIN__[];      // beginning of string table
extern const char __STABSTR_END__[];        // end of string table

/* debug information about a particular instruction pointer */
struct eipdebuginfo {
    const char *eip_file;                   // source code filename for eip
    int eip_line;                           // source code line number for eip
    const char *eip_fn_name;                // name of function containing eip
    int eip_fn_namelen;                     // length of function's name
    uintptr_t eip_fn_addr;                  // start address of function
    int eip_fn_narg;                        // number of function arguments
};

/* user STABS data structure  */
struct userstabdata {
    const struct stab *stabs;
    const struct stab *stab_end;
    const char *stabstr;
    const char *stabstr_end;
};

/* *
 * stab_binsearch - according to the input, the initial value of
 * range [*@region_left, *@region_right], find a single stab entry
```

```
 * that includes the address @addr and matches the type @type,
 * and then save its boundary to the locations that pointed
 * by @region_left and @region_right.
 *
 * Some stab types are arranged in increasing order by instruction address.
 * For example, N_FUN stabs (stab entries with n_type == N_FUN), which
 * mark functions, and N_SO stabs, which mark source files.
 *
 * Given an instruction address, this function finds the single stab entry
 * of type @type that contains that address.
 *
 * The search takes place within the range [*@region_left, *@region_right].
 * Thus, to search an entire set of N stabs, you might do:
 *
 *      left = 0;
 *      right = N - 1;     (rightmost stab)
 *      stab_binsearch(stabs, &left, &right, type, addr);
 *
 * The search modifies *region_left and *region_right to bracket the @addr.
 * *@region_left points to the matching stab that contains @addr,
 * and *@region_right points just before the next stab.
 * If *@region_left > *region_right, then @addr is not contained in any
 * matching stab.
 *
 * For example, given these N_SO stabs:
 *      Index  Type   Address
 *      0      SO     f0100000
 *      13     SO     f0100040
 *      117    SO     f0100176
 *      118    SO     f0100178
 *      555    SO     f0100652
 *      556    SO     f0100654
 *      657    SO     f0100849
 * this code:
 *      left = 0, right = 657;
 *      stab_binsearch(stabs, &left, &right, N_SO, 0xf0100184);
 * will exit setting left = 118, right = 554.
 */
static void
stab_binsearch(const struct stab *stabs, int *region_left, int *region_right,
           int type, uintptr_t addr) {
    int l = *region_left, r = *region_right, any_matches = 0;

    while (l <= r) {
        int true_m = (l + r) / 2, m = true_m;

        // search for earliest stab with right type
        while (m >= l && stabs[m].n_type != type) {
            m --;
        }
        if (m < l) {    // no match in [l, m]
            l = true_m + 1;
            continue;
        }

        // actual binary search
        any_matches = 1;
        if (stabs[m].n_value < addr) {
            *region_left = m;
            l = true_m + 1;
        } else if (stabs[m].n_value > addr) {
            *region_right = m - 1;
            r = m - 1;
        } else {
            // exact match for 'addr', but continue loop to find
```

```
            // *region_right
            *region_left = m;
            l = m;
            addr ++;
        }
    }

    if (!any_matches) {
        *region_right = *region_left - 1;
    }
    else {
        // find rightmost region containing 'addr'
        l = *region_right;
        for (; l > *region_left && stabs[l].n_type != type; l --)
            /* do nothing */;
        *region_left = l;
    }
}

/* *
 * debuginfo_eip - Fill in the @info structure with information about
 * the specified instruction address, @addr.  Returns 0 if information
 * was found, and negative if not.  But even if it returns negative it
 * has stored some information into '*info'.
 * */
int
debuginfo_eip(uintptr_t addr, struct eipdebuginfo *info) {
    const struct stab *stabs, *stab_end;
    const char *stabstr, *stabstr_end;

    info->eip_file = "<unknown>";
    info->eip_line = 0;
    info->eip_fn_name = "<unknown>";
    info->eip_fn_namelen = 9;
    info->eip_fn_addr = addr;
    info->eip_fn_narg = 0;

    // find the relevant set of stabs
    if (addr >= KERNBASE) {
        stabs = __STAB_BEGIN__;
        stab_end = __STAB_END__;
        stabstr = __STABSTR_BEGIN__;
        stabstr_end = __STABSTR_END__;
    }
    else {
        // user-program linker script, tools/user.ld puts the information about the
        // program's stabs (included __STAB_BEGIN__, __STAB_END__, __STABSTR_BEGIN__,
        // and __STABSTR_END__) in a structure located at virtual address USTAB.
        const struct userstabdata *usd = (struct userstabdata *)USTAB;

        // make sure that debugger (current process) can access this memory
        struct mm_struct *mm;
        if (current == NULL || (mm = current->mm) == NULL) {
            return -1;
        }
        if (!user_mem_check(mm, (uintptr_t)usd, sizeof(struct userstabdata), 0)) {
            return -1;
        }

        stabs = usd->stabs;
        stab_end = usd->stab_end;
        stabstr = usd->stabstr;
        stabstr_end = usd->stabstr_end;

        // make sure the STABS and string table memory is valid
```

```
        if (!user_mem_check(mm, (uintptr_t)stabs, (uintptr_t)stab_end - (uintptr_t)stabs, 0))
{
            return -1;
        }
        if (!user_mem_check(mm, (uintptr_t)stabstr, stabstr_end - stabstr, 0)) {
            return -1;
        }
    }

    // String table validity checks
    if (stabstr_end <= stabstr || stabstr_end[-1] != 0) {
        return -1;
    }

    // Now we find the right stabs that define the function containing
    // 'eip'.  First, we find the basic source file containing 'eip'.
    // Then, we look in that source file for the function.  Then we look
    // for the line number.

    // Search the entire set of stabs for the source file (type N_SO).
    int lfile = 0, rfile = (stab_end - stabs) - 1;
    stab_binsearch(stabs, &lfile, &rfile, N_SO, addr);
    if (lfile == 0)
        return -1;

    // Search within that file's stabs for the function definition
    // (N_FUN).
    int lfun = lfile, rfun = rfile;
    int lline, rline;
    stab_binsearch(stabs, &lfun, &rfun, N_FUN, addr);

    if (lfun <= rfun) {
        // stabs[lfun] points to the function name
        // in the string table, but check bounds just in case.
        if (stabs[lfun].n_strx < stabstr_end - stabstr) {
            info->eip_fn_name = stabstr + stabs[lfun].n_strx;
        }
        info->eip_fn_addr = stabs[lfun].n_value;
        addr -= info->eip_fn_addr;
        // Search within the function definition for the line number.
        lline = lfun;
        rline = rfun;
    } else {
        // Couldn't find function stab!  Maybe we're in an assembly
        // file.  Search the whole file for the line number.
        info->eip_fn_addr = addr;
        lline = lfile;
        rline = rfile;
    }
    info->eip_fn_namelen = strfind(info->eip_fn_name, ':') - info->eip_fn_name;

    // Search within [lline, rline] for the line number stab.
    // If found, set info->eip_line to the right line number.
    // If not found, return -1.
    stab_binsearch(stabs, &lline, &rline, N_SLINE, addr);
    if (lline <= rline) {
        info->eip_line = stabs[rline].n_desc;
    } else {
        return -1;
    }

    // Search backwards from the line number for the relevant filename stab.
    // We can't just use the "lfile" stab because inlined functions
    // can interpolate code from a different file!
    // Such included source files use the N_SOL stab type.
```

```
    while (lline >= lfile
           && stabs[lline].n_type != N_SOL
           && (stabs[lline].n_type != N_SO || !stabs[lline].n_value)) {
        lline --;
    }
    if (lline >= lfile && stabs[lline].n_strx < stabstr_end - stabstr) {
        info->eip_file = stabstr + stabs[lline].n_strx;
    }

    // Set eip_fn_narg to the number of arguments taken by the function,
    // or 0 if there was no containing function.
    if (lfun < rfun) {
        for (lline = lfun + 1;
             lline < rfun && stabs[lline].n_type == N_PSYM;
             lline ++) {
            info->eip_fn_narg ++;
        }
    }
    return 0;
}

/* *
 * print_kerninfo - print the information about kernel, including the location
 * of kernel entry, the start addresses of data and text segements, the start
 * address of free memory and how many memory that kernel has used.
 * */
void
print_kerninfo(void) {
    extern char etext[], edata[], end[], kern_init[];
    cprintf("Special kernel symbols:\n");
    cprintf("  entry  0x%08x (phys)\n", kern_init);
    cprintf("  etext  0x%08x (phys)\n", etext);
    cprintf("  edata  0x%08x (phys)\n", edata);
    cprintf("  end    0x%08x (phys)\n", end);
    cprintf("Kernel executable memory footprint: %dKB\n", (end - kern_init + 1023)/1024);
}

/* *
 * print_debuginfo - read and print the stat information for the address @eip,
 * and info.eip_fn_addr should be the first address of the related function.
 * */
void
print_debuginfo(uintptr_t eip) {
    struct eipdebuginfo info;
    if (debuginfo_eip(eip, &info) != 0) {
        cprintf("    <unknow>: -- 0x%08x --\n", eip);
    }
    else {
        char fnname[256];
        int j;
        for (j = 0; j < info.eip_fn_namelen; j ++) {
            fnname[j] = info.eip_fn_name[j];
        }
        fnname[j] = '\0';
        cprintf("    %s:%d: %s+%d\n", info.eip_file, info.eip_line,
                fnname, eip - info.eip_fn_addr);
    }
}

static __noinline uint32_t
read_eip(void) {
    uint32_t eip;
    asm volatile("movl 4(%%ebp), %0" : "=r" (eip));
    return eip;
}
```

```
static inline uint32_t read_esp(void)
{
        uint32_t esp;
        asm volatile("movl %%esp, %0" : "=r" (esp));
        return esp;
}


/* *
 * print_stackframe - print a list of the saved eip values from the nested 'call'
 * instructions that led to the current point of execution
 *
 * The x86 stack pointer, namely esp, points to the lowest location on the stack
 * that is currently in use. Everything below that location in stack is free. Pushing
 * a value onto the stack will invole decreasing the stack pointer and then writing
 * the value to the place that stack pointer pointes to. And popping a value do the
 * opposite.
 *
 * The ebp (base pointer) register, in contrast, is associated with the stack
 * primarily by software convention. On entry to a C function, the function's
 * prologue code normally saves the previous function's base pointer by pushing
 * it onto the stack, and then copies the current esp value into ebp for the duration
 * of the function. If all the functions in a program obey this convention,
 * then at any given point during the program's execution, it is possible to trace
 * back through the stack by following the chain of saved ebp pointers and determining
 * exactly what nested sequence of function calls caused this particular point in the
 * program to be reached. This capability can be particularly useful, for example,
 * when a particular function causes an assert failure or panic because bad arguments
 * were passed to it, but you aren't sure who passed the bad arguments. A stack
 * backtrace lets you find the offending function.
 *
 * The inline function read_ebp() can tell us the value of current ebp. And the
 * non-inline function read_eip() is useful, it can read the value of current eip,
 * since while calling this function, read_eip() can read the caller's eip from
 * stack easily.
 *
 * In print_debuginfo(), the function debuginfo_eip() can get enough information about
 * calling-chain. Finally print_stackframe() will trace and print them for debugging.
 *
 * Note that, the length of ebp-chain is limited. In boot/bootasm.S, before jumping
 * to the kernel entry, the value of ebp has been set to zero, that's the boundary.
 * */
//void
//print_stackframe(void) {
//     /* LAB1 YOUR CODE : STEP 1 */
//     /* (1) call read_ebp() to get the value of ebp. the type is (uint32_t);
//      * (2) call read_eip() to get the value of eip. the type is (uint32_t);
//      * (3) from 0 .. STACKFRAME_DEPTH
//      *    (3.1) printf value of ebp, eip
//      *    (3.2) (uint32_t)calling arguments [0..4] = the contents in address (uint32_t)ebp
+2 [0..4]
//      *    (3.3) cprintf("\n");
//      *    (3.4) call print_debuginfo(eip-1) to print the C calling function name and line n
umber, etc.
//      *    (3.5) popup a calling stackframe
//      *           NOTICE: the calling funciton's return addr eip  = ss:[ebp+4]
//      *                   the calling funciton's ebp = ss:[ebp]
//      */
//    uint32_t t_ebp = read_ebp(), t_eip = read_eip();
//    t_ebp = read_ebp();
//    t_eip = read_eip();
//
//    int i, j;
//    for (i = 0; t_ebp != 0 && i < STACKFRAME_DEPTH; i ++) {
//        cprintf("t_ebp:0x%08x t_eip:0x%08x args:", t_ebp, t_eip);
```

```
//         uint32_t *args = (uint32_t *)t_ebp + 2;
//         for (j = 0; j < 4; j ++) {
//             cprintf("0x%08x ", args[j]);
//         }
//         cprintf("\n");
//         print_debuginfo(t_eip - 1);
//         t_eip = ((uint32_t *)t_ebp)[1];
//         t_ebp = ((uint32_t *)t_ebp)[0];
//     }
//}

void print_stack(uint32_t addr, uint32_t len)
{
        while(len--)
        {
                cprintf("addr:0x%08x ", addr);
                cprintf("val:0x%08x \n", *((uint32_t*)addr));
                addr += sizeof(addr);
        }
}

void print_stackframe(void)
{
        uint32_t t_ebp = read_ebp();
        uint32_t t_eip = read_eip();
        int i = 0, j = 0;
        for (i = 0; t_ebp != 0 && i < STACKFRAME_DEPTH; i++) {
                cprintf("t_ebp:0x%x t_eip:0x%x args:", t_ebp, t_eip);
                uint32_t* args = (uint32_t*)t_ebp + 2;
                for (j = 0; j < 4; j++) {
                        cprintf("0x%08x ", args[j]);
                }
                cprintf("\n");
                print_debuginfo(t_eip - 1);
                t_eip = ((uint32_t *)t_ebp)[1];
                t_ebp = ((uint32_t *)t_ebp)[0];
        }
}
```

[ucore/lab8_result/kern/debug/stab.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DEBUG_STAB_H__
#define __KERN_DEBUG_STAB_H__

#include <defs.h>

/* *
 * STABS debugging info
 *
 * The kernel debugger can understand some debugging information in
 * the STABS format.  For more information on this format, see
 * http://sources.redhat.com/gdb/onlinedocs/stabs_toc.html
 *
 * The constants below define some symbol types used by various debuggers
 * and compilers.  Kernel uses the N_SO, N_SOL, N_FUN, and N_SLINE types.
 * */

#define N_GSYM      0x20    // global symbol
#define N_FNAME     0x22    // F77 function name
#define N_FUN       0x24    // procedure name
#define N_STSYM     0x26    // data segment variable
#define N_LCSYM     0x28    // bss segment variable
#define N_MAIN      0x2a    // main function name
#define N_PC        0x30    // global Pascal symbol
#define N_RSYM      0x40    // register variable
```

```
#define N_SLINE     0x44    // text segment line number
#define N_DSLINE    0x46    // data segment line number
#define N_BSLINE    0x48    // bss segment line number
#define N_SSYM      0x60    // structure/union element
#define N_SO        0x64    // main source file name
#define N_LSYM      0x80    // stack variable
#define N_BINCL     0x82    // include file beginning
#define N_SOL       0x84    // included source file name
#define N_PSYM      0xa0    // parameter variable
#define N_EINCL     0xa2    // include file end
#define N_ENTRY     0xa4    // alternate entry point
#define N_LBRAC     0xc0    // left bracket
#define N_EXCL      0xc2    // deleted include file
#define N_RBRAC     0xe0    // right bracket
#define N_BCOMM     0xe2    // begin common
#define N_ECOMM     0xe4    // end common
#define N_ECOML     0xe8    // end common (local name)
#define N_LENG      0xfe    // length of preceding entry

/* Entries in the STABS table are formatted as follows. */
struct stab {
    uint32_t n_strx;        // index into string table of name
    uint8_t n_type;         // type of symbol
    uint8_t n_other;        // misc info (usually empty)
    uint16_t n_desc;        // description field
    uintptr_t n_value;      // value of symbol
};

#endif /* !__KERN_DEBUG_STAB_H__ */
```

[ucore/lab8_result/kern/debug/kmonitor.h] ++++++++++++++++++++++++++++++

```
#ifndef __KERN_DEBUG_MONITOR_H__
#define __KERN_DEBUG_MONITOR_H__

#include <trap.h>

void kmonitor(struct trapframe *tf);

int mon_help(int argc, char **argv, struct trapframe *tf);
int mon_kerninfo(int argc, char **argv, struct trapframe *tf);
int mon_backtrace(int argc, char **argv, struct trapframe *tf);
int mon_continue(int argc, char **argv, struct trapframe *tf);
int mon_step(int argc, char **argv, struct trapframe *tf);
int mon_breakpoint(int argc, char **argv, struct trapframe *tf);
int mon_watchpoint(int argc, char **argv, struct trapframe *tf);
int mon_delete_dr(int argc, char **argv, struct trapframe *tf);
int mon_list_dr(int argc, char **argv, struct trapframe *tf);

#endif /* !__KERN_DEBUG_MONITOR_H__ */
```

[ucore/lab8_result/kern/debug/kmonitor.c] ++++++++++++++++++++++++++++++

```
#include <stdio.h>
#include <string.h>
#include <mmu.h>
#include <trap.h>
#include <kmonitor.h>
#include <kdebug.h>

/* *
 * Simple command-line kernel monitor useful for controlling the
 * kernel and exploring the system interactively.
 * */

struct command {
    const char *name;
```

```
    const char *desc;
    // return -1 to force monitor to exit
    int(*func)(int argc, char **argv, struct trapframe *tf);
};

static struct command commands[] = {
    {"help", "Display this list of commands.", mon_help},
    {"kerninfo", "Display information about the kernel.", mon_kerninfo},
    {"backtrace", "Print backtrace of stack frame.", mon_backtrace},
};

/* return if kernel is panic, in kern/debug/panic.c */
bool is_kernel_panic(void);

#define NCOMMANDS (sizeof(commands)/sizeof(struct command))

/***** Kernel monitor command interpreter *****/

#define MAXARGS         16
#define WHITESPACE      " \t\n\r"

/* parse - parse the command buffer into whitespace-separated arguments */
static int
parse(char *buf, char **argv) {
    int argc = 0;
    while (1) {
        // find global whitespace
        while (*buf != '\0' && strchr(WHITESPACE, *buf) != NULL) {
            *buf ++ = '\0';
        }
        if (*buf == '\0') {
            break;
        }

        // save and scan past next arg
        if (argc == MAXARGS - 1) {
            cprintf("Too many arguments (max %d).\n", MAXARGS);
        }
        argv[argc ++] = buf;
        while (*buf != '\0' && strchr(WHITESPACE, *buf) == NULL) {
            buf ++;
        }
    }
    return argc;
}

/* *
 * runcmd - parse the input string, split it into separated arguments
 * and then lookup and invoke some related commands/
 * */
static int
runcmd(char *buf, struct trapframe *tf) {
    char *argv[MAXARGS];
    int argc = parse(buf, argv);
    if (argc == 0) {
        return 0;
    }
    int i;
    for (i = 0; i < NCOMMANDS; i ++) {
        if (strcmp(commands[i].name, argv[0]) == 0) {
            return commands[i].func(argc - 1, argv + 1, tf);
        }
    }
    cprintf("Unknown command '%s'\n", argv[0]);
    return 0;
```

```
}

/***** Implementations of basic kernel monitor commands *****/

void
kmonitor(struct trapframe *tf) {
    cprintf("Welcome to the kernel debug monitor!!\n");
    cprintf("Type 'help' for a list of commands.\n");

    if (tf != NULL) {
        print_trapframe(tf);
    }

    char *buf;
    while (1) {
        if ((buf = readline("K> ")) != NULL) {
            if (runcmd(buf, tf) < 0) {
                break;
            }
        }
    }
}

/* mon_help - print the information about mon_* functions */
int
mon_help(int argc, char **argv, struct trapframe *tf) {
    int i;
    for (i = 0; i < NCOMMANDS; i ++) {
        cprintf("%s - %s\n", commands[i].name, commands[i].desc);
    }
    return 0;
}

/* *
 * mon_kerninfo - call print_kerninfo in kern/debug/kdebug.c to
 * print the memory occupancy in kernel.
 * */
int
mon_kerninfo(int argc, char **argv, struct trapframe *tf) {
    print_kerninfo();
    return 0;
}

/* *
 * mon_backtrace - call print_stackframe in kern/debug/kdebug.c to
 * print a backtrace of the stack.
 * */
int
mon_backtrace(int argc, char **argv, struct trapframe *tf) {
    print_stackframe();
    return 0;
}

[ucore/lab8_result/kern/mm/swap.c] ++++++++++++++++++++++++++++++
#include <swap.h>
#include <swapfs.h>
#include <swap_fifo.h>
#include <stdio.h>
#include <string.h>
#include <memlayout.h>
#include <pmm.h>
#include <mmu.h>
#include <default_pmm.h>
#include <kdebug.h>
```

```
// the valid vaddr for check is between 0~CHECK_VALID_VADDR-1
#define CHECK_VALID_VIR_PAGE_NUM 5
#define BEING_CHECK_VALID_VADDR 0X1000
#define CHECK_VALID_VADDR (CHECK_VALID_VIR_PAGE_NUM+1)*0x1000
// the max number of valid physical page for check
#define CHECK_VALID_PHY_PAGE_NUM 4
// the max access seq number
#define MAX_SEQ_NO 10

static struct swap_manager *sm;
size_t max_swap_offset;

volatile int swap_init_ok = 0;

unsigned int swap_page[CHECK_VALID_VIR_PAGE_NUM];

unsigned int swap_in_seq_no[MAX_SEQ_NO],swap_out_seq_no[MAX_SEQ_NO];

static void check_swap(void);

int
swap_init(void)
{
     swapfs_init();

     if (!(1024 <= max_swap_offset && max_swap_offset < MAX_SWAP_OFFSET_LIMIT))
     {
          panic("bad max_swap_offset %08x.\n", max_swap_offset);
     }


     sm = &swap_manager_fifo;
     int r = sm->init();

     if (r == 0)
     {
          swap_init_ok = 1;
          cprintf("SWAP: manager = %s\n", sm->name);
          check_swap();
     }

     return r;
}

int
swap_init_mm(struct mm_struct *mm)
{
     return sm->init_mm(mm);
}

int
swap_tick_event(struct mm_struct *mm)
{
     return sm->tick_event(mm);
}

int
swap_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{
     return sm->map_swappable(mm, addr, page, swap_in);
}

int
swap_set_unswappable(struct mm_struct *mm, uintptr_t addr)
{
```

```
     return sm->set_unswappable(mm, addr);
}

volatile unsigned int swap_out_num=0;

int
swap_out(struct mm_struct *mm, int n, int in_tick)
{
     int i;
     for (i = 0; i != n; ++ i)
     {
          uintptr_t v;
          //struct Page **ptr_page=NULL;
          struct Page *page;
          // cprintf("i %d, SWAP: call swap_out_victim\n",i);
          int r = sm->swap_out_victim(mm, &page, in_tick);
          if (r != 0) {
                    cprintf("i %d, swap_out: call swap_out_victim failed\n",i);
                  break;
          }
          //assert(!PageReserved(page));

          //cprintf("SWAP: choose victim page 0x%08x\n", page);

          v=page->pra_vaddr;
          pte_t *ptep = get_pte(mm->pgdir, v, 0);
          assert((*ptep & PTE_P) != 0);

          if (swapfs_write( (page->pra_vaddr/PGSIZE+1)<<8, page) != 0) {
                    cprintf("SWAP: failed to save\n");
                    sm->map_swappable(mm, v, page, 0);
                    continue;
          }
          else {
                    cprintf("swap_out: i %d, store page in vaddr 0x%x to disk swap entry %d\n"
, i, v, page->pra_vaddr/PGSIZE+1);
                    *ptep = (page->pra_vaddr/PGSIZE+1)<<8;
                    free_page(page);
          }

          tlb_invalidate(mm->pgdir, v);
     }
     return i;
}

int
swap_in(struct mm_struct *mm, uintptr_t addr, struct Page **ptr_result)
{
     struct Page *result = alloc_page();
     assert(result!=NULL);

     pte_t *ptep = get_pte(mm->pgdir, addr, 0);
     // cprintf("SWAP: load ptep %x swap entry %d to vaddr 0x%08x, page %x, No %d\n", ptep, (*
ptep)>>8, addr, result, (result-pages));

     int r;
     if ((r = swapfs_read((*ptep), result)) != 0)
     {
        assert(r!=0);
     }
     cprintf("swap_in: load disk swap entry %d with swap_page in vadr 0x%x\n", (*ptep)>>8, add
r);
     *ptr_result=result;
     return 0;
}
```

```
static inline void
check_content_set(void)
{
     *(unsigned char *)0x1000 = 0x0a;
     assert(pgfault_num==1);
     *(unsigned char *)0x1010 = 0x0a;
     assert(pgfault_num==1);
     *(unsigned char *)0x2000 = 0x0b;
     assert(pgfault_num==2);
     *(unsigned char *)0x2010 = 0x0b;
     assert(pgfault_num==2);
     *(unsigned char *)0x3000 = 0x0c;
     assert(pgfault_num==3);
     *(unsigned char *)0x3010 = 0x0c;
     assert(pgfault_num==3);
     *(unsigned char *)0x4000 = 0x0d;
     assert(pgfault_num==4);
     *(unsigned char *)0x4010 = 0x0d;
     assert(pgfault_num==4);
}

static inline int
check_content_access(void)
{
    int ret = sm->check_swap();
    return ret;
}

struct Page * check_rp[CHECK_VALID_PHY_PAGE_NUM];
pte_t * check_ptep[CHECK_VALID_PHY_PAGE_NUM];
unsigned int check_swap_addr[CHECK_VALID_VIR_PAGE_NUM];

extern free_area_t free_area;

#define free_list (free_area.free_list)
#define nr_free (free_area.nr_free)

static void
check_swap(void)
{
    //backup mem env
     int ret, count = 0, total = 0, i;
     list_entry_t *le = &free_list;
     while ((le = list_next(le)) != &free_list) {
        struct Page *p = le2page(le, page_link);
        assert(PageProperty(p));
        count ++, total += p->property;
     }
     assert(total == nr_free_pages());
     cprintf("BEGIN check_swap: count %d, total %d\n",count,total);

     //now we set the phy pages env
     struct mm_struct *mm = mm_create();
     assert(mm != NULL);

     extern struct mm_struct *check_mm_struct;
     assert(check_mm_struct == NULL);

     check_mm_struct = mm;

     pde_t *pgdir = mm->pgdir = boot_pgdir;
     assert(pgdir[0] == 0);
```

```
    struct vma_struct *vma = vma_create(BEING_CHECK_VALID_VADDR, CHECK_VALID_VADDR, VM_WRITE
| VM_READ);
    assert(vma != NULL);

    insert_vma_struct(mm, vma);

    //setup the temp Page Table vaddr 0~4MB
    cprintf("setup Page Table for vaddr 0X1000, so alloc a page\n");
    pte_t *temp_ptep=NULL;
    temp_ptep = get_pte(mm->pgdir, BEING_CHECK_VALID_VADDR, 1);
    assert(temp_ptep!= NULL);
    cprintf("setup Page Table vaddr 0~4MB OVER!\n");

    for (i=0;i<CHECK_VALID_PHY_PAGE_NUM;i++) {
         check_rp[i] = alloc_page();
         assert(check_rp[i] != NULL );
         assert(!PageProperty(check_rp[i]));
    }
    list_entry_t free_list_store = free_list;
    list_init(&free_list);
    assert(list_empty(&free_list));

    //assert(alloc_page() == NULL);

    unsigned int nr_free_store = nr_free;
    nr_free = 0;
    for (i=0;i<CHECK_VALID_PHY_PAGE_NUM;i++) {
       free_pages(check_rp[i],1);
    }
    assert(nr_free==CHECK_VALID_PHY_PAGE_NUM);

    cprintf("set up init env for check_swap begin!\n");
    //setup initial vir_page<->phy_page environment for page relpacement algorithm


    pgfault_num=0;

    check_content_set();
    assert( nr_free == 0);
    for(i = 0; i<MAX_SEQ_NO ; i++)
        swap_out_seq_no[i]=swap_in_seq_no[i]=-1;

    for (i= 0;i<CHECK_VALID_PHY_PAGE_NUM;i++) {
        check_ptep[i]=0;
        check_ptep[i] = get_pte(pgdir, (i+1)*0x1000, 0);
        //cprintf("i %d, check_ptep addr %x, value %x\n", i, check_ptep[i], *check_ptep[i]);
        assert(check_ptep[i] != NULL);
        assert(pte2page(*check_ptep[i]) == check_rp[i]);
        assert((*check_ptep[i] & PTE_P));
    }
    cprintf("set up init env for check_swap over!\n");
    // now access the virt pages to test  page relpacement algorithm
    ret=check_content_access();
    assert(ret==0);

    //restore kernel mem env
    for (i=0;i<CHECK_VALID_PHY_PAGE_NUM;i++) {
        free_pages(check_rp[i],1);
    }

    //free_page(pte2page(*temp_ptep));
   free_page(pde2page(pgdir[0]));
    pgdir[0] = 0;
    mm->pgdir = NULL;
```

```
    mm_destroy(mm);
    check_mm_struct = NULL;

    nr_free = nr_free_store;
    free_list = free_list_store;


    le = &free_list;
    while ((le = list_next(le)) != &free_list) {
        struct Page *p = le2page(le, page_link);
        count --, total -= p->property;
    }
    cprintf("count is %d, total is %d\n",count,total);
    //assert(count == 0);

    cprintf("check_swap() succeeded!\n");
}
[ucore/lab8_result/kern/mm/kmalloc.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_SLAB_H__
#define __KERN_MM_SLAB_H__

#include <defs.h>

#define KMALLOC_MAX_ORDER       10

void kmalloc_init(void);

void *kmalloc(size_t n);
void kfree(void *objp);

size_t kallocated(void);

#endif /* !__KERN_MM_SLAB_H__ */

[ucore/lab8_result/kern/mm/default_pmm.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_DEFAULT_PMM_H__
#define  __KERN_MM_DEFAULT_PMM_H__

#include <pmm.h>

extern const struct pmm_manager default_pmm_manager;
extern free_area_t free_area;
#endif /* ! __KERN_MM_DEFAULT_PMM_H__ */

[ucore/lab8_result/kern/mm/swap.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_SWAP_H__
#define __KERN_MM_SWAP_H__

#include <defs.h>
#include <memlayout.h>
#include <pmm.h>
#include <vmm.h>

/* *
 * swap_entry_t
 * ------------------------------------------
 * |         offset        |   reserved   | 0 |
 * ------------------------------------------
 *           24 bits            7 bits    1 bit
 * */

#define MAX_SWAP_OFFSET_LIMIT                   (1 << 24)

extern size_t max_swap_offset;
```

```
/* *
 * swap_offset - takes a swap_entry (saved in pte), and returns
 * the corresponding offset in swap mem_map.
 * */
#define swap_offset(entry) ({                                               \
               size_t __offset = (entry >> 8);                                \
               if (!(__offset > 0 && __offset < max_swap_offset)) {           \
                    panic("invalid swap_entry_t = %08x.\n", entry);           \
               }                                                              \
               __offset;                                                      \
          })

struct swap_manager
{
     const char *name;
     /* Global initialization for the swap manager */
     int (*init)            (void);
     /* Initialize the priv data inside mm_struct */
     int (*init_mm)         (struct mm_struct *mm);
     /* Called when tick interrupt occured */
     int (*tick_event)      (struct mm_struct *mm);
     /* Called when map a swappable page into the mm_struct */
     int (*map_swappable)   (struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap
_in);
     /* When a page is marked as shared, this routine is called to
      * delete the addr entry from the swap manager */
     int (*set_unswappable) (struct mm_struct *mm, uintptr_t addr);
     /* Try to swap out a page, return then victim */
     int (*swap_out_victim) (struct mm_struct *mm, struct Page **ptr_page, int in_tick);
     /* check the page relpacement algorithm */
     int (*check_swap)(void);
};

extern volatile int swap_init_ok;
int swap_init(void);
int swap_init_mm(struct mm_struct *mm);
int swap_tick_event(struct mm_struct *mm);
int swap_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in);
int swap_set_unswappable(struct mm_struct *mm, uintptr_t addr);
int swap_out(struct mm_struct *mm, int n, int in_tick);
int swap_in(struct mm_struct *mm, uintptr_t addr, struct Page **ptr_result);

//#define MEMBER_OFFSET(m,t) ((int)(&((t *)0)->m))
//#define FROM_MEMBER(m,t,a) ((t *)((char *)(a) - MEMBER_OFFSET(m,t)))

#endif
[ucore/lab8_result/kern/mm/vmm.c] ++++++++++++++++++++++++++++++
#include <vmm.h>
#include <sync.h>
#include <string.h>
#include <assert.h>
#include <stdio.h>
#include <error.h>
#include <pmm.h>
#include <x86.h>
#include <swap.h>
#include <kmalloc.h>

/*
  vmm design include two parts: mm_struct (mm) & vma_struct (vma)
  mm is the memory manager for the set of continuous virtual memory
  area which have the same PDT. vma is a continuous virtual memory area.
  There a linear link list for vma & a redblack link list for vma in mm.
---------------
  mm related functions:
```

```
   golbal functions
     struct mm_struct * mm_create(void)
     void mm_destroy(struct mm_struct *mm)
     int do_pgfault(struct mm_struct *mm, uint32_t error_code, uintptr_t addr)
--------------
  vma related functions:
   global functions
     struct vma_struct * vma_create (uintptr_t vm_start, uintptr_t vm_end,...)
     void insert_vma_struct(struct mm_struct *mm, struct vma_struct *vma)
     struct vma_struct * find_vma(struct mm_struct *mm, uintptr_t addr)
   local functions
     inline void check_vma_overlap(struct vma_struct *prev, struct vma_struct *next)
---------------
   check correctness functions
     void check_vmm(void);
     void check_vma_struct(void);
     void check_pgfault(void);
*/

static void check_vmm(void);
static void check_vma_struct(void);
static void check_pgfault(void);

// mm_create -  alloc a mm_struct & initialize it.
struct mm_struct *
mm_create(void) {
    struct mm_struct *mm = kmalloc(sizeof(struct mm_struct));

    if (mm != NULL) {
        list_init(&(mm->mmap_list));
        mm->mmap_cache = NULL;
        mm->pgdir = NULL;
        mm->map_count = 0;

        if (swap_init_ok) swap_init_mm(mm);
        else mm->sm_priv = NULL;

        set_mm_count(mm, 0);
        sem_init(&(mm->mm_sem), 1);
    }
    return mm;
}

// vma_create - alloc a vma_struct & initialize it. (addr range: vm_start~vm_end)
struct vma_struct *
vma_create(uintptr_t vm_start, uintptr_t vm_end, uint32_t vm_flags) {
    struct vma_struct *vma = kmalloc(sizeof(struct vma_struct));

    if (vma != NULL) {
        vma->vm_start = vm_start;
        vma->vm_end = vm_end;
        vma->vm_flags = vm_flags;
    }
    return vma;
}


// find_vma - find a vma  (vma->vm_start <= addr <= vma_vm_end)
struct vma_struct *
find_vma(struct mm_struct *mm, uintptr_t addr) {
    struct vma_struct *vma = NULL;
    if (mm != NULL) {
        vma = mm->mmap_cache;
        if (!(vma != NULL && vma->vm_start <= addr && vma->vm_end > addr)) {
                bool found = 0;
```

```
                list_entry_t *list = &(mm->mmap_list), *le = list;
                while ((le = list_next(le)) != list) {
                    vma = le2vma(le, list_link);
                    if (vma->vm_start<=addr && addr < vma->vm_end) {
                        found = 1;
                        break;
                    }
                }
                if (!found) {
                    vma = NULL;
                }
        }
        if (vma != NULL) {
            mm->mmap_cache = vma;
        }
    }
    return vma;
}


// check_vma_overlap - check if vma1 overlaps vma2 ?
static inline void
check_vma_overlap(struct vma_struct *prev, struct vma_struct *next) {
    assert(prev->vm_start < prev->vm_end);
    assert(prev->vm_end <= next->vm_start);
    assert(next->vm_start < next->vm_end);
}


// insert_vma_struct -insert vma in mm's list link
void
insert_vma_struct(struct mm_struct *mm, struct vma_struct *vma) {
    assert(vma->vm_start < vma->vm_end);
    list_entry_t *list = &(mm->mmap_list);
    list_entry_t *le_prev = list, *le_next;

        list_entry_t *le = list;
        while ((le = list_next(le)) != list) {
            struct vma_struct *mmap_prev = le2vma(le, list_link);
            if (mmap_prev->vm_start > vma->vm_start) {
                break;
            }
            le_prev = le;
        }

    le_next = list_next(le_prev);

    /* check overlap */
    if (le_prev != list) {
        check_vma_overlap(le2vma(le_prev, list_link), vma);
    }
    if (le_next != list) {
        check_vma_overlap(vma, le2vma(le_next, list_link));
    }

    vma->vm_mm = mm;
    list_add_after(le_prev, &(vma->list_link));

    mm->map_count ++;
}

// mm_destroy - free mm and mm internal fields
void
mm_destroy(struct mm_struct *mm) {
    assert(mm_count(mm) == 0);
```

```
    list_entry_t *list = &(mm->mmap_list), *le;
    while ((le = list_next(list)) != list) {
        list_del(le);
        kfree(le2vma(le, list_link));  //kfree vma
    }
    kfree(mm); //kfree mm
    mm=NULL;
}

int
mm_map(struct mm_struct *mm, uintptr_t addr, size_t len, uint32_t vm_flags,
       struct vma_struct **vma_store) {
    uintptr_t start = ROUNDDOWN(addr, PGSIZE), end = ROUNDUP(addr + len, PGSIZE);
    if (!USER_ACCESS(start, end)) {
        return -E_INVAL;
    }

    assert(mm != NULL);

    int ret = -E_INVAL;

    struct vma_struct *vma;
    if ((vma = find_vma(mm, start)) != NULL && end > vma->vm_start) {
        goto out;
    }
    ret = -E_NO_MEM;

    if ((vma = vma_create(start, end, vm_flags)) == NULL) {
        goto out;
    }
    insert_vma_struct(mm, vma);
    if (vma_store != NULL) {
        *vma_store = vma;
    }
    ret = 0;

out:
    return ret;
}

int
dup_mmap(struct mm_struct *to, struct mm_struct *from) {
    assert(to != NULL && from != NULL);
    list_entry_t *list = &(from->mmap_list), *le = list;
    while ((le = list_prev(le)) != list) {
        struct vma_struct *vma, *nvma;
        vma = le2vma(le, list_link);
        nvma = vma_create(vma->vm_start, vma->vm_end, vma->vm_flags);
        if (nvma == NULL) {
            return -E_NO_MEM;
        }

        insert_vma_struct(to, nvma);

        bool share = 0;
        if (copy_range(to->pgdir, from->pgdir, vma->vm_start, vma->vm_end, share) != 0) {
            return -E_NO_MEM;
        }
    }
    return 0;
}

void
exit_mmap(struct mm_struct *mm) {
```

```
    assert(mm != NULL && mm_count(mm) == 0);
    pde_t *pgdir = mm->pgdir;
    list_entry_t *list = &(mm->mmap_list), *le = list;
    while ((le = list_next(le)) != list) {
        struct vma_struct *vma = le2vma(le, list_link);
        unmap_range(pgdir, vma->vm_start, vma->vm_end);
    }
    while ((le = list_next(le)) != list) {
        struct vma_struct *vma = le2vma(le, list_link);
        exit_range(pgdir, vma->vm_start, vma->vm_end);
    }
}

bool
copy_from_user(struct mm_struct *mm, void *dst, const void *src, size_t len, bool writable) {
    if (!user_mem_check(mm, (uintptr_t)src, len, writable)) {
        return 0;
    }
    memcpy(dst, src, len);
    return 1;
}

bool
copy_to_user(struct mm_struct *mm, void *dst, const void *src, size_t len) {
    if (!user_mem_check(mm, (uintptr_t)dst, len, 1)) {
        return 0;
    }
    memcpy(dst, src, len);
    return 1;
}

// vmm_init - initialize virtual memory management
//          - now just call check_vmm to check correctness of vmm
void
vmm_init(void) {
    check_vmm();
}

// check_vmm - check correctness of vmm
static void
check_vmm(void) {
    size_t nr_free_pages_store = nr_free_pages();

    check_vma_struct();
    check_pgfault();

    //assert(nr_free_pages_store == nr_free_pages());

    cprintf("check_vmm() succeeded.\n");
}

static void
check_vma_struct(void) {
    size_t nr_free_pages_store = nr_free_pages();

    struct mm_struct *mm = mm_create();
    assert(mm != NULL);

    int step1 = 10, step2 = step1 * 10;

    int i;
    for (i = step1; i >= 1; i --) {
        struct vma_struct *vma = vma_create(i * 5, i * 5 + 2, 0);
        assert(vma != NULL);
        insert_vma_struct(mm, vma);
```

```
    }

    for (i = step1 + 1; i <= step2; i ++) {
        struct vma_struct *vma = vma_create(i * 5, i * 5 + 2, 0);
        assert(vma != NULL);
        insert_vma_struct(mm, vma);
    }

    list_entry_t *le = list_next(&(mm->mmap_list));

    for (i = 1; i <= step2; i ++) {
        assert(le != &(mm->mmap_list));
        struct vma_struct *mmap = le2vma(le, list_link);
        assert(mmap->vm_start == i * 5 && mmap->vm_end == i * 5 + 2);
        le = list_next(le);
    }

    for (i = 5; i <= 5 * step2; i +=5) {
        struct vma_struct *vma1 = find_vma(mm, i);
        assert(vma1 != NULL);
        struct vma_struct *vma2 = find_vma(mm, i+1);
        assert(vma2 != NULL);
        struct vma_struct *vma3 = find_vma(mm, i+2);
        assert(vma3 == NULL);
        struct vma_struct *vma4 = find_vma(mm, i+3);
        assert(vma4 == NULL);
        struct vma_struct *vma5 = find_vma(mm, i+4);
        assert(vma5 == NULL);

        assert(vma1->vm_start == i  && vma1->vm_end == i  + 2);
        assert(vma2->vm_start == i  && vma2->vm_end == i  + 2);
    }

    for (i =4; i>=0; i--) {
        struct vma_struct *vma_below_5= find_vma(mm,i);
        if (vma_below_5 != NULL ) {
           cprintf("vma_below_5: i %x, start %x, end %x\n",i, vma_below_5->vm_start, vma_below
_5->vm_end);
        }
        assert(vma_below_5 == NULL);
    }

    mm_destroy(mm);

  //  assert(nr_free_pages_store == nr_free_pages());

    cprintf("check_vma_struct() succeeded!\n");
}

struct mm_struct *check_mm_struct;

// check_pgfault - check correctness of pgfault handler
static void
check_pgfault(void) {
    size_t nr_free_pages_store = nr_free_pages();

    check_mm_struct = mm_create();
    assert(check_mm_struct != NULL);

    struct mm_struct *mm = check_mm_struct;
    pde_t *pgdir = mm->pgdir = boot_pgdir;
    assert(pgdir[0] == 0);

    struct vma_struct *vma = vma_create(0, PTSIZE, VM_WRITE);
    assert(vma != NULL);
```

```
    insert_vma_struct(mm, vma);

    uintptr_t addr = 0x100;
    assert(find_vma(mm, addr) == vma);

    int i, sum = 0;
    for (i = 0; i < 100; i ++) {
        *(char *)(addr + i) = i;
        sum += i;
    }
    for (i = 0; i < 100; i ++) {
        sum -= *(char *)(addr + i);
    }
    assert(sum == 0);

    page_remove(pgdir, ROUNDDOWN(addr, PGSIZE));
    free_page(pde2page(pgdir[0]));
    pgdir[0] = 0;

    mm->pgdir = NULL;
    mm_destroy(mm);
    check_mm_struct = NULL;

    assert(nr_free_pages_store == nr_free_pages());

    cprintf("check_pgfault() succeeded!\n");
}
//page fault number
volatile unsigned int pgfault_num=0;

/* do_pgfault - interrupt handler to process the page fault execption
 * @mm         : the control struct for a set of vma using the same PDT
 * @error_code : the error code recorded in trapframe->tf_err which is setted by x86 hardware
 * @addr       : the addr which causes a memory access exception, (the contents of the CR2 reg
ister)
 *
 * CALL GRAPH: trap--> trap_dispatch-->pgfault_handler-->do_pgfault
 * The processor provides ucore's do_pgfault function with two items of information to aid in
diagnosing
 * the exception and recovering from it.
 *   (1) The contents of the CR2 register. The processor loads the CR2 register with the
 *       32-bit linear address that generated the exception. The do_pgfault fun can
 *       use this address to locate the corresponding page directory and page-table
 *       entries.
 *   (2) An error code on the kernel stack. The error code for a page fault has a format diffe
rent from
 *       that for other exceptions. The error code tells the exception handler three things:
 *         -- The P flag   (bit 0) indicates whether the exception was due to a not-present pa
ge (0)
 *            or to either an access rights violation or the use of a reserved bit (1).
 *         -- The W/R flag (bit 1) indicates whether the memory access that caused the excepti
on
 *            was a read (0) or write (1).
 *         -- The U/S flag (bit 2) indicates whether the processor was executing at user mode
(1)
 *            or supervisor mode (0) at the time of the exception.
 */
int
do_pgfault(struct mm_struct *mm, uint32_t error_code, uintptr_t addr) {
    int ret = -E_INVAL;
    //try to find a vma which include addr
    struct vma_struct *vma = find_vma(mm, addr);

    pgfault_num++;
```

```
    //If the addr is in the range of a mm's vma?
    if (vma == NULL || vma->vm_start > addr) {
        cprintf("not valid addr %x, and  can not find it in vma\n", addr);
        goto failed;
    }
    //check the error_code
    switch (error_code & 3) {
    default:
            /* error code flag : default is 3 ( W/R=1, P=1): write, present */
    case 2: /* error code flag : (W/R=1, P=0): write, not present */
        if (!(vma->vm_flags & VM_WRITE)) {
            cprintf("do_pgfault failed: error code flag = write AND not present, but the addr'
s vma cannot write\n");
            goto failed;
        }
        break;
    case 1: /* error code flag : (W/R=0, P=1): read, present */
        cprintf("do_pgfault failed: error code flag = read AND present\n");
        goto failed;
    case 0: /* error code flag : (W/R=0, P=0): read, not present */
        if (!(vma->vm_flags & (VM_READ | VM_EXEC))) {
            cprintf("do_pgfault failed: error code flag = read AND not present, but the addr's
 vma cannot read or exec\n");
            goto failed;
        }
    }
    /* IF (write an existed addr ) OR
     *    (write an non_existed addr && addr is writable) OR
     *    (read  an non_existed addr && addr is readable)
     * THEN
     *    continue process
     */
    uint32_t perm = PTE_U;
    if (vma->vm_flags & VM_WRITE) {
        perm |= PTE_W;
    }
    addr = ROUNDDOWN(addr, PGSIZE);

    ret = -E_NO_MEM;

    pte_t *ptep=NULL;
    /*LAB3 EXERCISE 1: YOUR CODE
    * Maybe you want help comment, BELOW comments can help you finish the code
    *
    * Some Useful MACROs and DEFINEs, you can use them in below implementation.
    * MACROs or Functions:
    *   get_pte : get an pte and return the kernel virtual address of this pte for la
    *             if the PT contians this pte didn't exist, alloc a page for PT (notice the 3t
h parameter '1')
    *   pgdir_alloc_page : call alloc_page & page_insert functions to allocate a page size mem
ory & setup
    *             an addr map pa<--->la with linear address la and the PDT pgdir
    * DEFINES:
    *   VM_WRITE  : If vma->vm_flags & VM_WRITE == 1/0, then the vma is writable/non writable
    *   PTE_W           0x002                   // page table/directory entry flags bit : Writ
eable
    *   PTE_U           0x004                   // page table/directory entry flags bit : User
 can access
    * VARIABLES:
    *   mm->pgdir : the PDT of these vma
    *
    */
#if 0
    /*LAB3 EXERCISE 1: YOUR CODE*/
    ptep = ???              //(1) try to find a pte, if pte's PT(Page Table) isn't existed, th
```

```
en create a PT.
    if (*ptep == 0) {
                                //(2) if the phy addr isn't exist, then alloc a page & map the phy
 addr with logical addr

    }
    else {
    /*LAB3 EXERCISE 2: YOUR CODE
    * Now we think this pte is a  swap entry, we should load data from disk to a page with phy
 addr,
    * and map the phy addr with logical addr, trigger swap manager to record the access situat
ion of this page.
    *
    *  Some Useful MACROs and DEFINEs, you can use them in below implementation.
    *  MACROs or Functions:
    *    swap_in(mm, addr, &page) : alloc a memory page, then according to the swap entry in P
TE for addr,
    *                                find the addr of disk page, read the content of disk page
into this memroy page
    *    page_insert ï¼\232 build the map of phy addr of an Page with the linear addr la
    *    swap_map_swappable ï¼\232 set the page swappable
    */
    /*
     * LAB5 CHALLENGE ( the implmentation Copy on Write)
                There are 2 situlations when code comes here.
                  1) *ptep & PTE_P == 1, it means one process try to write a readonly page.
                     If the vma includes this addr is writable, then we can set the page writa
ble by rewrite the *ptep.
                     This method could be used to implement the Copy on Write (COW) thchnology
(a fast fork process method).
                  2) *ptep & PTE_P == 0 & but *ptep!=0, it means this pte is a  swap entry.
                     We should add the LAB3's results here.
     */
        if(swap_init_ok) {
            struct Page *page=NULL;
                                //(1ï¼\211According to the mm AND addr, try to load the co
ntent of right disk page
                                //    into the memory which page managed.
                                //(2) According to the mm, addr AND page, setup the map of
 phy addr <---> logical addr
                                //(3) make the page swappable.
                                //(4) [NOTICE]: you myabe need to update your lab3's imple
mentation for LAB5's normal execution.
        }
        else {
            cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
            goto failed;
        }
   }
#endif
    // try to find a pte, if pte's PT(Page Table) isn't existed, then create a PT.
    // (notice the 3th parameter '1')
    if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {
        cprintf("get_pte in do_pgfault failed\n");
        goto failed;
    }

    if (*ptep == 0) { // if the phy addr isn't exist, then alloc a page & map the phy addr wit
h logical addr
        if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {
            cprintf("pgdir_alloc_page in do_pgfault failed\n");
            goto failed;
        }
    }
    else {
```

```
        struct Page *page=NULL;
        cprintf("do pgfault: ptep %x, pte %x\n",ptep, *ptep);
        if (*ptep & PTE_P) {
            //if process write to this existed readonly page (PTE_P means existed), then shoul
d be here now.
            //we can implement the delayed memory space copy for fork child process (AKA copy
on write, COW).
            //we didn't implement now, we will do it in future.
            panic("error write a non-writable pte");
            //page = pte2page(*ptep);
        } else{
           // if this pte is a swap entry, then load data from disk to a page with phy addr
           // and call page_insert to map the phy addr with logical addr
           if(swap_init_ok) {
               if ((ret = swap_in(mm, addr, &page)) != 0) {
                   cprintf("swap_in in do_pgfault failed\n");
                   goto failed;
               }

           }
           else {
            cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
            goto failed;
           }
       }
       page_insert(mm->pgdir, page, addr, perm);
       swap_map_swappable(mm, addr, page, 1);
       page->pra_vaddr = addr;
   }
   ret = 0;
failed:
    return ret;
}

bool
user_mem_check(struct mm_struct *mm, uintptr_t addr, size_t len, bool write) {
    if (mm != NULL) {
        if (!USER_ACCESS(addr, addr + len)) {
            return 0;
        }
        struct vma_struct *vma;
        uintptr_t start = addr, end = addr + len;
        while (start < end) {
            if ((vma = find_vma(mm, start)) == NULL || start < vma->vm_start) {
                return 0;
            }
            if (!(vma->vm_flags & ((write) ? VM_WRITE : VM_READ))) {
                return 0;
            }
            if (write && (vma->vm_flags & VM_STACK)) {
                if (start < vma->vm_start + PGSIZE) { //check stack start & size
                    return 0;
                }
            }
            start = vma->vm_end;
        }
        return 1;
    }
    return KERN_ACCESS(addr, addr + len);
}

bool
copy_string(struct mm_struct *mm, char *dst, const char *src, size_t maxn) {
    size_t alen, part = ROUNDDOWN((uintptr_t)src + PGSIZE, PGSIZE) - (uintptr_t)src;
    while (1) {
```

```
        if (part > maxn) {
            part = maxn;
        }
        if (!user_mem_check(mm, (uintptr_t)src, part, 0)) {
            return 0;
        }
        if ((alen = strnlen(src, part)) < part) {
            memcpy(dst, src, alen + 1);
            return 1;
        }
        if (part == maxn) {
            return 0;
        }
        memcpy(dst, src, part);
        dst += part, src += part, maxn -= part;
        part = PGSIZE;
    }
}
[ucore/lab8_result/kern/mm/pmm.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <x86.h>
#include <stdio.h>
#include <string.h>
#include <mmu.h>
#include <memlayout.h>
#include <pmm.h>
#include <default_pmm.h>
#include <sync.h>
#include <error.h>
#include <swap.h>
#include <vmm.h>
#include <kmalloc.h>

/* *
 * Task State Segment:
 *
 * The TSS may reside anywhere in memory. A special segment register called
 * the Task Register (TR) holds a segment selector that points a valid TSS
 * segment descriptor which resides in the GDT. Therefore, to use a TSS
 * the following must be done in function gdt_init:
 *   - create a TSS descriptor entry in GDT
 *   - add enough information to the TSS in memory as needed
 *   - load the TR register with a segment selector for that segment
 *
 * There are several fileds in TSS for specifying the new stack pointer when a
 * privilege level change happens. But only the fields SS0 and ESP0 are useful
 * in our os kernel.
 *
 * The field SS0 contains the stack segment selector for CPL = 0, and the ESP0
 * contains the new ESP value for CPL = 0. When an interrupt happens in protected
 * mode, the x86 CPU will look in the TSS for SS0 and ESP0 and load their value
 * into SS and ESP respectively.
 * */
static struct taskstate ts = {0};

// virtual address of physicall page array
struct Page *pages;
// amount of physical memory (in pages)
size_t npage = 0;

// virtual address of boot-time page directory
extern pde_t __boot_pgdir;
pde_t *boot_pgdir = &__boot_pgdir;
// physical address of boot-time page directory
uintptr_t boot_cr3;
```

```c
// physical memory management
const struct pmm_manager *pmm_manager;

/* *
 * The page directory entry corresponding to the virtual address range
 * [VPT, VPT + PTSIZE) points to the page directory itself. Thus, the page
 * directory is treated as a page table as well as a page directory.
 *
 * One result of treating the page directory as a page table is that all PTEs
 * can be accessed though a "virtual page table" at virtual address VPT. And the
 * PTE for number n is stored in vpt[n].
 *
 * A second consequence is that the contents of the current page directory will
 * always available at virtual address PGADDR(PDX(VPT), PDX(VPT), 0), to which
 * vpd is set bellow.
 * */
pte_t * const vpt = (pte_t *)VPT;
pde_t * const vpd = (pde_t *)PGADDR(PDX(VPT), PDX(VPT), 0);

/* *
 * Global Descriptor Table:
 *
 * The kernel and user segments are identical (except for the DPL). To load
 * the %ss register, the CPL must equal the DPL. Thus, we must duplicate the
 * segments for the user and the kernel. Defined as follows:
 *   - 0x0 :  unused (always faults -- for trapping NULL far pointers)
 *   - 0x8 :  kernel code segment
 *   - 0x10:  kernel data segment
 *   - 0x18:  user code segment
 *   - 0x20:  user data segment
 *   - 0x28:  defined for tss, initialized in gdt_init
 * */
static struct segdesc gdt[] = {
    SEG_NULL,
    [SEG_KTEXT] = SEG(STA_X | STA_R, 0x0, 0xFFFFFFFF, DPL_KERNEL),
    [SEG_KDATA] = SEG(STA_W, 0x0, 0xFFFFFFFF, DPL_KERNEL),
    [SEG_UTEXT] = SEG(STA_X | STA_R, 0x0, 0xFFFFFFFF, DPL_USER),
    [SEG_UDATA] = SEG(STA_W, 0x0, 0xFFFFFFFF, DPL_USER),
    [SEG_TSS]   = SEG_NULL,
};

static struct pseudodesc gdt_pd = {
    sizeof(gdt) - 1, (uintptr_t)gdt
};

static void check_alloc_page(void);
static void check_pgdir(void);
static void check_boot_pgdir(void);

/* *
 * lgdt - load the global descriptor table register and reset the
 * data/code segement registers for kernel.
 * */
static inline void
lgdt(struct pseudodesc *pd) {
    asm volatile ("lgdt (%0)" :: "r" (pd));
    asm volatile ("movw %%ax, %%gs" :: "a" (USER_DS));
    asm volatile ("movw %%ax, %%fs" :: "a" (USER_DS));
    asm volatile ("movw %%ax, %%es" :: "a" (KERNEL_DS));
    asm volatile ("movw %%ax, %%ds" :: "a" (KERNEL_DS));
    asm volatile ("movw %%ax, %%ss" :: "a" (KERNEL_DS));
    // reload cs
    asm volatile ("ljmp %0, $1f\n 1:\n" :: "i" (KERNEL_CS));
}
```

```
/* *
 * load_esp0 - change the ESP0 in default task state segment,
 * so that we can use different kernel stack when we trap frame
 * user to kernel.
 * */
void
load_esp0(uintptr_t esp0) {
    ts.ts_esp0 = esp0;
}

/* gdt_init - initialize the default GDT and TSS */
static void
gdt_init(void) {
    // set boot kernel stack and default SS0
    load_esp0((uintptr_t)bootstacktop);
    ts.ts_ss0 = KERNEL_DS;

    // initialize the TSS filed of the gdt
    gdt[SEG_TSS] = SEGTSS(STS_T32A, (uintptr_t)&ts, sizeof(ts), DPL_KERNEL);

    // reload all segment registers
    lgdt(&gdt_pd);

    // load the TSS
    ltr(GD_TSS);
}

//init_pmm_manager - initialize a pmm_manager instance
static void
init_pmm_manager(void) {
    pmm_manager = &default_pmm_manager;
    cprintf("memory management: %s\n", pmm_manager->name);
    pmm_manager->init();
}

//init_memmap - call pmm->init_memmap to build Page struct for free memory
static void
init_memmap(struct Page *base, size_t n) {
    pmm_manager->init_memmap(base, n);
}

//alloc_pages - call pmm->alloc_pages to allocate a continuous n*PAGESIZE memory
struct Page *
alloc_pages(size_t n) {
    struct Page *page=NULL;
    bool intr_flag;

    while (1)
    {
         local_intr_save(intr_flag);
         {
              page = pmm_manager->alloc_pages(n);
         }
         local_intr_restore(intr_flag);

         if (page != NULL || n > 1 || swap_init_ok == 0) break;

         extern struct mm_struct *check_mm_struct;
         //cprintf("page %x, call swap_out in alloc_pages %d\n",page, n);
         swap_out(check_mm_struct, n, 0);
    }
    //cprintf("n %d,get page %x, No %d in alloc_pages\n",n,page,(page-pages));
    return page;
}
```

```
//free_pages - call pmm->free_pages to free a continuous n*PAGESIZE memory
void
free_pages(struct Page *base, size_t n) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        pmm_manager->free_pages(base, n);
    }
    local_intr_restore(intr_flag);
}

//nr_free_pages - call pmm->nr_free_pages to get the size (nr*PAGESIZE)
//of current free memory
size_t
nr_free_pages(void) {
    size_t ret;
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        ret = pmm_manager->nr_free_pages();
    }
    local_intr_restore(intr_flag);
    return ret;
}

/* pmm_init - initialize the physical memory management */
static void
page_init(void) {
    struct e820map *memmap = (struct e820map *)(0x8000 + KERNBASE);
    uint64_t maxpa = 0;

    cprintf("e820map:\n");
    int i;
    for (i = 0; i < memmap->nr_map; i ++) {
        uint64_t begin = memmap->map[i].addr, end = begin + memmap->map[i].size;
        cprintf("  memory: %08llx, [%08llx, %08llx], type = %d.\n",
                memmap->map[i].size, begin, end - 1, memmap->map[i].type);
        if (memmap->map[i].type == E820_ARM) {
            if (maxpa < end && begin < KMEMSIZE) {
                maxpa = end;
            }
        }
    }
    if (maxpa > KMEMSIZE) {
        maxpa = KMEMSIZE;
    }

    extern char end[];

    npage = maxpa / PGSIZE;
    pages = (struct Page *)ROUNDUP((void *)end, PGSIZE);

    for (i = 0; i < npage; i ++) {
        SetPageReserved(pages + i);
    }

    uintptr_t freemem = PADDR((uintptr_t)pages + sizeof(struct Page) * npage);

    for (i = 0; i < memmap->nr_map; i ++) {
        uint64_t begin = memmap->map[i].addr, end = begin + memmap->map[i].size;
        if (memmap->map[i].type == E820_ARM) {
            if (begin < freemem) {
                begin = freemem;
            }
```

```
            if (end > KMEMSIZE) {
                end = KMEMSIZE;
            }
            if (begin < end) {
                begin = ROUNDUP(begin, PGSIZE);
                end = ROUNDDOWN(end, PGSIZE);
                if (begin < end) {
                    init_memmap(pa2page(begin), (end - begin) / PGSIZE);
                }
            }
        }
    }
}

//boot_map_segment - setup&enable the paging mechanism
// parameters
//  la:   linear address of this memory need to map (after x86 segment map)
//  size: memory size
//  pa:   physical address of this memory
//  perm: permission of this memory
static void
boot_map_segment(pde_t *pgdir, uintptr_t la, size_t size, uintptr_t pa, uint32_t perm) {
    assert(PGOFF(la) == PGOFF(pa));
    size_t n = ROUNDUP(size + PGOFF(la), PGSIZE) / PGSIZE;
    la = ROUNDDOWN(la, PGSIZE);
    pa = ROUNDDOWN(pa, PGSIZE);
    for (; n > 0; n --, la += PGSIZE, pa += PGSIZE) {
        pte_t *ptep = get_pte(pgdir, la, 1);
        assert(ptep != NULL);
        *ptep = pa | PTE_P | perm;
    }
}

//boot_alloc_page - allocate one page using pmm->alloc_pages(1)
// return value: the kernel virtual address of this allocated page
//note: this function is used to get the memory for PDT(Page Directory Table)&PT(Page Table)
static void *
boot_alloc_page(void) {
    struct Page *p = alloc_page();
    if (p == NULL) {
        panic("boot_alloc_page failed.\n");
    }
    return page2kva(p);
}

//pmm_init - setup a pmm to manage physical memory, build PDT&PT to setup paging mechanism
//         - check the correctness of pmm & paging mechanism, print PDT&PT
void
pmm_init(void) {
    // We've already enabled paging
    boot_cr3 = PADDR(boot_pgdir);

    //We need to alloc/free the physical memory (granularity is 4KB or other size).
    //So a framework of physical memory manager (struct pmm_manager)is defined in pmm.h
    //First we should init a physical memory manager(pmm) based on the framework.
    //Then pmm can alloc/free the physical memory.
    //Now the first_fit/best_fit/worst_fit/buddy_system pmm are available.
    init_pmm_manager();

    // detect physical memory space, reserve already used memory,
    // then use pmm->init_memmap to create free page list
    page_init();

    //use pmm->check to verify the correctness of the alloc/free function in a pmm
    check_alloc_page();
```

```
    check_pgdir();

    static_assert(KERNBASE % PTSIZE == 0 && KERNTOP % PTSIZE == 0);

    // recursively insert boot_pgdir in itself
    // to form a virtual page table at virtual address VPT
    boot_pgdir[PDX(VPT)] = PADDR(boot_pgdir) | PTE_P | PTE_W;

    // map all physical memory to linear memory with base linear addr KERNBASE
    // linear_addr KERNBASE ~ KERNBASE + KMEMSIZE = phy_addr 0 ~ KMEMSIZE
    boot_map_segment(boot_pgdir, KERNBASE, KMEMSIZE, 0, PTE_W);

    // Since we are using bootloader's GDT,
    // we should reload gdt (second time, the last time) to get user segments and the TSS
    // map virtual_addr 0 ~ 4G = linear_addr 0 ~ 4G
    // then set kernel stack (ss:esp) in TSS, setup TSS in gdt, load TSS
    gdt_init();

    //now the basic virtual memory map(see memalyout.h) is established.
    //check the correctness of the basic virtual memory map.
    check_boot_pgdir();

    print_pgdir();

    kmalloc_init();

}

//get_pte - get pte and return the kernel virtual address of this pte for la
//        - if the PT contians this pte didn't exist, alloc a page for PT
// parameter:
//  pgdir:  the kernel virtual base address of PDT
//  la:     the linear address need to map
//  create: a logical value to decide if alloc a page for PT
// return vaule: the kernel virtual address of this pte
pte_t *
get_pte(pde_t *pgdir, uintptr_t la, bool create) {
    /* LAB2 EXERCISE 2: YOUR CODE
     *
     * If you need to visit a physical address, please use KADDR()
     * please read pmm.h for useful macros
     *
     * Maybe you want help comment, BELOW comments can help you finish the code
     *
     * Some Useful MACROs and DEFINEs, you can use them in below implementation.
     * MACROs or Functions:
     *   PDX(la) = the index of page directory entry of VIRTUAL ADDRESS la.
     *   KADDR(pa) : takes a physical address and returns the corresponding kernel virtual add
ress.
     *   set_page_ref(page,1) : means the page be referenced by one time
     *   page2pa(page): get the physical address of memory which this (struct Page *) page  ma
nages
     *   struct Page * alloc_page() : allocation a page
     *   memset(void *s, char c, size_t n) : sets the first n bytes of the memory area pointed
 by s
     *                                       to the specified value c.
     * DEFINEs:
     *   PTE_P           0x001                   // page table/directory entry flags bit : Pre
sent
     *   PTE_W           0x002                   // page table/directory entry flags bit : Wri
teable
     *   PTE_U           0x004                   // page table/directory entry flags bit : Use
r can access
     */
```

```
#if 0
    pde_t *pdep = NULL;    // (1) find page directory entry
    if (0) {               // (2) check if entry is not present
                           // (3) check if creating is needed, then alloc page for page table
                           // CAUTION: this page is used for page table, not for common data pa
ge
                           // (4) set page reference
        uintptr_t pa = 0; // (5) get linear address of page
                           // (6) clear page content using memset
                           // (7) set page directory entry's permission
    }
    return NULL;           // (8) return page table entry
#endif
    pde_t *pdep = &pgdir[PDX(la)];
    if (!(*pdep & PTE_P)) {
        struct Page *page;
        if (!create || (page = alloc_page()) == NULL) {
            return NULL;
        }
        set_page_ref(page, 1);
        uintptr_t pa = page2pa(page);
        memset(KADDR(pa), 0, PGSIZE);
        *pdep = pa | PTE_U | PTE_W | PTE_P;
    }
    return &((pte_t *)KADDR(PDE_ADDR(*pdep)))[PTX(la)];
}

//get_page - get related Page struct for linear address la using PDT pgdir
struct Page *
get_page(pde_t *pgdir, uintptr_t la, pte_t **ptep_store) {
    pte_t *ptep = get_pte(pgdir, la, 0);
    if (ptep_store != NULL) {
        *ptep_store = ptep;
    }
    if (ptep != NULL && *ptep & PTE_P) {
        return pte2page(*ptep);
    }
    return NULL;
}

//page_remove_pte - free an Page sturct which is related linear address la
//                - and clean(invalidate) pte which is related linear address la
//note: PT is changed, so the TLB need to be invalidate
static inline void
page_remove_pte(pde_t *pgdir, uintptr_t la, pte_t *ptep) {
    /* LAB2 EXERCISE 3: YOUR CODE
     *
     * Please check if ptep is valid, and tlb must be manually updated if mapping is updated
     *
     * Maybe you want help comment, BELOW comments can help you finish the code
     *
     * Some Useful MACROs and DEFINEs, you can use them in below implementation.
     * MACROs or Functions:
     *   struct Page *page pte2page(*ptep): get the according page from the value of a ptep
     *   free_page : free a page
     *   page_ref_dec(page) : decrease page->ref. NOTICE: ff page->ref == 0 , then this page s
hould be free.
     *   tlb_invalidate(pde_t *pgdir, uintptr_t la) : Invalidate a TLB entry, but only if the
page tables being
     *                        edited are the ones currently in use by the processor.
     * DEFINEs:
     *   PTE_P           0x001                   // page table/directory entry flags bit : Pre
sent
     */
#if 0
```

```
    if (0) {                        //(1) check if page directory is present
        struct Page *page = NULL;   //(2) find corresponding page to pte
                                    //(3) decrease page reference
                                    //(4) and free this page when page reference reachs 0
                                    //(5) clear second page table entry
                                    //(6) flush tlb
    }
#endif
    if (*ptep & PTE_P) {
        struct Page *page = pte2page(*ptep);
        if (page_ref_dec(page) == 0) {
            free_page(page);
        }
        *ptep = 0;
        tlb_invalidate(pgdir, la);
    }
}

void
unmap_range(pde_t *pgdir, uintptr_t start, uintptr_t end) {
    assert(start % PGSIZE == 0 && end % PGSIZE == 0);
    assert(USER_ACCESS(start, end));

    do {
        pte_t *ptep = get_pte(pgdir, start, 0);
        if (ptep == NULL) {
            start = ROUNDDOWN(start + PTSIZE, PTSIZE);
            continue ;
        }
        if (*ptep != 0) {
            page_remove_pte(pgdir, start, ptep);
        }
        start += PGSIZE;
    } while (start != 0 && start < end);
}

void
exit_range(pde_t *pgdir, uintptr_t start, uintptr_t end) {
    assert(start % PGSIZE == 0 && end % PGSIZE == 0);
    assert(USER_ACCESS(start, end));

    start = ROUNDDOWN(start, PTSIZE);
    do {
        int pde_idx = PDX(start);
        if (pgdir[pde_idx] & PTE_P) {
            free_page(pde2page(pgdir[pde_idx]));
            pgdir[pde_idx] = 0;
        }
        start += PTSIZE;
    } while (start != 0 && start < end);
}
/* copy_range - copy content of memory (start, end) of one process A to another process B
 * @to:    the addr of process B's Page Directory
 * @from:  the addr of process A's Page Directory
 * @share: flags to indicate to dup OR share. We just use dup method, so it didn't be used.
 *
 * CALL GRAPH: copy_mm-->dup_mmap-->copy_range
 */
int
copy_range(pde_t *to, pde_t *from, uintptr_t start, uintptr_t end, bool share) {
    assert(start % PGSIZE == 0 && end % PGSIZE == 0);
    assert(USER_ACCESS(start, end));
    // copy content by page unit.
    do {
        //call get_pte to find process A's pte according to the addr start
```

```
        pte_t *ptep = get_pte(from, start, 0), *nptep;
        if (ptep == NULL) {
            start = ROUNDDOWN(start + PTSIZE, PTSIZE);
            continue ;
        }
        //call get_pte to find process B’s pte according to the addr start. If pte is NULL, ju
st alloc a PT
        if (*ptep & PTE_P) {
            if ((nptep = get_pte(to, start, 1)) == NULL) {
                return -E_NO_MEM;
            }
        uint32_t perm = (*ptep & PTE_USER);
        //get page from ptep
        struct Page *page = pte2page(*ptep);
        // alloc a page for process B
        struct Page *npage=alloc_page();
        assert(page!=NULL);
        assert(npage!=NULL);
        int ret=0;
        /* LAB5:EXERCISE2 YOUR CODE
         * replicate content of page to npage, build the map of phy addr of nage with the line
ar addr start
         *
         * Some Useful MACROs and DEFINEs, you can use them in below implementation.
         * MACROs or Functions:
         *    page2kva(struct Page *page): return the kernel vritual addr of memory which page
 managed (SEE pmm.h)
         *    page_insert: build the map of phy addr of an Page with the linear addr la
         *    memcpy: typical memory copy function
         *
         * (1) find src_kvaddr: the kernel virtual address of page
         * (2) find dst_kvaddr: the kernel virtual address of npage
         * (3) memory copy from src_kvaddr to dst_kvaddr, size is PGSIZE
         * (4) build the map of phy addr of  nage with the linear addr start
         */
        void * kva_src = page2kva(page);
        void * kva_dst = page2kva(npage);

        memcpy(kva_dst, kva_src, PGSIZE);

        ret = page_insert(to, npage, start, perm);
        assert(ret == 0);
        }
        start += PGSIZE;
    } while (start != 0 && start < end);
    return 0;
}

//page_remove - free an Page which is related linear address la and has an validated pte
void
page_remove(pde_t *pgdir, uintptr_t la) {
    pte_t *ptep = get_pte(pgdir, la, 0);
    if (ptep != NULL) {
        page_remove_pte(pgdir, la, ptep);
    }
}

//page_insert - build the map of phy addr of an Page with the linear addr la
// paramemters:
//  pgdir: the kernel virtual base address of PDT
//  page:  the Page which need to map
//  la:    the linear address need to map
//  perm:  the permission of this Page which is setted in related pte
// return value: always 0
//note: PT is changed, so the TLB need to be invalidate
```

```
int
page_insert(pde_t *pgdir, struct Page *page, uintptr_t la, uint32_t perm) {
    pte_t *ptep = get_pte(pgdir, la, 1);
    if (ptep == NULL) {
        return -E_NO_MEM;
    }
    page_ref_inc(page);
    if (*ptep & PTE_P) {
        struct Page *p = pte2page(*ptep);
        if (p == page) {
            page_ref_dec(page);
        }
        else {
            page_remove_pte(pgdir, la, ptep);
        }
    }
    *ptep = page2pa(page) | PTE_P | perm;
    tlb_invalidate(pgdir, la);
    return 0;
}

// invalidate a TLB entry, but only if the page tables being
// edited are the ones currently in use by the processor.
void
tlb_invalidate(pde_t *pgdir, uintptr_t la) {
    if (rcr3() == PADDR(pgdir)) {
        invlpg((void *)la);
    }
}

// pgdir_alloc_page - call alloc_page & page_insert functions to
//                  - allocate a page size memory & setup an addr map
//                  - pa<->la with linear address la and the PDT pgdir
struct Page *
pgdir_alloc_page(pde_t *pgdir, uintptr_t la, uint32_t perm) {
    struct Page *page = alloc_page();
    if (page != NULL) {
        if (page_insert(pgdir, page, la, perm) != 0) {
            free_page(page);
            return NULL;
        }
        if (swap_init_ok){
            if(check_mm_struct!=NULL) {
                swap_map_swappable(check_mm_struct, la, page, 0);
                page->pra_vaddr=la;
                assert(page_ref(page) == 1);
                //cprintf("get No. %d  page: pra_vaddr %x, pra_link.prev %x, pra_link_next %x
in pgdir_alloc_page\n", (page-pages), page->pra_vaddr,page->pra_page_link.prev, page->pra_page
_link.next);
            }
            else  {  //now current is existed, should fix it in the future
                //swap_map_swappable(current->mm, la, page, 0);
                //page->pra_vaddr=la;
                //assert(page_ref(page) == 1);
                //panic("pgdir_alloc_page: no pages. now current is existed, should fix it in
the future\n");
            }
        }

    }

    return page;
}

static void
```

```
check_alloc_page(void) {
    pmm_manager->check();
    cprintf("check_alloc_page() succeeded!\n");
}

static void
check_pgdir(void) {
    assert(npage <= KMEMSIZE / PGSIZE);
    assert(boot_pgdir != NULL && (uint32_t)PGOFF(boot_pgdir) == 0);
    assert(get_page(boot_pgdir, 0x0, NULL) == NULL);

    struct Page *p1, *p2;
    p1 = alloc_page();
    assert(page_insert(boot_pgdir, p1, 0x0, 0) == 0);

    pte_t *ptep;
    assert((ptep = get_pte(boot_pgdir, 0x0, 0)) != NULL);
    assert(pte2page(*ptep) == p1);
    assert(page_ref(p1) == 1);

    ptep = &((pte_t *)KADDR(PDE_ADDR(boot_pgdir[0])))[1];
    assert(get_pte(boot_pgdir, PGSIZE, 0) == ptep);

    p2 = alloc_page();
    assert(page_insert(boot_pgdir, p2, PGSIZE, PTE_U | PTE_W) == 0);
    assert((ptep = get_pte(boot_pgdir, PGSIZE, 0)) != NULL);
    assert(*ptep & PTE_U);
    assert(*ptep & PTE_W);
    assert(boot_pgdir[0] & PTE_U);
    assert(page_ref(p2) == 1);

    assert(page_insert(boot_pgdir, p1, PGSIZE, 0) == 0);
    assert(page_ref(p1) == 2);
    assert(page_ref(p2) == 0);
    assert((ptep = get_pte(boot_pgdir, PGSIZE, 0)) != NULL);
    assert(pte2page(*ptep) == p1);
    assert((*ptep & PTE_U) == 0);

    page_remove(boot_pgdir, 0x0);
    assert(page_ref(p1) == 1);
    assert(page_ref(p2) == 0);

    page_remove(boot_pgdir, PGSIZE);
    assert(page_ref(p1) == 0);
    assert(page_ref(p2) == 0);

    assert(page_ref(pde2page(boot_pgdir[0])) == 1);
    free_page(pde2page(boot_pgdir[0]));
    boot_pgdir[0] = 0;

    cprintf("check_pgdir() succeeded!\n");
}

static void
check_boot_pgdir(void) {
    pte_t *ptep;
    int i;
    for (i = 0; i < npage; i += PGSIZE) {
        assert((ptep = get_pte(boot_pgdir, (uintptr_t)KADDR(i), 0)) != NULL);
        assert(PTE_ADDR(*ptep) == i);
    }

    assert(PDE_ADDR(boot_pgdir[PDX(VPT)]) == PADDR(boot_pgdir));

    assert(boot_pgdir[0] == 0);
```

```
    struct Page *p;
    p = alloc_page();
    assert(page_insert(boot_pgdir, p, 0x100, PTE_W) == 0);
    assert(page_ref(p) == 1);
    assert(page_insert(boot_pgdir, p, 0x100 + PGSIZE, PTE_W) == 0);
    assert(page_ref(p) == 2);

    const char *str = "ucore: Hello world!!";
    strcpy((void *)0x100, str);
    assert(strcmp((void *)0x100, (void *)(0x100 + PGSIZE)) == 0);

    *(char *)(page2kva(p) + 0x100) = '\0';
    assert(strlen((const char *)0x100) == 0);

    free_page(p);
    free_page(pde2page(boot_pgdir[0]));
    boot_pgdir[0] = 0;

    cprintf("check_boot_pgdir() succeeded!\n");
}

//perm2str - use string 'u,r,w,-' to present the permission
static const char *
perm2str(int perm) {
    static char str[4];
    str[0] = (perm & PTE_U) ? 'u' : '-';
    str[1] = 'r';
    str[2] = (perm & PTE_W) ? 'w' : '-';
    str[3] = '\0';
    return str;
}

//get_pgtable_items - In [left, right] range of PDT or PT, find a continuous linear addr space
//                  - (left_store*X_SIZE~right_store*X_SIZE) for PDT or PT
//                  - X_SIZE=PTSIZE=4M, if PDT; X_SIZE=PGSIZE=4K, if PT
// paramemters:
//  left:        no use ???
//  right:       the high side of table's range
//  start:       the low side of table's range
//  table:       the beginning addr of table
//  left_store:  the pointer of the high side of table's next range
//  right_store: the pointer of the low side of table's next range
// return value: 0 - not a invalid item range, perm - a valid item range with perm permission
static int
get_pgtable_items(size_t left, size_t right, size_t start, uintptr_t *table, size_t *left_stor
e, size_t *right_store) {
    if (start >= right) {
        return 0;
    }
    while (start < right && !(table[start] & PTE_P)) {
        start ++;
    }
    if (start < right) {
        if (left_store != NULL) {
            *left_store = start;
        }
        int perm = (table[start ++] & PTE_USER);
        while (start < right && (table[start] & PTE_USER) == perm) {
            start ++;
        }
        if (right_store != NULL) {
            *right_store = start;
        }
        return perm;
```

```
    }
    return 0;
}

//print_pgdir - print the PDT&PT
void
print_pgdir(void) {
    cprintf("-------------------- BEGIN --------------------\n");
    size_t left, right = 0, perm;
    while ((perm = get_pgtable_items(0, NPDEENTRY, right, vpd, &left, &right)) != 0) {
        cprintf("PDE(%03x) %08x-%08x %08x %s\n", right - left,
                left * PTSIZE, right * PTSIZE, (right - left) * PTSIZE, perm2str(perm));
        size_t l, r = left * NPTEENTRY;
        while ((perm = get_pgtable_items(left * NPTEENTRY, right * NPTEENTRY, r, vpt, &l, &r))
 != 0) {
            cprintf("  |-- PTE(%05x) %08x-%08x %08x %s\n", r - l,
                    l * PGSIZE, r * PGSIZE, (r - l) * PGSIZE, perm2str(perm));
        }
    }
    cprintf("--------------------- END ---------------------\n");
}
[ucore/lab8_result/kern/mm/default_pmm.c] ++++++++++++++++++++++++++++++
#include <pmm.h>
#include <list.h>
#include <string.h>
#include <default_pmm.h>

/*  In the First Fit algorithm, the allocator keeps a list of free blocks
 * (known as the free list). Once receiving a allocation request for memory,
 * it scans along the list for the first block that is large enough to satisfy
 * the request. If the chosen block is significantly larger than requested, it
 * is usually splitted, and the remainder will be added into the list as
 * another free block.
 *  Please refer to Page 196~198, Section 8.2 of Yan Wei Min's Chinese book
 * "Data Structure -- C programming language".
*/
// LAB2 EXERCISE 1: YOUR CODE
// you should rewrite functions: `default_init`, `default_init_memmap`,
// `default_alloc_pages`, `default_free_pages`.
/*
 * Details of FFMA
 * (1) Preparation:
 *  In order to implement the First-Fit Memory Allocation (FFMA), we should
 * manage the free memory blocks using a list. The struct `free_area_t` is used
 * for the management of free memory blocks.
 *  First, you should get familiar with the struct `list` in list.h. Struct
 * `list` is a simple doubly linked list implementation. You should know how to
 * USE `list_init`, `list_add`(`list_add_after`), `list_add_before`, `list_del`,
 * `list_next`, `list_prev`.
 *  There's a tricky method that is to transform a general `list` struct to a
 * special struct (such as struct `page`), using the following MACROs: `le2page`
 * (in memlayout.h), (and in future labs: `le2vma` (in vmm.h), `le2proc` (in
 * proc.h), etc).
 * (2) `default_init`:
 *  You can reuse the demo `default_init` function to initialize the `free_list`
 * and set `nr_free` to 0. `free_list` is used to record the free memory blocks.
 * `nr_free` is the total number of the free memory blocks.
 * (3) `default_init_memmap`:
 *  CALL GRAPH: `kern_init` --> `pmm_init` --> `page_init` --> `init_memmap` -->
 * `pmm_manager` --> `init_memmap`.
 *  This function is used to initialize a free block (with parameter `addr_base`,
 * `page_number`). In order to initialize a free block, firstly, you should
 * initialize each page (defined in memlayout.h) in this free block. This
 * procedure includes:
 *  - Setting the bit `PG_property` of `p->flags`, which means this page is
```

```
 * valid. P.S. In function `pmm_init` (in pmm.c), the bit `PG_reserved` of
 * `p->flags` is already set.
 *  - If this page is free and is not the first page of a free block,
 * `p->property` should be set to 0.
 *  - If this page is free and is the first page of a free block, `p->property`
 * should be set to be the total number of pages in the block.
 *  - `p->ref` should be 0, because now `p` is free and has no reference.
 *  After that, We can use `p->page_link` to link this page into `free_list`.
 * (e.g.: `list_add_before(&free_list, &(p->page_link));` )
 *  Finally, we should update the sum of the free memory blocks: `nr_free += n`.
 * (4) `default_alloc_pages`:
 *  Search for the first free block (block size >= n) in the free list and reszie
 * the block found, returning the address of this block as the address required by
 * `malloc`.
 *  (4.1)
 *      So you should search the free list like this:
 *          list_entry_t le = &free_list;
 *          while((le=list_next(le)) != &free_list) {
 *          ...
 *      (4.1.1)
 *          In the while loop, get the struct `page` and check if `p->property`
 *      (recording the num of free pages in this block) >= n.
 *              struct Page *p = le2page(le, page_link);
 *              if(p->property >= n){ ...
 *      (4.1.2)
 *          If we find this `p`, it means we've found a free block with its size
 *      >= n, whose first `n` pages can be malloced. Some flag bits of this page
 *      should be set as the following: `PG_reserved = 1`, `PG_property = 0`.
 *      Then, unlink the pages from `free_list`.
 *          (4.1.2.1)
 *              If `p->property > n`, we should re-calculate number of the rest
 *          pages of this free block. (e.g.: `le2page(le,page_link))->property
 *          = p->property - n;`)
 *          (4.1.3)
 *              Re-caluclate `nr_free` (number of the the rest of all free block).
 *          (4.1.4)
 *              return `p`.
 *      (4.2)
 *          If we can not find a free block with its size >=n, then return NULL.
 * (5) `default_free_pages`:
 *  re-link the pages into the free list, and may merge small free blocks into
 * the big ones.
 *  (5.1)
 *      According to the base address of the withdrawed blocks, search the free
 *  list for its correct position (with address from low to high), and insert
 *  the pages. (May use `list_next`, `le2page`, `list_add_before`)
 *  (5.2)
 *      Reset the fields of the pages, such as `p->ref` and `p->flags` (PageProperty)
 *  (5.3)
 *      Try to merge blocks at lower or higher addresses. Notice: This should
 *  change some pages' `p->property` correctly.
 */
free_area_t free_area;

#define free_list (free_area.free_list)
#define nr_free (free_area.nr_free)

static void
default_init(void) {
    list_init(&free_list);
    nr_free = 0;
}

static void
default_init_memmap(struct Page *base, size_t n) {
```

```
    assert(n > 0);
    struct Page *p = base;
    for (; p != base + n; p ++) {
        assert(PageReserved(p));
        p->flags = p->property = 0;
        set_page_ref(p, 0);
    }
    base->property = n;
    SetPageProperty(base);
    nr_free += n;
    list_add_before(&free_list, &(base->page_link));
}

static struct Page *
default_alloc_pages(size_t n) {
    assert(n > 0);
    if (n > nr_free) {
        return NULL;
    }
    struct Page *page = NULL;
    list_entry_t *le = &free_list;
    // TODO: optimize (next-fit)
    while ((le = list_next(le)) != &free_list) {
        struct Page *p = le2page(le, page_link);
        if (p->property >= n) {
            page = p;
            break;
        }
    }
    if (page != NULL) {
        if (page->property > n) {
            struct Page *p = page + n;
            p->property = page->property - n;
            SetPageProperty(p);
            list_add_after(&(page->page_link), &(p->page_link));
        }
        list_del(&(page->page_link));
        nr_free -= n;
        ClearPageProperty(page);
    }
    return page;
}

static void
default_free_pages(struct Page *base, size_t n) {
    assert(n > 0);
    struct Page *p = base;
    for (; p != base + n; p ++) {
        assert(!PageReserved(p) && !PageProperty(p));
        p->flags = 0;
        set_page_ref(p, 0);
    }
    base->property = n;
    SetPageProperty(base);
    list_entry_t *le = list_next(&free_list);
    while (le != &free_list) {
        p = le2page(le, page_link);
        le = list_next(le);
        // TODO: optimize
        if (base + base->property == p) {
            base->property += p->property;
            ClearPageProperty(p);
            list_del(&(p->page_link));
        }
        else if (p + p->property == base) {
```

```
            p->property += base->property;
            ClearPageProperty(base);
            base = p;
            list_del(&(p->page_link));
        }
    }
    nr_free += n;
    le = list_next(&free_list);
    while (le != &free_list) {
        p = le2page(le, page_link);
        if (base + base->property <= p) {
            assert(base + base->property != p);
            break;
        }
        le = list_next(le);
    }
    list_add_before(le, &(base->page_link));
}

static size_t
default_nr_free_pages(void) {
    return nr_free;
}

static void
basic_check(void) {
    struct Page *p0, *p1, *p2;
    p0 = p1 = p2 = NULL;
    assert((p0 = alloc_page()) != NULL);
    assert((p1 = alloc_page()) != NULL);
    assert((p2 = alloc_page()) != NULL);

    assert(p0 != p1 && p0 != p2 && p1 != p2);
    assert(page_ref(p0) == 0 && page_ref(p1) == 0 && page_ref(p2) == 0);

    assert(page2pa(p0) < npage * PGSIZE);
    assert(page2pa(p1) < npage * PGSIZE);
    assert(page2pa(p2) < npage * PGSIZE);

    list_entry_t free_list_store = free_list;
    list_init(&free_list);
    assert(list_empty(&free_list));

    unsigned int nr_free_store = nr_free;
    nr_free = 0;

    assert(alloc_page() == NULL);

    free_page(p0);
    free_page(p1);
    free_page(p2);
    assert(nr_free == 3);

    assert((p0 = alloc_page()) != NULL);
    assert((p1 = alloc_page()) != NULL);
    assert((p2 = alloc_page()) != NULL);

    assert(alloc_page() == NULL);

    free_page(p0);
    assert(!list_empty(&free_list));

    struct Page *p;
    assert((p = alloc_page()) == p0);
    assert(alloc_page() == NULL);
```

```
    assert(nr_free == 0);
    free_list = free_list_store;
    nr_free = nr_free_store;

    free_page(p);
    free_page(p1);
    free_page(p2);
}

// LAB2: below code is used to check the first fit allocation algorithm (your EXERCISE 1)
// NOTICE: You SHOULD NOT CHANGE basic_check, default_check functions!
static void
default_check(void) {
    int count = 0, total = 0;
    list_entry_t *le = &free_list;
    while ((le = list_next(le)) != &free_list) {
        struct Page *p = le2page(le, page_link);
        assert(PageProperty(p));
        count ++, total += p->property;
    }
    assert(total == nr_free_pages());

    basic_check();

    struct Page *p0 = alloc_pages(5), *p1, *p2;
    assert(p0 != NULL);
    assert(!PageProperty(p0));

    list_entry_t free_list_store = free_list;
    list_init(&free_list);
    assert(list_empty(&free_list));
    assert(alloc_page() == NULL);

    unsigned int nr_free_store = nr_free;
    nr_free = 0;

    free_pages(p0 + 2, 3);
    assert(alloc_pages(4) == NULL);
    assert(PageProperty(p0 + 2) && p0[2].property == 3);
    assert((p1 = alloc_pages(3)) != NULL);
    assert(alloc_page() == NULL);
    assert(p0 + 2 == p1);

    p2 = p0 + 1;
    free_page(p0);
    free_pages(p1, 3);
    assert(PageProperty(p0) && p0->property == 1);
    assert(PageProperty(p1) && p1->property == 3);

    assert((p0 = alloc_page()) == p2 - 1);
    free_page(p0);
    assert((p0 = alloc_pages(2)) == p2 + 1);

    free_pages(p0, 2);
    free_page(p2);

    assert((p0 = alloc_pages(5)) != NULL);
    assert(alloc_page() == NULL);

    assert(nr_free == 0);
    nr_free = nr_free_store;

    free_list = free_list_store;
    free_pages(p0, 5);
```

```
    le = &free_list;
    while ((le = list_next(le)) != &free_list) {
        struct Page *p = le2page(le, page_link);
        count --, total -= p->property;
    }
    assert(count == 0);
    assert(total == 0);
}

const struct pmm_manager default_pmm_manager = {
    .name = "default_pmm_manager",
    .init = default_init,
    .init_memmap = default_init_memmap,
    .alloc_pages = default_alloc_pages,
    .free_pages = default_free_pages,
    .nr_free_pages = default_nr_free_pages,
    .check = default_check,
};
[ucore/lab8_result/kern/mm/vmm.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_VMM_H__
#define __KERN_MM_VMM_H__

#include <defs.h>
#include <list.h>
#include <memlayout.h>
#include <sync.h>
#include <proc.h>
#include <sem.h>

//pre define
struct mm_struct;

// the virtual continuous memory area(vma)
struct vma_struct {
    struct mm_struct *vm_mm; // the set of vma using the same PDT
    uintptr_t vm_start;      //    start addr of vma
    uintptr_t vm_end;        // end addr of vma
    uint32_t vm_flags;       // flags of vma
    list_entry_t list_link;  // linear list link which sorted by start addr of vma
};

#define le2vma(le, member)                  \
    to_struct((le), struct vma_struct, member)

#define VM_READ                 0x00000001
#define VM_WRITE                0x00000002
#define VM_EXEC                 0x00000004
#define VM_STACK                0x00000008

// the control struct for a set of vma using the same PDT
struct mm_struct {
    list_entry_t mmap_list;        // linear list link which sorted by start addr of vma
    struct vma_struct *mmap_cache; // current accessed vma, used for speed purpose
    pde_t *pgdir;                  // the PDT of these vma
    int map_count;                 // the count of these vma
    void *sm_priv;                 // the private data for swap manager
    int mm_count;                  // the number ofprocess which shared the mm
    semaphore_t mm_sem;            // mutex for using dup_mmap fun to duplicat the mm
    int locked_by;                 // the lock owner process's pid

};

struct vma_struct *find_vma(struct mm_struct *mm, uintptr_t addr);
struct vma_struct *vma_create(uintptr_t vm_start, uintptr_t vm_end, uint32_t vm_flags);
```

```
void insert_vma_struct(struct mm_struct *mm, struct vma_struct *vma);

struct mm_struct *mm_create(void);
void mm_destroy(struct mm_struct *mm);

void vmm_init(void);
int mm_map(struct mm_struct *mm, uintptr_t addr, size_t len, uint32_t vm_flags,
           struct vma_struct **vma_store);
int do_pgfault(struct mm_struct *mm, uint32_t error_code, uintptr_t addr);

int mm_unmap(struct mm_struct *mm, uintptr_t addr, size_t len);
int dup_mmap(struct mm_struct *to, struct mm_struct *from);
void exit_mmap(struct mm_struct *mm);
uintptr_t get_unmapped_area(struct mm_struct *mm, size_t len);
int mm_brk(struct mm_struct *mm, uintptr_t addr, size_t len);

extern volatile unsigned int pgfault_num;
extern struct mm_struct *check_mm_struct;

bool user_mem_check(struct mm_struct *mm, uintptr_t start, size_t len, bool write);
bool copy_from_user(struct mm_struct *mm, void *dst, const void *src, size_t len, bool writabl
e);
bool copy_to_user(struct mm_struct *mm, void *dst, const void *src, size_t len);
bool copy_string(struct mm_struct *mm, char *dst, const char *src, size_t maxn);

static inline int
mm_count(struct mm_struct *mm) {
    return mm->mm_count;
}

static inline void
set_mm_count(struct mm_struct *mm, int val) {
    mm->mm_count = val;
}

static inline int
mm_count_inc(struct mm_struct *mm) {
    mm->mm_count += 1;
    return mm->mm_count;
}

static inline int
mm_count_dec(struct mm_struct *mm) {
    mm->mm_count -= 1;
    return mm->mm_count;
}

static inline void
lock_mm(struct mm_struct *mm) {
    if (mm != NULL) {
        down(&(mm->mm_sem));
        if (current != NULL) {
            mm->locked_by = current->pid;
        }
    }
}

static inline void
unlock_mm(struct mm_struct *mm) {
    if (mm != NULL) {
        up(&(mm->mm_sem));
        mm->locked_by = 0;
    }
}
```

```
#endif /* !__KERN_MM_VMM_H__ */

[ucore/lab8_result/kern/mm/swap_fifo.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <x86.h>
#include <stdio.h>
#include <string.h>
#include <swap.h>
#include <swap_fifo.h>
#include <list.h>

/* [wikipedia]The simplest Page Replacement Algorithm(PRA) is a FIFO algorithm. The first-in,
first-out
 * page replacement algorithm is a low-overhead algorithm that requires little book-keeping on
 * the part of the operating system. The idea is obvious from the name - the operating system
 * keeps track of all the pages in memory in a queue, with the most recent arrival at the back
,
 * and the earliest arrival in front. When a page needs to be replaced, the page at the front
 * of the queue (the oldest page) is selected. While FIFO is cheap and intuitive, it performs
 * poorly in practical application. Thus, it is rarely used in its unmodified form. This
 * algorithm experiences Belady's anomaly.
 *
 * Details of FIFO PRA
 * (1) Prepare: In order to implement FIFO PRA, we should manage all swappable pages, so we ca
n
 *              link these pages into pra_list_head according the time order. At first you sho
uld
 *              be familiar to the struct list in list.h. struct list is a simple doubly linke
d list
 *              implementation. You should know howto USE: list_init, list_add(list_add_after)
,
 *              list_add_before, list_del, list_next, list_prev. Another tricky method is to t
ransform
 *              a general list struct to a special struct (such as struct page). You can find
some MACRO:
 *              le2page (in memlayout.h), (in future labs: le2vma (in vmm.h), le2proc (in proc
.h),etc.
 */

list_entry_t pra_list_head;
/*
 * (2) _fifo_init_mm: init pra_list_head and let  mm->sm_priv point to the addr of pra_list_he
ad.
 *              Now, From the memory control struct mm_struct, we can access FIFO PRA
 */
static int
_fifo_init_mm(struct mm_struct *mm)
{
     list_init(&pra_list_head);
     mm->sm_priv = &pra_list_head;
     //cprintf(" mm->sm_priv %x in fifo_init_mm\n",mm->sm_priv);
     return 0;
}
/*
 * (3)_fifo_map_swappable: According FIFO PRA, we should link the most recent arrival page at
the back of pra_list_head qeueue
 */
static int
_fifo_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    list_entry_t *entry=&(page->pra_page_link);

    assert(entry != NULL && head != NULL);
    //record the page access situlation
```

```
    /*LAB3 EXERCISE 2: YOUR CODE*/
    //(1)link the most recent arrival page at the back of the pra_list_head qeueue.
    list_add(head, entry);
    return 0;
}
/*
 *  (4)_fifo_swap_out_victim: According FIFO PRA, we should unlink the  earliest arrival page
in front of pra_list_head qeueue,
 *                            then assign the value of *ptr_page to the addr of this page.
 */
static int
_fifo_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int in_tick)
{
     list_entry_t *head=(list_entry_t*) mm->sm_priv;
         assert(head != NULL);
     assert(in_tick==0);
     /* Select the victim */
     /*LAB3 EXERCISE 2: YOUR CODE*/
     //(1)  unlink the  earliest arrival page in front of pra_list_head qeueue
     //(2)  assign the value of *ptr_page to the addr of this page
     /* Select the tail */
     list_entry_t *le = head->prev;
     assert(head!=le);
     struct Page *p = le2page(le, pra_page_link);
     list_del(le);
     assert(p !=NULL);
     *ptr_page = p;
     return 0;
}

static int
_fifo_check_swap(void) {
    cprintf("write Virt Page c in fifo_check_swap\n");
    *(unsigned char *)0x3000 = 0x0c;
    assert(pgfault_num==4);
    cprintf("write Virt Page a in fifo_check_swap\n");
    *(unsigned char *)0x1000 = 0x0a;
    assert(pgfault_num==4);
    cprintf("write Virt Page d in fifo_check_swap\n");
    *(unsigned char *)0x4000 = 0x0d;
    assert(pgfault_num==4);
    cprintf("write Virt Page b in fifo_check_swap\n");
    *(unsigned char *)0x2000 = 0x0b;
    assert(pgfault_num==4);
    cprintf("write Virt Page e in fifo_check_swap\n");
    *(unsigned char *)0x5000 = 0x0e;
    assert(pgfault_num==5);
    cprintf("write Virt Page b in fifo_check_swap\n");
    *(unsigned char *)0x2000 = 0x0b;
    assert(pgfault_num==5);
    cprintf("write Virt Page a in fifo_check_swap\n");
    *(unsigned char *)0x1000 = 0x0a;
    assert(pgfault_num==6);
    cprintf("write Virt Page b in fifo_check_swap\n");
    *(unsigned char *)0x2000 = 0x0b;
    assert(pgfault_num==7);
    cprintf("write Virt Page c in fifo_check_swap\n");
    *(unsigned char *)0x3000 = 0x0c;
    assert(pgfault_num==8);
    cprintf("write Virt Page d in fifo_check_swap\n");
    *(unsigned char *)0x4000 = 0x0d;
    assert(pgfault_num==9);
    cprintf("write Virt Page e in fifo_check_swap\n");
    *(unsigned char *)0x5000 = 0x0e;
    assert(pgfault_num==10);
```

```
    cprintf("write Virt Page a in fifo_check_swap\n");
    assert(*(unsigned char *)0x1000 == 0x0a);
    *(unsigned char *)0x1000 = 0x0a;
    assert(pgfault_num==11);
    return 0;
}


static int
_fifo_init(void)
{
    return 0;
}

static int
_fifo_set_unswappable(struct mm_struct *mm, uintptr_t addr)
{
    return 0;
}

static int
_fifo_tick_event(struct mm_struct *mm)
{ return 0; }


struct swap_manager swap_manager_fifo =
{
     .name            = "fifo swap manager",
     .init            = &_fifo_init,
     .init_mm         = &_fifo_init_mm,
     .tick_event      = &_fifo_tick_event,
     .map_swappable   = &_fifo_map_swappable,
     .set_unswappable = &_fifo_set_unswappable,
     .swap_out_victim = &_fifo_swap_out_victim,
     .check_swap      = &_fifo_check_swap,
};
[ucore/lab8_result/kern/mm/memlayout.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_MEMLAYOUT_H__
#define __KERN_MM_MEMLAYOUT_H__

/* This file contains the definitions for memory management in our OS. */

/* global segment number */
#define SEG_KTEXT   1
#define SEG_KDATA   2
#define SEG_UTEXT   3
#define SEG_UDATA   4
#define SEG_TSS     5

/* global descrptor numbers */
#define GD_KTEXT    ((SEG_KTEXT) << 3)      // kernel text
#define GD_KDATA    ((SEG_KDATA) << 3)      // kernel data
#define GD_UTEXT    ((SEG_UTEXT) << 3)      // user text
#define GD_UDATA    ((SEG_UDATA) << 3)      // user data
#define GD_TSS      ((SEG_TSS) << 3)        // task segment selector

#define DPL_KERNEL  (0)
#define DPL_USER    (3)

#define KERNEL_CS   ((GD_KTEXT) | DPL_KERNEL)
#define KERNEL_DS   ((GD_KDATA) | DPL_KERNEL)
#define USER_CS     ((GD_UTEXT) | DPL_USER)
#define USER_DS     ((GD_UDATA) | DPL_USER)

/* *
```

```
 * Virtual memory map:                                   Permissions
 *                                                       kernel/user
 *
 *     4G ------------------> +---------------------------------+
 *                            |                                 |
 *                            |         Empty Memory (*)        |
 *                            |                                 |
 *                            +---------------------------------+ 0xFB000000
 *                            |   Cur. Page Table (Kern, RW)    | RW/-- PTSIZE
 *     VPT ----------------->  +---------------------------------+ 0xFAC00000
 *                            |        Invalid Memory (*)       | --/--
 *     KERNTOP ------------>  +---------------------------------+ 0xF8000000
 *                            |                                 |
 *                            |    Remapped Physical Memory     | RW/-- KMEMSIZE
 *                            |                                 |
 *     KERNBASE ----------->  +---------------------------------+ 0xC0000000
 *                            |        Invalid Memory (*)       | --/--
 *     USERTOP ------------>  +---------------------------------+ 0xB0000000
 *                            |           User stack            |
 *                            +---------------------------------+
 *                            |                                 |
 *                            :                                 :
 *                            |         ~~~~~~~~~~~~~~~         |
 *                            :                                 :
 *                            |                                 |
 *                            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *                            |       User Program & Heap       |
 *     UTEXT --------------->  +---------------------------------+ 0x00800000
 *                            |        Invalid Memory (*)       | --/--
 *                            |  - - - - - - - - - - - - - - -  |
 *                            |    User STAB Data (optional)    |
 *     USERBASE, USTAB------> +---------------------------------+ 0x00200000
 *                            |        Invalid Memory (*)       | --/--
 *     0 ------------------>  +---------------------------------+ 0x00000000
 * (*) Note: The kernel ensures that "Invalid Memory" is *never* mapped.
 *     "Empty Memory" is normally unmapped, but user programs may map pages
 *     there if desired.
 *
 * */

/* All physical memory mapped at this address */
#define KERNBASE            0xC0000000
#define KMEMSIZE            0x38000000                  // the maximum amount of physical memo
ry
#define KERNTOP             (KERNBASE + KMEMSIZE)

/* *
 * Virtual page table. Entry PDX[VPT] in the PD (Page Directory) contains
 * a pointer to the page directory itself, thereby turning the PD into a page
 * table, which maps all the PTEs (Page Table Entry) containing the page mappings
 * for the entire virtual address space into that 4 Meg region starting at VPT.
 * */
#define VPT                 0xFAC00000

#define KSTACKPAGE          2                           // # of pages in kernel stack
#define KSTACKSIZE          (KSTACKPAGE * PGSIZE)       // sizeof kernel stack

#define USERTOP             0xB0000000
#define USTACKTOP           USERTOP
#define USTACKPAGE          256                         // # of pages in user stack
#define USTACKSIZE          (USTACKPAGE * PGSIZE)       // sizeof user stack

#define USERBASE            0x00200000
#define UTEXT               0x00800000                  // where user programs generally begin
#define USTAB               USERBASE                    // the location of the user STABS data
```

structure

```
#define USER_ACCESS(start, end)                     \
(USERBASE <= (start) && (start) < (end) && (end) <= USERTOP)

#define KERN_ACCESS(start, end)                     \
(KERNBASE <= (start) && (start) < (end) && (end) <= KERNTOP)

#ifndef __ASSEMBLER__

#include <defs.h>
#include <atomic.h>
#include <list.h>

typedef uintptr_t pte_t;
typedef uintptr_t pde_t;
typedef pte_t swap_entry_t; //the pte can also be a swap entry

// some constants for bios interrupt 15h AX = 0xE820
#define E820MAX             20      // number of entries in E820MAP
#define E820_ARM            1       // address range memory
#define E820_ARR            2       // address range reserved

struct e820map {
    int nr_map;
    struct {
        uint64_t addr;
        uint64_t size;
        uint32_t type;
    } __attribute__((packed)) map[E820MAX];
};

/* *
 * struct Page - Page descriptor structures. Each Page describes one
 * physical page. In kern/mm/pmm.h, you can find lots of useful functions
 * that convert Page to other data types, such as phyical address.
 * */
struct Page {
    int ref;                        // page frame's reference counter
    uint32_t flags;                 // array of flags that describe the status of the page fra
me
    unsigned int property;          // used in buddy system, stores the order (the X in 2^X) o
f the continuous memory block
    int zone_num;                   // used in buddy system, the No. of zone which the page be
longs to
    list_entry_t page_link;         // free list link
    list_entry_t pra_page_link;     // used for pra (page replace algorithm)
    uintptr_t pra_vaddr;            // used for pra (page replace algorithm)
};

/* Flags describing the status of a page frame */
#define PG_reserved                 0       // the page descriptor is reserved for kernel or u
nusable
#define PG_property                 1       // the member 'property' is valid

#define SetPageReserved(page)       set_bit(PG_reserved, &((page)->flags))
#define ClearPageReserved(page)     clear_bit(PG_reserved, &((page)->flags))
#define PageReserved(page)          test_bit(PG_reserved, &((page)->flags))
#define SetPageProperty(page)       set_bit(PG_property, &((page)->flags))
#define ClearPageProperty(page)     clear_bit(PG_property, &((page)->flags))
#define PageProperty(page)          test_bit(PG_property, &((page)->flags))

// convert list entry to page
#define le2page(le, member)                 \
    to_struct((le), struct Page, member)
```

```
/* free_area_t - maintains a doubly linked list to record free (unused) pages */
typedef struct {
    list_entry_t free_list;         // the list header
    unsigned int nr_free;           // # of free pages in this free list
} free_area_t;

/* for slab style kmalloc */
//#define PG_slab                     2       // page frame is included in a slab
//#define SetPageSlab(page)           set_bit(PG_slab, &((page)->flags))
//#define ClearPageSlab(page)         clear_bit(PG_slab, &((page)->flags))
//#define PageSlab(page)              test_bit(PG_slab, &((page)->flags))

#endif /* !__ASSEMBLER__ */

#endif /* !__KERN_MM_MEMLAYOUT_H__ */

[ucore/lab8_result/kern/mm/pmm.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_PMM_H__
#define __KERN_MM_PMM_H__

#include <defs.h>
#include <mmu.h>
#include <memlayout.h>
#include <atomic.h>
#include <assert.h>

// pmm_manager is a physical memory management class. A special pmm manager - XXX_pmm_manager
// only needs to implement the methods in pmm_manager class, then XXX_pmm_manager can be used
// by ucore to manage the total physical memory space.
struct pmm_manager {
    const char *name;                                 // XXX_pmm_manager's name
    void (*init)(void);                               // initialize internal description&manag
ement data structure
                                                      // (free block list, number of free bloc
k) of XXX_pmm_manager
    void (*init_memmap)(struct Page *base, size_t n); // setup description&management data str
uctcure according to
                                                      // the initial free physical memory spac
e
    struct Page *(*alloc_pages)(size_t n);            // allocate >=n pages, depend on the all
ocation algorithm
    void (*free_pages)(struct Page *base, size_t n);  // free >=n pages with "base" addr of Pa
ge descriptor structures(memlayout.h)
    size_t (*nr_free_pages)(void);                    // return the number of free pages
    void (*check)(void);                              // check the correctness of XXX_pmm_mana
ger
};

extern const struct pmm_manager *pmm_manager;
extern pde_t *boot_pgdir;
extern uintptr_t boot_cr3;

void pmm_init(void);

struct Page *alloc_pages(size_t n);
void free_pages(struct Page *base, size_t n);
size_t nr_free_pages(void);

#define alloc_page() alloc_pages(1)
#define free_page(page) free_pages(page, 1)

pte_t *get_pte(pde_t *pgdir, uintptr_t la, bool create);
struct Page *get_page(pde_t *pgdir, uintptr_t la, pte_t **ptep_store);
void page_remove(pde_t *pgdir, uintptr_t la);
```

```
int page_insert(pde_t *pgdir, struct Page *page, uintptr_t la, uint32_t perm);

void load_esp0(uintptr_t esp0);
void tlb_invalidate(pde_t *pgdir, uintptr_t la);
struct Page *pgdir_alloc_page(pde_t *pgdir, uintptr_t la, uint32_t perm);
void unmap_range(pde_t *pgdir, uintptr_t start, uintptr_t end);
void exit_range(pde_t *pgdir, uintptr_t start, uintptr_t end);
int copy_range(pde_t *to, pde_t *from, uintptr_t start, uintptr_t end, bool share);

void print_pgdir(void);

/* *
 * PADDR - takes a kernel virtual address (an address that points above KERNBASE),
 * where the machine's maximum 256MB of physical memory is mapped and returns the
 * corresponding physical address.  It panics if you pass it a non-kernel virtual address.
 * */
#define PADDR(kva) ({                                                   \
            uintptr_t __m_kva = (uintptr_t)(kva);                       \
            if (__m_kva < KERNBASE) {                                   \
                panic("PADDR called with invalid kva %08lx", __m_kva);  \
            }                                                           \
            __m_kva - KERNBASE;                                         \
        })

/* *
 * KADDR - takes a physical address and returns the corresponding kernel virtual
 * address. It panics if you pass an invalid physical address.
 * */
#define KADDR(pa) ({                                                    \
            uintptr_t __m_pa = (pa);                                    \
            size_t __m_ppn = PPN(__m_pa);                               \
            if (__m_ppn >= npage) {                                     \
                panic("KADDR called with invalid pa %08lx", __m_pa);    \
            }                                                           \
            (void *) (__m_pa + KERNBASE);                               \
        })

extern struct Page *pages;
extern size_t npage;

static inline ppn_t
page2ppn(struct Page *page) {
    return page - pages;
}

static inline uintptr_t
page2pa(struct Page *page) {
    return page2ppn(page) << PGSHIFT;
}

static inline struct Page *
pa2page(uintptr_t pa) {
    if (PPN(pa) >= npage) {
        panic("pa2page called with invalid pa");
    }
    return &pages[PPN(pa)];
}

static inline void *
page2kva(struct Page *page) {
    return KADDR(page2pa(page));
}

static inline struct Page *
kva2page(void *kva) {
```

```
    return pa2page(PADDR(kva));
}

static inline struct Page *
pte2page(pte_t pte) {
    if (!(pte & PTE_P)) {
        panic("pte2page called with invalid pte");
    }
    return pa2page(PTE_ADDR(pte));
}

static inline struct Page *
pde2page(pde_t pde) {
    return pa2page(PDE_ADDR(pde));
}

static inline int
page_ref(struct Page *page) {
    return page->ref;
}

static inline void
set_page_ref(struct Page *page, int val) {
    page->ref = val;
}

static inline int
page_ref_inc(struct Page *page) {
    page->ref += 1;
    return page->ref;
}

static inline int
page_ref_dec(struct Page *page) {
    page->ref -= 1;
    return page->ref;
}

extern char bootstack[], bootstacktop[];

#endif /* !__KERN_MM_PMM_H__ */
```

[ucore/lab8_result/kern/mm/kmalloc.c] ++++++++++++++++++++++++++++++

```
#include <defs.h>
#include <list.h>
#include <memlayout.h>
#include <assert.h>
#include <kmalloc.h>
#include <sync.h>
#include <pmm.h>
#include <stdio.h>

/*
 * SLOB Allocator: Simple List Of Blocks
 *
 * Matt Mackall <mpm@selenic.com> 12/30/03
 *
 * How SLOB works:
 *
 * The core of SLOB is a traditional K&R style heap allocator, with
 * support for returning aligned objects. The granularity of this
 * allocator is 8 bytes on x86, though it's perhaps possible to reduce
 * this to 4 if it's deemed worth the effort. The slob heap is a
 * singly-linked list of pages from __get_free_page, grown on demand
 * and allocation from the heap is currently first-fit.
```

```
 *
 * Above this is an implementation of kmalloc/kfree. Blocks returned
 * from kmalloc are 8-byte aligned and prepended with a 8-byte header.
 * If kmalloc is asked for objects of PAGE_SIZE or larger, it calls
 * __get_free_pages directly so that it can return page-aligned blocks
 * and keeps a linked list of such pages and their orders. These
 * objects are detected in kfree() by their page alignment.
 *
 * SLAB is emulated on top of SLOB by simply calling constructors and
 * destructors for every SLAB allocation. Objects are returned with
 * the 8-byte alignment unless the SLAB_MUST_HWCACHE_ALIGN flag is
 * set, in which case the low-level allocator will fragment blocks to
 * create the proper alignment. Again, objects of page-size or greater
 * are allocated by calling __get_free_pages. As SLAB objects know
 * their size, no separate size bookkeeping is necessary and there is
 * essentially no allocation space overhead.
 */


//some helper
#define spin_lock_irqsave(l, f) local_intr_save(f)
#define spin_unlock_irqrestore(l, f) local_intr_restore(f)
typedef unsigned int gfp_t;
#ifndef PAGE_SIZE
#define PAGE_SIZE PGSIZE
#endif

#ifndef L1_CACHE_BYTES
#define L1_CACHE_BYTES 64
#endif

#ifndef ALIGN
#define ALIGN(addr,size)    (((addr)+(size)-1)&(~((size)-1)))
#endif


struct slob_block {
        int units;
        struct slob_block *next;
};
typedef struct slob_block slob_t;

#define SLOB_UNIT sizeof(slob_t)
#define SLOB_UNITS(size) (((size) + SLOB_UNIT - 1)/SLOB_UNIT)
#define SLOB_ALIGN L1_CACHE_BYTES

struct bigblock {
        int order;
        void *pages;
        struct bigblock *next;
};
typedef struct bigblock bigblock_t;

static slob_t arena = { .next = &arena, .units = 1 };
static slob_t *slobfree = &arena;
static bigblock_t *bigblocks;


static void* __slob_get_free_pages(gfp_t gfp, int order)
{
  struct Page * page = alloc_pages(1 << order);
  if(!page)
    return NULL;
  return page2kva(page);
}
```

```
#define __slob_get_free_page(gfp) __slob_get_free_pages(gfp, 0)

static inline void __slob_free_pages(unsigned long kva, int order)
{
  free_pages(kva2page(kva), 1 << order);
}

static void slob_free(void *b, int size);

static void *slob_alloc(size_t size, gfp_t gfp, int align)
{
  assert( (size + SLOB_UNIT) < PAGE_SIZE );

        slob_t *prev, *cur, *aligned = 0;
        int delta = 0, units = SLOB_UNITS(size);
        unsigned long flags;

        spin_lock_irqsave(&slob_lock, flags);
        prev = slobfree;
        for (cur = prev->next; ; prev = cur, cur = cur->next) {
                if (align) {
                        aligned = (slob_t *)ALIGN((unsigned long)cur, align);
                        delta = aligned - cur;
                }
                if (cur->units >= units + delta) { /* room enough? */
                        if (delta) { /* need to fragment head to align? */
                                aligned->units = cur->units - delta;
                                aligned->next = cur->next;
                                cur->next = aligned;
                                cur->units = delta;
                                prev = cur;
                                cur = aligned;
                        }

                        if (cur->units == units) /* exact fit? */
                                prev->next = cur->next; /* unlink */
                        else { /* fragment */
                                prev->next = cur + units;
                                prev->next->units = cur->units - units;
                                prev->next->next = cur->next;
                                cur->units = units;
                        }

                        slobfree = prev;
                        spin_unlock_irqrestore(&slob_lock, flags);
                        return cur;
                }
                if (cur == slobfree) {
                        spin_unlock_irqrestore(&slob_lock, flags);

                        if (size == PAGE_SIZE) /* trying to shrink arena? */
                                return 0;

                        cur = (slob_t *)__slob_get_free_page(gfp);
                        if (!cur)
                                return 0;

                        slob_free(cur, PAGE_SIZE);
                        spin_lock_irqsave(&slob_lock, flags);
                        cur = slobfree;
                }
        }
}
```

```
static void slob_free(void *block, int size)
{
        slob_t *cur, *b = (slob_t *)block;
        unsigned long flags;

        if (!block)
                return;

        if (size)
                b->units = SLOB_UNITS(size);

        /* Find reinsertion point */
        spin_lock_irqsave(&slob_lock, flags);
        for (cur = slobfree; !(b > cur && b < cur->next); cur = cur->next)
                if (cur >= cur->next && (b > cur || b < cur->next))
                        break;

        if (b + b->units == cur->next) {
                b->units += cur->next->units;
                b->next = cur->next->next;
        } else
                b->next = cur->next;

        if (cur + cur->units == b) {
                cur->units += b->units;
                cur->next = b->next;
        } else
                cur->next = b;

        slobfree = cur;

        spin_unlock_irqrestore(&slob_lock, flags);
}



void check_slab(void) {
  cprintf("check_slab() success\n");
}

void
slab_init(void) {
  cprintf("use SLOB allocator\n");
  check_slab();
}

inline void
kmalloc_init(void) {
    slab_init();
    cprintf("kmalloc_init() succeeded!\n");
}

size_t
slab_allocated(void) {
  return 0;
}

size_t
kallocated(void) {
   return slab_allocated();
}

static int find_order(int size)
{
        int order = 0;
```

```
        for ( ; size > 4096 ; size >>=1)
                order++;
        return order;
}

static void *__kmalloc(size_t size, gfp_t gfp)
{
        slob_t *m;
        bigblock_t *bb;
        unsigned long flags;

        if (size < PAGE_SIZE - SLOB_UNIT) {
                m = slob_alloc(size + SLOB_UNIT, gfp, 0);
                return m ? (void *)(m + 1) : 0;
        }

        bb = slob_alloc(sizeof(bigblock_t), gfp, 0);
        if (!bb)
                return 0;

        bb->order = find_order(size);
        bb->pages = (void *)__slob_get_free_pages(gfp, bb->order);

        if (bb->pages) {
                spin_lock_irqsave(&block_lock, flags);
                bb->next = bigblocks;
                bigblocks = bb;
                spin_unlock_irqrestore(&block_lock, flags);
                return bb->pages;
        }

        slob_free(bb, sizeof(bigblock_t));
        return 0;
}

void *
kmalloc(size_t size)
{
  return __kmalloc(size, 0);
}


void kfree(void *block)
{
        bigblock_t *bb, **last = &bigblocks;
        unsigned long flags;

        if (!block)
                return;

        if (!((unsigned long)block & (PAGE_SIZE-1))) {
                /* might be on the big block list */
                spin_lock_irqsave(&block_lock, flags);
                for (bb = bigblocks; bb; last = &bb->next, bb = bb->next) {
                        if (bb->pages == block) {
                                *last = bb->next;
                                spin_unlock_irqrestore(&block_lock, flags);
                                __slob_free_pages((unsigned long)block, bb->order);
                                slob_free(bb, sizeof(bigblock_t));
                                return;
                        }
                }
                spin_unlock_irqrestore(&block_lock, flags);
        }
```

```
        slob_free((slob_t *)block - 1, 0);
        return;
}


unsigned int ksize(const void *block)
{
        bigblock_t *bb;
        unsigned long flags;

        if (!block)
                return 0;

        if (!((unsigned long)block & (PAGE_SIZE-1))) {
                spin_lock_irqsave(&block_lock, flags);
                for (bb = bigblocks; bb; bb = bb->next)
                        if (bb->pages == block) {
                                spin_unlock_irqrestore(&slob_lock, flags);
                                return PAGE_SIZE << bb->order;
                        }
                spin_unlock_irqrestore(&block_lock, flags);
        }

        return ((slob_t *)block - 1)->units * SLOB_UNIT;
}



[ucore/lab8_result/kern/mm/swap_fifo.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_SWAP_FIFO_H__
#define __KERN_MM_SWAP_FIFO_H__

#include <swap.h>
extern struct swap_manager swap_manager_fifo;

#endif
[ucore/lab8_result/kern/mm/mmu.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_MMU_H__
#define __KERN_MM_MMU_H__

/* Eflags register */
#define FL_CF           0x00000001  // Carry Flag
#define FL_PF           0x00000004  // Parity Flag
#define FL_AF           0x00000010  // Auxiliary carry Flag
#define FL_ZF           0x00000040  // Zero Flag
#define FL_SF           0x00000080  // Sign Flag
#define FL_TF           0x00000100  // Trap Flag
#define FL_IF           0x00000200  // Interrupt Flag
#define FL_DF           0x00000400  // Direction Flag
#define FL_OF           0x00000800  // Overflow Flag
#define FL_IOPL_MASK    0x00003000  // I/O Privilege Level bitmask
#define FL_IOPL_0       0x00000000  //   IOPL == 0
#define FL_IOPL_1       0x00001000  //   IOPL == 1
#define FL_IOPL_2       0x00002000  //   IOPL == 2
#define FL_IOPL_3       0x00003000  //   IOPL == 3
#define FL_NT           0x00004000  // Nested Task
#define FL_RF           0x00010000  // Resume Flag
#define FL_VM           0x00020000  // Virtual 8086 mode
#define FL_AC           0x00040000  // Alignment Check
#define FL_VIF          0x00080000  // Virtual Interrupt Flag
#define FL_VIP          0x00100000  // Virtual Interrupt Pending
#define FL_ID           0x00200000  // ID flag

/* Application segment type bits */
#define STA_X           0x8         // Executable segment
```

```
#define STA_E           0x4         // Expand down (non-executable segments)
#define STA_C           0x4         // Conforming code segment (executable only)
#define STA_W           0x2         // Writeable (non-executable segments)
#define STA_R           0x2         // Readable (executable segments)
#define STA_A           0x1         // Accessed

/* System segment type bits */
#define STS_T16A        0x1         // Available 16-bit TSS
#define STS_LDT         0x2         // Local Descriptor Table
#define STS_T16B        0x3         // Busy 16-bit TSS
#define STS_CG16        0x4         // 16-bit Call Gate
#define STS_TG          0x5         // Task Gate / Coum Transmitions
#define STS_IG16        0x6         // 16-bit Interrupt Gate
#define STS_TG16        0x7         // 16-bit Trap Gate
#define STS_T32A        0x9         // Available 32-bit TSS
#define STS_T32B        0xB         // Busy 32-bit TSS
#define STS_CG32        0xC         // 32-bit Call Gate
#define STS_IG32        0xE         // 32-bit Interrupt Gate
#define STS_TG32        0xF         // 32-bit Trap Gate

#ifdef __ASSEMBLER__

#define SEG_NULL                                                    \
    .word 0, 0;                                                     \
    .byte 0, 0, 0, 0

#define SEG_ASM(type,base,lim)                                      \
    .word (((lim) >> 12) & 0xffff), ((base) & 0xffff);              \
    .byte (((base) >> 16) & 0xff), (0x90 | (type)),                 \
        (0xC0 | (((lim) >> 28) & 0xf)), (((base) >> 24) & 0xff)

#else /* not __ASSEMBLER__ */

#include <defs.h>

/* Gate descriptors for interrupts and traps */
struct gatedesc {
    unsigned gd_off_15_0 : 16;      // low 16 bits of offset in segment
    unsigned gd_ss : 16;            // segment selector
    unsigned gd_args : 5;           // # args, 0 for interrupt/trap gates
    unsigned gd_rsv1 : 3;           // reserved(should be zero I guess)
    unsigned gd_type : 4;           // type(STS_{TG,IG32,TG32})
    unsigned gd_s : 1;              // must be 0 (system)
    unsigned gd_dpl : 2;            // descriptor(meaning new) privilege level
    unsigned gd_p : 1;              // Present
    unsigned gd_off_31_16 : 16;     // high bits of offset in segment
};

/* *
 * Set up a normal interrupt/trap gate descriptor
 *   - istrap: 1 for a trap (= exception) gate, 0 for an interrupt gate
 *   - sel: Code segment selector for interrupt/trap handler
 *   - off: Offset in code segment for interrupt/trap handler
 *   - dpl: Descriptor Privilege Level - the privilege level required
 *          for software to invoke this interrupt/trap gate explicitly
 *          using an int instruction.
 * */
#define SETGATE(gate, istrap, sel, off, dpl) {              \
        (gate).gd_off_15_0 = (uint32_t)(off) & 0xffff;      \
        (gate).gd_ss = (sel);                               \
        (gate).gd_args = 0;                                 \
        (gate).gd_rsv1 = 0;                                 \
        (gate).gd_type = (istrap) ? STS_TG32 : STS_IG32;    \
        (gate).gd_s = 0;                                    \
        (gate).gd_dpl = (dpl);                              \
```

```
        (gate).gd_p = 1;                                        \
        (gate).gd_off_31_16 = (uint32_t)(off) >> 16;            \
    }

/* Set up a call gate descriptor */
#define SETCALLGATE(gate, ss, off, dpl) {                       \
        (gate).gd_off_15_0 = (uint32_t)(off) & 0xffff;          \
        (gate).gd_ss = (ss);                                    \
        (gate).gd_args = 0;                                     \
        (gate).gd_rsv1 = 0;                                     \
        (gate).gd_type = STS_CG32;                              \
        (gate).gd_s = 0;                                        \
        (gate).gd_dpl = (dpl);                                  \
        (gate).gd_p = 1;                                        \
        (gate).gd_off_31_16 = (uint32_t)(off) >> 16;            \
    }

/* segment descriptors */
struct segdesc {
    unsigned sd_lim_15_0 : 16;      // low bits of segment limit
    unsigned sd_base_15_0 : 16;     // low bits of segment base address
    unsigned sd_base_23_16 : 8;     // middle bits of segment base address
    unsigned sd_type : 4;           // segment type (see STS_ constants)
    unsigned sd_s : 1;              // 0 = system, 1 = application
    unsigned sd_dpl : 2;            // descriptor Privilege Level
    unsigned sd_p : 1;              // present
    unsigned sd_lim_19_16 : 4;      // high bits of segment limit
    unsigned sd_avl : 1;            // unused (available for software use)
    unsigned sd_rsv1 : 1;           // reserved
    unsigned sd_db : 1;             // 0 = 16-bit segment, 1 = 32-bit segment
    unsigned sd_g : 1;              // granularity: limit scaled by 4K when set
    unsigned sd_base_31_24 : 8;     // high bits of segment base address
};

#define SEG_NULL                                                \
    (struct segdesc) {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

#define SEG(type, base, lim, dpl)                               \
    (struct segdesc) {                                          \
        ((lim) >> 12) & 0xffff, (base) & 0xffff,                \
        ((base) >> 16) & 0xff, type, 1, dpl, 1,                 \
        (unsigned)(lim) >> 28, 0, 0, 1, 1,                      \
        (unsigned) (base) >> 24                                 \
    }

#define SEGTSS(type, base, lim, dpl)                            \
    (struct segdesc) {                                          \
        (lim) & 0xffff, (base) & 0xffff,                        \
        ((base) >> 16) & 0xff, type, 0, dpl, 1,                 \
        (unsigned) (lim) >> 16, 0, 0, 1, 0,                     \
        (unsigned) (base) >> 24                                 \
    }

/* task state segment format (as described by the Pentium architecture book) */
struct taskstate {
    uint32_t ts_link;       // old ts selector
    uintptr_t ts_esp0;      // stack pointers and segment selectors
    uint16_t ts_ss0;        // after an increase in privilege level
    uint16_t ts_padding1;
    uintptr_t ts_esp1;
    uint16_t ts_ss1;
    uint16_t ts_padding2;
    uintptr_t ts_esp2;
    uint16_t ts_ss2;
    uint16_t ts_padding3;
```

```
    uintptr_t ts_cr3;       // page directory base
    uintptr_t ts_eip;       // saved state from last task switch
    uint32_t ts_eflags;
    uint32_t ts_eax;        // more saved state (registers)
    uint32_t ts_ecx;
    uint32_t ts_edx;
    uint32_t ts_ebx;
    uintptr_t ts_esp;
    uintptr_t ts_ebp;
    uint32_t ts_esi;
    uint32_t ts_edi;
    uint16_t ts_es;         // even more saved state (segment selectors)
    uint16_t ts_padding4;
    uint16_t ts_cs;
    uint16_t ts_padding5;
    uint16_t ts_ss;
    uint16_t ts_padding6;
    uint16_t ts_ds;
    uint16_t ts_padding7;
    uint16_t ts_fs;
    uint16_t ts_padding8;
    uint16_t ts_gs;
    uint16_t ts_padding9;
    uint16_t ts_ldt;
    uint16_t ts_padding10;
    uint16_t ts_t;          // trap on task switch
    uint16_t ts_iomb;       // i/o map base address
} __attribute__((packed));

#endif /* !__ASSEMBLER__ */

// A linear address 'la' has a three-part structure as follows:
//
// +--------10------+-------10-------+---------12----------+
// | Page Directory |   Page Table    | Offset within Page  |
// |      Index     |      Index      |                     |
// +----------------+-----------------+---------------------+
//  \--- PDX(la) --/ \--- PTX(la) --/ \---- PGOFF(la) ----/
//  \---------- PPN(la) -----------/
//
// The PDX, PTX, PGOFF, and PPN macros decompose linear addresses as shown.
// To construct a linear address la from PDX(la), PTX(la), and PGOFF(la),
// use PGADDR(PDX(la), PTX(la), PGOFF(la)).

// page directory index
#define PDX(la) ((((uintptr_t)(la)) >> PDXSHIFT) & 0x3FF)

// page table index
#define PTX(la) ((((uintptr_t)(la)) >> PTXSHIFT) & 0x3FF)

// page number field of address
#define PPN(la) (((uintptr_t)(la)) >> PTXSHIFT)

// offset in page
#define PGOFF(la) (((uintptr_t)(la)) & 0xFFF)

// construct linear address from indexes and offset
#define PGADDR(d, t, o) ((uintptr_t)((d) << PDXSHIFT | (t) << PTXSHIFT | (o)))

// address in page table or page directory entry
#define PTE_ADDR(pte)   ((uintptr_t)(pte) & ~0xFFF)
#define PDE_ADDR(pde)   PTE_ADDR(pde)

/* page directory and page table constants */
#define NPDEENTRY       1024                    // page directory entries per page directory
```

```
#define NPTEENTRY       1024                    // page table entries per page table

#define PGSIZE          4096                    // bytes mapped by a page
#define PGSHIFT         12                      // log2(PGSIZE)
#define PTSIZE          (PGSIZE * NPTEENTRY)    // bytes mapped by a page directory entry
#define PTSHIFT         22                      // log2(PTSIZE)

#define PTXSHIFT        12                      // offset of PTX in a linear address
#define PDXSHIFT        22                      // offset of PDX in a linear address

/* page table/directory entry flags */
#define PTE_P           0x001                   // Present
#define PTE_W           0x002                   // Writeable
#define PTE_U           0x004                   // User
#define PTE_PWT         0x008                   // Write-Through
#define PTE_PCD         0x010                   // Cache-Disable
#define PTE_A           0x020                   // Accessed
#define PTE_D           0x040                   // Dirty
#define PTE_PS          0x080                   // Page Size
#define PTE_MBZ         0x180                   // Bits must be zero
#define PTE_AVAIL       0xE00                   // Available for software use
                                                // The PTE_AVAIL bits aren't used by the kerne
l or interpreted by the
                                                // hardware, so user processes are allowed to
set them arbitrarily.

#define PTE_USER        (PTE_U | PTE_W | PTE_P)

/* Control Register flags */
#define CR0_PE          0x00000001              // Protection Enable
#define CR0_MP          0x00000002              // Monitor coProcessor
#define CR0_EM          0x00000004              // Emulation
#define CR0_TS          0x00000008              // Task Switched
#define CR0_ET          0x00000010              // Extension Type
#define CR0_NE          0x00000020              // Numeric Errror
#define CR0_WP          0x00010000              // Write Protect
#define CR0_AM          0x00040000              // Alignment Mask
#define CR0_NW          0x20000000              // Not Writethrough
#define CR0_CD          0x40000000              // Cache Disable
#define CR0_PG          0x80000000              // Paging

#define CR4_PCE         0x00000100              // Performance counter enable
#define CR4_MCE         0x00000040              // Machine Check Enable
#define CR4_PSE         0x00000010              // Page Size Extensions
#define CR4_DE          0x00000008              // Debugging Extensions
#define CR4_TSD         0x00000004              // Time Stamp Disable
#define CR4_PVI         0x00000002              // Protected-Mode Virtual Interrupts
#define CR4_VME         0x00000001              // V86 Mode Extensions

#endif /* !__KERN_MM_MMU_H__ */

[ucore/lab8_result/kern/schedule/default_sched.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <list.h>
#include <proc.h>
#include <assert.h>
#include <default_sched.h>

#define USE_SKEW_HEAP 1

/* You should define the BigStride constant here*/
/* LAB6: YOUR CODE */
#define BIG_STRIDE    0x7FFFFFFF /* ??? */

/* The compare function for two skew_heap_node_t's and the
```

```
 * corresponding procs*/
static int
proc_stride_comp_f(void *a, void *b)
{
     struct proc_struct *p = le2proc(a, lab6_run_pool);
     struct proc_struct *q = le2proc(b, lab6_run_pool);
     int32_t c = p->lab6_stride - q->lab6_stride;
     if (c > 0) return 1;
     else if (c == 0) return 0;
     else return -1;
}

/*
 * stride_init initializes the run-queue rq with correct assignment for
 * member variables, including:
 *
 *   - run_list: should be a empty list after initialization.
 *   - lab6_run_pool: NULL
 *   - proc_num: 0
 *   - max_time_slice: no need here, the variable would be assigned by the caller.
 *
 * hint: see proj13.1/libs/list.h for routines of the list structures.
 */
static void
stride_init(struct run_queue *rq) {
     /* LAB6: YOUR CODE */
     list_init(&(rq->run_list));
     rq->lab6_run_pool = NULL;
     rq->proc_num = 0;
}

/*
 * stride_enqueue inserts the process ``proc'' into the run-queue
 * ``rq''. The procedure should verify/initialize the relevant members
 * of ``proc'', and then put the ``lab6_run_pool'' node into the
 * queue(since we use priority queue here). The procedure should also
 * update the meta date in ``rq'' structure.
 *
 * proc->time_slice denotes the time slices allocation for the
 * process, which should set to rq->max_time_slice.
 *
 * hint: see proj13.1/libs/skew_heap.h for routines of the priority
 * queue structures.
 */
static void
stride_enqueue(struct run_queue *rq, struct proc_struct *proc) {
     /* LAB6: YOUR CODE */
#if USE_SKEW_HEAP
     rq->lab6_run_pool =
          skew_heap_insert(rq->lab6_run_pool, &(proc->lab6_run_pool), proc_stride_comp_f);
#else
     assert(list_empty(&(proc->run_link)));
     list_add_before(&(rq->run_list), &(proc->run_link));
#endif
     if (proc->time_slice == 0 || proc->time_slice > rq->max_time_slice) {
          proc->time_slice = rq->max_time_slice;
     }
     proc->rq = rq;
     rq->proc_num ++;
}

/*
 * stride_dequeue removes the process ``proc'' from the run-queue
 * ``rq'', the operation would be finished by the skew_heap_remove
 * operations. Remember to update the ``rq'' structure.
```

```
 *
 * hint: see proj13.1/libs/skew_heap.h for routines of the priority
 * queue structures.
 */
static void
stride_dequeue(struct run_queue *rq, struct proc_struct *proc) {
     /* LAB6: YOUR CODE */
#if USE_SKEW_HEAP
     rq->lab6_run_pool =
          skew_heap_remove(rq->lab6_run_pool, &(proc->lab6_run_pool), proc_stride_comp_f);
#else
     assert(!list_empty(&(proc->run_link)) && proc->rq == rq);
     list_del_init(&(proc->run_link));
#endif
     rq->proc_num --;
}
/*
 * stride_pick_next pick the element from the ``run-queue'', with the
 * minimum value of stride, and returns the corresponding process
 * pointer. The process pointer would be calculated by macro le2proc,
 * see proj13.1/kern/process/proc.h for definition. Return NULL if
 * there is no process in the queue.
 *
 * When one proc structure is selected, remember to update the stride
 * property of the proc. (stride += BIG_STRIDE / priority)
 *
 * hint: see proj13.1/libs/skew_heap.h for routines of the priority
 * queue structures.
 */
static struct proc_struct *
stride_pick_next(struct run_queue *rq) {
     /* LAB6: YOUR CODE */
#if USE_SKEW_HEAP
     if (rq->lab6_run_pool == NULL) return NULL;
     struct proc_struct *p = le2proc(rq->lab6_run_pool, lab6_run_pool);
#else
     list_entry_t *le = list_next(&(rq->run_list));

     if (le == &rq->run_list)
          return NULL;

     struct proc_struct *p = le2proc(le, run_link);
     le = list_next(le);
     while (le != &rq->run_list)
     {
          struct proc_struct *q = le2proc(le, run_link);
          if ((int32_t)(p->lab6_stride - q->lab6_stride) > 0)
               p = q;
          le = list_next(le);
     }
#endif
     if (p->lab6_priority == 0)
          p->lab6_stride += BIG_STRIDE;
     else p->lab6_stride += BIG_STRIDE / p->lab6_priority;
     return p;
}

/*
 * stride_proc_tick works with the tick event of current process. You
 * should check whether the time slices for current process is
 * exhausted and update the proc struct ``proc''. proc->time_slice
 * denotes the time slices left for current
 * process. proc->need_resched is the flag variable for process
 * switching.
 */
```

```
static void
stride_proc_tick(struct run_queue *rq, struct proc_struct *proc) {
     /* LAB6: YOUR CODE */
     if (proc->time_slice > 0) {
          proc->time_slice --;
     }
     if (proc->time_slice == 0) {
          proc->need_resched = 1;
     }
}

struct sched_class default_sched_class = {
     .name = "stride_scheduler",
     .init = stride_init,
     .enqueue = stride_enqueue,
     .dequeue = stride_dequeue,
     .pick_next = stride_pick_next,
     .proc_tick = stride_proc_tick,
};
```

[ucore/lab8_result/kern/schedule/sched.c] ++++++++++++++++++++++++++++++

```
#include <list.h>
#include <sync.h>
#include <proc.h>
#include <sched.h>
#include <stdio.h>
#include <assert.h>
#include <default_sched.h>

static list_entry_t timer_list;

static struct sched_class *sched_class;

static struct run_queue *rq;

static inline void
sched_class_enqueue(struct proc_struct *proc) {
    if (proc != idleproc) {
        sched_class->enqueue(rq, proc);
    }
}

static inline void
sched_class_dequeue(struct proc_struct *proc) {
    sched_class->dequeue(rq, proc);
}

static inline struct proc_struct *
sched_class_pick_next(void) {
    return sched_class->pick_next(rq);
}

static void
sched_class_proc_tick(struct proc_struct *proc) {
    if (proc != idleproc) {
        sched_class->proc_tick(rq, proc);
    }
    else {
        proc->need_resched = 1;
    }
}

static struct run_queue __rq;

void
```

```
sched_init(void) {
    list_init(&timer_list);

    sched_class = &default_sched_class;

    rq = &__rq;
    rq->max_time_slice = 5;
    sched_class->init(rq);

    cprintf("sched class: %s\n", sched_class->name);
}

void
wakeup_proc(struct proc_struct *proc) {
    assert(proc->state != PROC_ZOMBIE);
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        if (proc->state != PROC_RUNNABLE) {
            proc->state = PROC_RUNNABLE;
            proc->wait_state = 0;
            if (proc != current) {
                sched_class_enqueue(proc);
            }
        }
        else {
            warn("wakeup runnable process.\n");
        }
    }
    local_intr_restore(intr_flag);
}

void
schedule(void) {
    bool intr_flag;
    struct proc_struct *next;
    local_intr_save(intr_flag);
    {
        current->need_resched = 0;
        if (current->state == PROC_RUNNABLE) {
            sched_class_enqueue(current);
        }
        if ((next = sched_class_pick_next()) != NULL) {
            sched_class_dequeue(next);
        }
        if (next == NULL) {
            next = idleproc;
        }
        next->runs ++;
        if (next != current) {
            proc_run(next);
        }
    }
    local_intr_restore(intr_flag);
}

void
add_timer(timer_t *timer) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        assert(timer->expires > 0 && timer->proc != NULL);
        assert(list_empty(&(timer->timer_link)));
        list_entry_t *le = list_next(&timer_list);
        while (le != &timer_list) {
```

```
            timer_t *next = le2timer(le, timer_link);
            if (timer->expires < next->expires) {
                next->expires -= timer->expires;
                break;
            }
            timer->expires -= next->expires;
            le = list_next(le);
        }
        list_add_before(le, &(timer->timer_link));
    }
    local_intr_restore(intr_flag);
}

void
del_timer(timer_t *timer) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        if (!list_empty(&(timer->timer_link))) {
            if (timer->expires != 0) {
                list_entry_t *le = list_next(&(timer->timer_link));
                if (le != &timer_list) {
                    timer_t *next = le2timer(le, timer_link);
                    next->expires += timer->expires;
                }
            }
            list_del_init(&(timer->timer_link));
        }
    }
    local_intr_restore(intr_flag);
}

void
run_timer_list(void) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        list_entry_t *le = list_next(&timer_list);
        if (le != &timer_list) {
            timer_t *timer = le2timer(le, timer_link);
            assert(timer->expires != 0);
            timer->expires --;
            while (timer->expires == 0) {
                le = list_next(le);
                struct proc_struct *proc = timer->proc;
                if (proc->wait_state != 0) {
                    assert(proc->wait_state & WT_INTERRUPTED);
                }
                else {
                    warn("process %d's wait_state == 0.\n", proc->pid);
                }
                wakeup_proc(proc);
                del_timer(timer);
                if (le == &timer_list) {
                    break;
                }
                timer = le2timer(le, timer_link);
            }
        }
        sched_class_proc_tick(current);
    }
    local_intr_restore(intr_flag);
}
[ucore/lab8_result/kern/schedule/sched.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SCHEDULE_SCHED_H__
```

```c
#define __KERN_SCHEDULE_SCHED_H__

#include <defs.h>
#include <list.h>
#include <skew_heap.h>

struct proc_struct;

typedef struct {
    unsigned int expires;
    struct proc_struct *proc;
    list_entry_t timer_link;
} timer_t;

#define le2timer(le, member)            \
to_struct((le), timer_t, member)

static inline timer_t *
timer_init(timer_t *timer, struct proc_struct *proc, int expires) {
    timer->expires = expires;
    timer->proc = proc;
    list_init(&(timer->timer_link));
    return timer;
}

struct run_queue;

// The introduction of scheduling classes is borrrowed from Linux, and makes the
// core scheduler quite extensible. These classes (the scheduler modules) encapsulate
// the scheduling policies.
struct sched_class {
    // the name of sched_class
    const char *name;
    // Init the run queue
    void (*init)(struct run_queue *rq);
    // put the proc into runqueue, and this function must be called with rq_lock
    void (*enqueue)(struct run_queue *rq, struct proc_struct *proc);
    // get the proc out runqueue, and this function must be called with rq_lock
    void (*dequeue)(struct run_queue *rq, struct proc_struct *proc);
    // choose the next runnable task
    struct proc_struct *(*pick_next)(struct run_queue *rq);
    // dealer of the time-tick
    void (*proc_tick)(struct run_queue *rq, struct proc_struct *proc);
    /* for SMP support in the future
     *  load_balance
     *     void (*load_balance)(struct rq* rq);
     *  get some proc from this rq, used in load_balance,
     *  return value is the num of gotten proc
     *  int (*get_proc)(struct rq* rq, struct proc* procs_moved[]);
     */
};

struct run_queue {
    list_entry_t run_list;
    unsigned int proc_num;
    int max_time_slice;
    // For LAB6 ONLY
    skew_heap_entry_t *lab6_run_pool;
};

void sched_init(void);
void wakeup_proc(struct proc_struct *proc);
void schedule(void);
void add_timer(timer_t *timer);
void del_timer(timer_t *timer);
```

```
void run_timer_list(void);

#endif /* !__KERN_SCHEDULE_SCHED_H__ */

[ucore/lab8_result/kern/schedule/default_sched.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SCHEDULE_SCHED_RR_H__
#define __KERN_SCHEDULE_SCHED_RR_H__

#include <sched.h>

extern struct sched_class default_sched_class;

#endif /* !__KERN_SCHEDULE_SCHED_RR_H__ */

[ucore/lab8_result/boot/bootasm.S] ++++++++++++++++++++++++++++++
#include <asm.h>

# Start the CPU: switch to 32-bit protected mode, jump into C.
# The BIOS loads this code from the first sector of the hard disk into
# memory at physical address 0x7c00 and starts executing in real mode
# with %cs=0 %ip=7c00.

.set PROT_MODE_CSEG,        0x8                     # kernel code segment selector
.set PROT_MODE_DSEG,        0x10                    # kernel data segment selector
.set CR0_PE_ON,             0x1                     # protected mode enable flag
.set SMAP,                  0x534d4150

# start address should be 0:7c00, in real mode, the beginning address of the running bootloade
r
.globl start
start:
.code16                                             # Assemble for 16-bit mode
    cli                                             # Disable interrupts
    cld                                             # String operations increment

    # Set up the important data segment registers (DS, ES, SS).
    xorw %ax, %ax                                   # Segment number zero
    movw %ax, %ds                                   # -> Data Segment
    movw %ax, %es                                   # -> Extra Segment
    movw %ax, %ss                                   # -> Stack Segment

    # Enable A20:
    #  For backwards compatibility with the earliest PCs, physical
    #  address line 20 is tied low, so that addresses higher than
    #  1MB wrap around to zero by default. This code undoes this.
seta20.1:
    inb $0x64, %al                                  # Wait for not busy
    testb $0x2, %al
    jnz seta20.1

    movb $0xd1, %al                                 # 0xd1 -> port 0x64
    outb %al, $0x64

seta20.2:
    inb $0x64, %al                                  # Wait for not busy
    testb $0x2, %al
    jnz seta20.2

    movb $0xdf, %al                                 # 0xdf -> port 0x60
    outb %al, $0x60

probe_memory:
    movl $0, 0x8000
    xorl %ebx, %ebx
    movw $0x8004, %di
```

```
start_probe:
    movl $0xE820, %eax
    movl $20, %ecx
    movl $SMAP, %edx
    int $0x15
    jnc cont
    movw $12345, 0x8000
    jmp finish_probe
cont:
    addw $20, %di
    incl 0x8000
    cmpl $0, %ebx
    jnz start_probe
finish_probe:

    # Switch from real to protected mode, using a bootstrap GDT
    # and segment translation that makes virtual addresses
    # identical to physical addresses, so that the
    # effective memory map does not change during the switch.
    lgdt gdtdesc
    movl %cr0, %eax
    orl $CR0_PE_ON, %eax
    movl %eax, %cr0

    # Jump to next instruction, but in 32-bit code segment.
    # Switches processor into 32-bit mode.
    ljmp $PROT_MODE_CSEG, $protcseg

.code32                                             # Assemble for 32-bit mode
protcseg:
    # Set up the protected-mode data segment registers
    movw $PROT_MODE_DSEG, %ax                       # Our data segment selector
    movw %ax, %ds                                   # -> DS: Data Segment
    movw %ax, %es                                   # -> ES: Extra Segment
    movw %ax, %fs                                   # -> FS
    movw %ax, %gs                                   # -> GS
    movw %ax, %ss                                   # -> SS: Stack Segment

    # Set up the stack pointer and call into C. The stack region is from 0--start(0x7c00)
    movl $0x0, %ebp
    movl $start, %esp
    call bootmain

    # If bootmain returns (it shouldn't), loop.
spin:
    jmp spin

.data
# Bootstrap GDT
.p2align 2                                          # force 4 byte alignment
gdt:
    SEG_NULLASM                                     # null seg
    SEG_ASM(STA_X|STA_R, 0x0, 0xffffffff)           # code seg for bootloader and kernel
    SEG_ASM(STA_W, 0x0, 0xffffffff)                 # data seg for bootloader and kernel

gdtdesc:
    .word 0x17                                      # sizeof(gdt) - 1
    .long gdt                                       # address gdt
[ucore/lab8_result/boot/asm.h] ++++++++++++++++++++++++++++++
#ifndef __BOOT_ASM_H__
#define __BOOT_ASM_H__

/* Assembler macros to create x86 segments */

/* Normal segment */
```

```
#define SEG_NULLASM                                             \
    .word 0, 0;                                                 \
    .byte 0, 0, 0, 0

#define SEG_ASM(type,base,lim)                                  \
    .word (((lim) >> 12) & 0xffff), ((base) & 0xffff);          \
    .byte (((base) >> 16) & 0xff), (0x90 | (type)),             \
        (0xC0 | (((lim) >> 28) & 0xf)), (((base) >> 24) & 0xff)


/* Application segment type bits */
#define STA_X       0x8     // Executable segment
#define STA_E       0x4     // Expand down (non-executable segments)
#define STA_C       0x4     // Conforming code segment (executable only)
#define STA_W       0x2     // Writeable (non-executable segments)
#define STA_R       0x2     // Readable (executable segments)
#define STA_A       0x1     // Accessed

#endif /* !__BOOT_ASM_H__ */

[ucore/lab8_result/boot/bootmain.c] ++++++++++++++++++++++++++++++
#include <defs.h>
#include <x86.h>
#include <elf.h>

/* *********************************************************************
 * This a dirt simple boot loader, whose sole job is to boot
 * an ELF kernel image from the first IDE hard disk.
 *
 * DISK LAYOUT
 *  * This program(bootasm.S and bootmain.c) is the bootloader.
 *    It should be stored in the first sector of the disk.
 *
 *  * The 2nd sector onward holds the kernel image.
 *
 *  * The kernel image must be in ELF format.
 *
 * BOOT UP STEPS
 *  * when the CPU boots it loads the BIOS into memory and executes it
 *
 *  * the BIOS intializes devices, sets of the interrupt routines, and
 *    reads the first sector of the boot device(e.g., hard-drive)
 *    into memory and jumps to it.
 *
 *  * Assuming this boot loader is stored in the first sector of the
 *    hard-drive, this code takes over...
 *
 *  * control starts in bootasm.S -- which sets up protected mode,
 *    and a stack so C code then run, then calls bootmain()
 *
 *  * bootmain() in this file takes over, reads in the kernel and jumps to it.
 * */

#define SECTSIZE        512
#define ELFHDR          ((struct elfhdr *)0x10000)      // scratch space

/* waitdisk - wait for disk ready */
static void
waitdisk(void) {
    while ((inb(0x1F7) & 0xC0) != 0x40)
        /* do nothing */;
}

/* readsect - read a single sector at @secno into @dst */
static void
```

```
readsect(void *dst, uint32_t secno) {
    // wait for disk to be ready
    waitdisk();

    outb(0x1F2, 1);                         // count = 1
    outb(0x1F3, secno & 0xFF);
    outb(0x1F4, (secno >> 8) & 0xFF);
    outb(0x1F5, (secno >> 16) & 0xFF);
    outb(0x1F6, ((secno >> 24) & 0xF) | 0xE0);
    outb(0x1F7, 0x20);                      // cmd 0x20 - read sectors

    // wait for disk to be ready
    waitdisk();

    // read a sector
    insl(0x1F0, dst, SECTSIZE / 4);
}

/* *
 * readseg - read @count bytes at @offset from kernel into virtual address @va,
 * might copy more than asked.
 * */
static void
readseg(uintptr_t va, uint32_t count, uint32_t offset) {
    uintptr_t end_va = va + count;

    // round down to sector boundary
    va -= offset % SECTSIZE;

    // translate from bytes to sectors; kernel starts at sector 1
    uint32_t secno = (offset / SECTSIZE) + 1;

    // If this is too slow, we could read lots of sectors at a time.
    // We'd write more to memory than asked, but it doesn't matter --
    // we load in increasing order.
    for (; va < end_va; va += SECTSIZE, secno ++) {
        readsect((void *)va, secno);
    }
}

/* bootmain - the entry of bootloader */
void
bootmain(void) {
    // read the 1st page off disk
    readseg((uintptr_t)ELFHDR, SECTSIZE * 8, 0);

    // is this a valid ELF?
    if (ELFHDR->e_magic != ELF_MAGIC) {
        goto bad;
    }

    struct proghdr *ph, *eph;

    // load each program segment (ignores ph flags)
    ph = (struct proghdr *)((uintptr_t)ELFHDR + ELFHDR->e_phoff);
    eph = ph + ELFHDR->e_phnum;
    for (; ph < eph; ph ++) {
        readseg(ph->p_va & 0xFFFFFF, ph->p_memsz, ph->p_offset);
    }

    // call the entry point from the ELF header
    // note: does not return
    ((void (*)(void))(ELFHDR->e_entry & 0xFFFFFF))();

bad:
```

```
    outw(0x8A00, 0x8A00);
    outw(0x8A00, 0x8E00);

    /* do nothing */
    while (1);
}
```