[src/user/ls.c] ++++++++++++++++++++++++++++
[src/user/faultread.c] ++++++++++++++++++++++++++++
[src/user/faultreadkernel.c] ++++++++++++++++++++++++++++
[src/user/sleepkill.c] ++++++++++++++++++++++++++++
[src/user/hello.c] ++++++++++++++++++++++++++++
[src/user/badarg.c] ++++++++++++++++++++++++++++
[src/user/exit.c] ++++++++++++++++++++++++++++
[src/user/softint.c] ++++++++++++++++++++++++++++
[src/user/priority.c] ++++++++++++++++++++++++++++
[src/user/badsegment.c] ++++++++++++++++++++++++++++
[src/user/forktest.c] ++++++++++++++++++++++++++++
[src/user/waitkill.c] ++++++++++++++++++++++++++++
[src/user/yield.c] ++++++++++++++++++++++++++++
[src/user/matrix.c] ++++++++++++++++++++++++++++
[src/user/pgdir.c] ++++++++++++++++++++++++++++
[src/user/sh.c] ++++++++++++++++++++++++++++
[src/user/sfs_filetest1.c] ++++++++++++++++++++++++++++
[src/user/testbss.c] ++++++++++++++++++++++++++++
[src/user/libs/syscall.c] ++++++++++++++++++++++++++++
[src/user/libs/file.h] ++++++++++++++++++++++++++++
[src/user/libs/panic.c] ++++++++++++++++++++++++++++
[src/user/libs/umain.c] ++++++++++++++++++++++++++++
[src/user/libs/dir.c] ++++++++++++++++++++++++++++
[src/user/libs/ulib.c] ++++++++++++++++++++++++++++
[src/user/libs/stdio.c] ++++++++++++++++++++++++++++
[src/user/libs/initcode.S] ++++++++++++++++++++++++++++
[src/user/libs/syscall.h] ++++++++++++++++++++++++++++
[src/user/libs/ulib.h] ++++++++++++++++++++++++++++
[src/user/libs/dir.h] ++++++++++++++++++++++++++++
[src/user/libs/file.c] ++++++++++++++++++++++++++++
[src/user/libs/lock.h] ++++++++++++++++++++++++++++
[src/user/spin.c] ++++++++++++++++++++++++++++
[src/user/sleep.c] ++++++++++++++++++++++++++++
[src/user/forktree.c] ++++++++++++++++++++++++++++
[src/user/divzero.c] ++++++++++++++++++++++++++++
[src/tools/sign.c] ++++++++++++++++++++++++++++

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

int main(int argc, char* argv[])
{
        struct stat st;
        if (argc != 3) {
                fprintf(stderr, "Usage: <input file> <output file>\n");
        }
        if (stat(argv[1], &st) != 0) {
                fprintf(stderr, "open file error: %s : %s \n", argv[1], strerror(errno));
        }
        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* fp_in = fopen(argv[1], "rb");
        int size = fread(buf, 1, st.st_size, fp_in);
        if (size != st.st_size) {
                fprintf(stderr, "read %s error, size diff %ld vs. %d\n", argv[1], st.st_size,
size);
                return -1;
        }
        fclose(fp_in);
```

```
        buf[510] = 0x55;
        buf[511] = 0xAA;
        FILE* fp_out = fopen(argv[2], "wb+");
        size = fwrite(buf, 1, 512, fp_out);
        if (size != 512) {
                fprintf(stderr, "write %s error, size diff %d vs. %d\n", argv[2], 512, size);
        }
        fclose(fp_out);
        printf("build 512 bytes for boot sector: %s success!\n", argv[2]);
        return 0;
}
[src/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>
#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;
typedef char i8;
typedef unsigned char u8;
typedef short i16;
typedef unsigned short u16;
typedef int i32;
typedef unsigned int u32;
typedef long long i64;
typedef unsigned long long u64;

#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 u32 __hash32(u32 val, unsigned int bits) {
        u32 hash = val * GOLDEN_RATIO_PRIME_32;
        return (hash >> (32 - bits));
}

static u32 hash32(u32 val) {
        return __hash32(val, HASH_SHIFT);
}

static u32 hash64(u64 val) {
        return __hash32((u32)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 {
        u32 ino;
        struct cache_block *hash_next;
        void *cache;
};

struct cache_inode {
        struct inode {
                u32 size;
                u16 type;
                u16 nlinks;
                u32 blocks;
                u32 direct[SFS_NDIRECT];
                u32 indirect;
                u32 db_indirect;
        } inode;
        ino_t real;
        u32 ino;
        u32 nblks;
        struct cache_block *l1, *l2;
        struct cache_inode *hash_next;
};

struct sfs_fs {
        struct {
                u32 magic;
                u32 blocks;
                u32 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;
        u32 ninos, next_ino;
        struct cache_inode *root;
        struct cache_inode *inodes[HASH_LIST_SIZE];
        struct cache_block *blocks[HASH_LIST_SIZE];
};

struct sfs_entry {
        u32 ino;
        char name[SFS_MAX_FNAME_LEN + 1];
};

static u32 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, u32 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, u32 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, u32 ino, u16 typ
e) {
        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) {
```

```
        u32 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 bloc
ks).\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, u32 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];
        u32 i, j, ino = SFS_BLKN_FREEMAP;
        u32 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) {
                        u32 start = 0, end = SFS_BLKBITS;
                        if (i < next_ino) {
                                start = next_ino - i;
                        }
                        if (i + SFS_BLKBITS > ninos) {
                                end = ninos - i;
                        }
                        u32 *data = (u32 *)buffer;
                        const u32 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(u32))
#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, u32 *inop) {
        u32 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, u32 ino, c
onst char *filename) {
        static_assert(SFS_LN_NBLKS <= SFS_L2_NBLKS, "SFS_LN_NBLKS <= SFS_L2_NBLKS");
        assert(size <= SFS_BLKSIZE);
        u32 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));
                u32 *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));
                u32 *data2 = file->l2->cache;
                update_cache(sfs, &(file->l1), &data2[nblks / SFS_BLK_NENTRY]);
                u32 *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 *fil
e, const char *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);
        u32 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 c
urfd, 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, in
t 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, in
o_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, m
ode, 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);
                u32 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];
        u32 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;
}

[src/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;
}
[src/tools/print_elf.c] ++++++++++++++++++++++++++++++
#include <stdio.h>
```

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

typedef int bool;
typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long u64;

#define ELF_MAGIC 0x464C457FU

// flag bits for proghdr::p_flags
#define ELF_PF_X 1
#define ELF_PF_W 2
#define ELF_PF_R 4

struct elfhdr32 {
        u32 e_magic;            // must equal ELF_MAGIC
        u8 e_elf[12];
        u16 e_type;             // 1=relocatable, 2=executable, 3=shared object, 4=core image
        u16 e_machine;             // 3=x86, 4=68K, etc.
        u32 e_version;             // file version = 1
        u32 e_entry;            // entry point if executable
        u32 e_phoff;            // program header offset
        u32 e_shoff;            // section header offset
        u32 e_flags;            // architecture-specific flags = 0
        u16 e_ehsize;            // elf header size
        u16 e_phentsize;        // program header entry size
        u16 e_phnum;            // program header number
        u16 e_shentsize;        // section header entry size
        u16 e_shnum;            // section header number
        u16 e_shstrndx;         // section header name string index
};

struct proghdr32 {
        u32 p_type;      // loadable code or data, dynamic linking info, etc.
        u32 p_offset;    // file segment offset
        u32 p_va;            // virtual address to map segment
        u32 p_pa;            // physical address, not used
        u32 p_filesz;    // size of segment in file
        u32 p_memsz;     // size of segment in memory (bigger if contains bss)
        u32 p_flags;     // read/write/execut bits
        u32 p_align;     // required alignment, invariably hardware page size
};

struct elfhdr64 {
        u32 e_magic;    // must equal ELF_MAGIC
        u8 e_elf[12];
        u16 e_type;             // 1=relocatable, 2=executable, 3=shared object, 4=core image
        u16 e_machine;             // 3=x86, 4=68K, etc.
        u32 e_version;             // file version = 1
        u64 e_entry;            // entry point if executable
        u64 e_phoff;            // program header offset
        u64 e_shoff;            // section header offset
        u32 e_flags;            // architecture-specific flags = 0
        u16 e_ehsize;            // elf header size
        u16 e_phentsize;    // program header entry size
        u16 e_phnum;            // program header number
        u16 e_shentsize;    // section header entry size
        u16 e_shnum;            // section header number
        u16 e_shstrndx;     // section header name string index
};
```

```
struct proghdr64 {
        u32 p_type;      // loadable code or data, dynamic linking info, etc.
        u32 p_offset;    // file segment offset
        u64 p_va;            // virtual address to map segment
        u64 p_pa;            // physical address, not used
        u64 p_filesz;    // size of segment in file
        u64 p_memsz;     // size of segment in memory (bigger if contains bss)
        u64 p_flags;     // read/write/execut bits
        u64 p_align;     // required alignment, invariably hardware page size
};

void print_elf32(struct elfhdr32* elf_head)
{
        if (elf_head->e_magic != ELF_MAGIC) {
                fprintf(stderr, "bad elf magic!\r\n");
                return;
        }
        printf("parsing elf32 ==================================================\r\n");
        printf("[elf header]\r\n");
        printf(
                        "e_magic          :   0x%x\r\n"
                        "e_elf            :   0x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x\r\n"
                        "e_type           :   0x%x\r\n"
                        "e_machine        :   0x%x\r\n"
                        "e_version        :   0x%x\r\n"
                        "e_entry          :   0x%x\r\n"
                        "e_phoff          :   0x%x\r\n"
                        "e_shoff          :   0x%x\r\n"
                        "e_flags          :   0x%x\r\n"
                        "e_ehsize         :   0x%x\r\n"
                        "e_phentsize      :   0x%x\r\n"
                        "e_phnum          :   0x%x\r\n"
                        "e_shentsize      :   0x%x\r\n"
                        "e_shnum          :   0x%x\r\n"
                        "e_shstrndx       :   0x%x\r\n",
                        elf_head->e_magic         ,
                        elf_head->e_elf[0]           ,
                        elf_head->e_elf[1]           ,
                        elf_head->e_elf[2]           ,
                        elf_head->e_elf[3]           ,
                        elf_head->e_elf[4]           ,
                        elf_head->e_elf[5]           ,
                        elf_head->e_elf[6]           ,
                        elf_head->e_elf[7]           ,
                        elf_head->e_elf[8]           ,
                        elf_head->e_elf[9]           ,
                        elf_head->e_elf[10]          ,
                        elf_head->e_elf[11]          ,
                        elf_head->e_type          ,
                        elf_head->e_machine       ,
                        elf_head->e_version       ,
                        elf_head->e_entry         ,
                        elf_head->e_phoff         ,
                        elf_head->e_shoff         ,
                        elf_head->e_flags         ,
                        elf_head->e_ehsize        ,
                        elf_head->e_phentsize     ,
                        elf_head->e_phnum         ,
                        elf_head->e_shentsize     ,
                        elf_head->e_shnum         ,
                        elf_head->e_shstrndx      );

        printf("[program header]\r\n");
        struct proghdr32 *ph, *eph;
```

```
        ph = (struct proghdr32*)((u64)elf_head + elf_head->e_phoff);
        eph = ph + elf_head->e_phnum;
        for (; ph < eph; ph++) {
                printf(
                                "------------------------------\r\n"
                                "p_type            :   0x%x\r\n"
                                "p_offset          :   0x%x\r\n"
                                "p_va              :   0x%x\r\n"
                                "p_pa              :   0x%x\r\n"
                                "p_filesz          :   0x%x\r\n"
                                "p_memsz           :   0x%x\r\n"
                                "p_flags           :   0x%x\r\n"
                                "p_align           :   0x%x\r\n",
                                ph->p_type                      ,
                                ph->p_offset                    ,
                                ph->p_va                        ,
                                ph->p_pa                        ,
                                ph->p_filesz                    ,
                                ph->p_memsz                     ,
                                ph->p_flags                     ,
                                ph->p_align                     );
        }
}

void print_elf64(struct elfhdr64* elf_head)
{
        if (elf_head->e_magic != ELF_MAGIC) {
                fprintf(stderr, "bad elf magic!\r\n");
                return;
        }
        printf("parsing elf64 ==================================================\r\n");
        printf("[elf header]\r\n");
        printf(
                        "e_magic           :   0x%x\r\n"
                        "e_elf             :   0x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%0
2x\r\n"
                        "e_type            :   0x%x\r\n"
                        "e_machine         :   0x%x\r\n"
                        "e_version         :   0x%x\r\n"
                        "e_entry           :   0x%lx\r\n"
                        "e_phoff           :   0x%lx\r\n"
                        "e_shoff           :   0x%lx\r\n"
                        "e_flags           :   0x%x\r\n"
                        "e_ehsize          :   0x%x\r\n"
                        "e_phentsize       :   0x%x\r\n"
                        "e_phnum           :   0x%x\r\n"
                        "e_shentsize       :   0x%x\r\n"
                        "e_shnum           :   0x%x\r\n"
                        "e_shstrndx        :   0x%x\r\n",
                        elf_head->e_magic          ,
                        elf_head->e_elf[0]            ,
                        elf_head->e_elf[1]            ,
                        elf_head->e_elf[2]            ,
                        elf_head->e_elf[3]            ,
                        elf_head->e_elf[4]            ,
                        elf_head->e_elf[5]            ,
                        elf_head->e_elf[6]            ,
                        elf_head->e_elf[7]            ,
                        elf_head->e_elf[8]            ,
                        elf_head->e_elf[9]            ,
                        elf_head->e_elf[10]           ,
                        elf_head->e_elf[11]           ,
                        elf_head->e_type           ,
                        elf_head->e_machine        ,
                        elf_head->e_version        ,
```

```
                        elf_head->e_entry         ,
                        elf_head->e_phoff         ,
                        elf_head->e_shoff         ,
                        elf_head->e_flags         ,
                        elf_head->e_ehsize        ,
                        elf_head->e_phentsize     ,
                        elf_head->e_phnum         ,
                        elf_head->e_shentsize     ,
                        elf_head->e_shnum         ,
                        elf_head->e_shstrndx      );

        printf("[program header]\r\n");
        struct proghdr64 *ph, *eph;
        ph = (struct proghdr64*)((u64)elf_head + elf_head->e_phoff);
        eph = ph + elf_head->e_phnum;
        for (; ph < eph; ph++) {
                printf(
                                "------------------------------\r\n"
                                "p_type             :   0x%x\r\n"
                                "p_offset           :   0x%x\r\n"
                                "p_va               :   0x%lx\r\n"
                                "p_pa               :   0x%lx\r\n"
                                "p_filesz           :   0x%lx\r\n"
                                "p_memsz            :   0x%lx\r\n"
                                "p_flags            :   0x%lx\r\n"
                                "p_align            :   0x%lx\r\n",
                                ph->p_type                      ,
                                ph->p_offset                    ,
                                ph->p_va                        ,
                                ph->p_pa                        ,
                                ph->p_filesz                    ,
                                ph->p_memsz                     ,
                                ph->p_flags                     ,
                                ph->p_align                     );
        }
}

void print_elf_auto(struct elfhdr32* elf_head)
{
        if (elf_head->e_magic != ELF_MAGIC) {
                fprintf(stderr, "bad elf magic!\r\n");
                return;
        }
        switch (elf_head->e_elf[0]) {
                case 0:
                        fprintf(stderr, "bad arch %d!\r\n", elf_head->e_elf[0]);
                        return;
                case 1:
                        printf("arch32\r\n");
                        print_elf32(elf_head);
                        break;
                case 2:
                        printf("arch64\r\n");
                        print_elf64(elf_head);
                        break;
                default:
                        fprintf(stderr, "bad arch %d!\r\n", elf_head->e_elf[0]);
                        break;
        }
}



int main(int argc, char* argv[])
{
```

```
        struct stat st;
        if (argc != 2) {
                fprintf(stderr, "Usage: <input file>\n");
                return -1;
        }
        if (stat(argv[1], &st) != 0) {
                fprintf(stderr, "open file error: %s : %s \n", argv[1], strerror(errno));
                return -1;
        }
        printf("%s size: %lld bytes\n", argv[1], (long long)st.st_size);

        void* buf = malloc(st.st_size);
        memset(buf, 0, st.st_size);
        FILE* fp_in = fopen(argv[1], "rb");
        u64 size = fread(buf, 1, st.st_size, fp_in);
        if (size != st.st_size) {
                fprintf(stderr, "read %s error, size diff %ld vs. %ld\n", argv[1], st.st_size,
 size);
                return -1;
        }
        print_elf_auto(buf);
}
[src/libs/udebug.h] ++++++++++++++++++++++++++++++
#ifndef  __UDEBUG_H__
#define  __UDEBUG_H__

#define ENDIANNESS ({\
                union { char c[4]; unsigned long l; } endian_test = { { 'l', '?', '?', 'b' } }
; \
                (char)endian_test.l; \
                })

#define L2B32(little)  (((little&0xff)<<24) | ((little&0xff00)<<8) | ((little&0xff0000)>>8) |
((little&0xff000000)>>24))

// ============================================================wait if
#define wait_if(expr, cnt, desc) ({\
                udebug("wait:%d, desc:%s", cnt, desc); \
                int _i = 0; \
                for (;_i < cnt; _i++) { if (!(expr)) break; if (0 == _i%100) uclean(".");} \
                if (_i == cnt) uerror("==>[%d/%d] [timeout]\n", _i, cnt); \
                else uclean("==>[%d/%d]\n", _i, cnt); \
                _i;\
                })
// ============================================================wait if

// ============================================================log
#define LEVEL_DEBUG  0
#define LEVEL_INFO   1
#define LEVEL_ERROR  2
#define LEVEL_OFF    3
#define LEVEL_SIMPLE 4
#define ULOG_LEVEL LEVEL_DEBUG
//#define ULOG_LEVEL LEVEL_OFF

#define printf cprintf

#if ULOG_LEVEL == LEVEL_OFF
#define udebug(fmt, args...)
#define uinfo(fmt, args...)
#define uerror(fmt, args...)
#define uclean(fmt, args...)
#define ulog(fmt, args...)
#elif ULOG_LEVEL == LEVEL_DEBUG
#define udebug(fmt, args...) printf("[D][%s:%d][%s] " fmt, __FILE__, __LINE__, __FUNCTION__, #
```

```
#args)
#define uinfo(fmt, args...) printf("[I][%d][%s] " fmt, __LINE__, __FUNCTION__, ##args)
#define uerror(fmt, args...) printf("[E][%d][%s] " fmt, __LINE__, __FUNCTION__, ##args)
#define uclean printf
#define ulog uinfo
#elif ULOG_LEVEL == LEVEL_INFO
#define udebug(fmt, args...)
#define uinfo(fmt, args...) printf("[I][%d][%s] " fmt, __LINE__, __FUNCTION__, ##args)
#define uerror(fmt, args...) printf("[E][%d][%s] " fmt, __LINE__, __FUNCTION__, ##args)
#define uclean printf
#define ulog uinfo
#elif ULOG_LEVEL == LEVEL_ERROR
#define udebug(fmt, args...)
#define uinfo(fmt, args...)
#define uerror(fmt, args...) printf("[E][%d][%s] " fmt, __LINE__, __FUNCTION__, ##args)
#define uclean printf
#define ulog uinfo
#elif ULOG_LEVEL == LEVEL_SIMPLE
#define udebug(fmt, args...) printf("[D][%d] " fmt, __LINE__, ##args)
#define uinfo(fmt, args...) printf("[I][%d] " fmt, __LINE__, ##args)
#define uerror(fmt, args...) printf("[E][%d] " fmt, __LINE__, ##args)
#define uclean printf
#define ulog uinfo
#else
#define udebug printf
#define uinfo printf
#define uerror printf
#define uclean printf
#define ulog printf
#endif

#endif  /* __UDEBUG_H__ */
[src/libs/string.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_STRING_H__
#define __LIBS_STRING_H__

#include <libs/defs.h>

size_t strlen(const char *s);
size_t strnlen(const char *s, size_t len);
u64 str2n(const char* s);

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
[src/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__ */

[src/libs/dirent.h] ++++++++++++++++++++++++++++++
[src/libs/stdio.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_STDIIO_H__
#define __LIBS_STDIIO_H__

#include <libs/defs.h>
#include <libs/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
[src/libs/string.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>

size_t strlen(const char* s)
{
        size_t cnt = 0;
        while (*s++ != '\0') {
                ++cnt;
        }
```

```
        return cnt;
}

size_t strnlen(const char* s, size_t len)
{
        size_t cnt = 0;
        while (cnt < len && *s++ != '\0') {
                ++cnt;
        }
        return cnt;
}

u64 str2n(const char* s)
{
        u64 ret = 0;
        int base = 10;
        if (s != 0 && *s == '0' && *(s+1) == 'x')
        {
                base=16;
                s += 2;
        }

        while(*s != '\0')
        {
                ret *= base;
                if (base == 10)
                {
                        ret += (*s - '0');
                } else if (base == 16) {
                        if ('0' <= *s && *s <= '9') {
                                ret += (*s - '0');
                        } else if ('a' <= *s && *s <= 'f') {
                                ret += (*s - 'a' + 10);
                        } else if ('A' <= *s && *s <= 'F') {
                                ret += (*s - 'A' + 10);
                        } else {
                                return ret;
                        }
                }
                s++;
        }
        return ret;
}

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

char* strcpy(char* dst, const char* src)
{
#ifdef __HAVE_ARCH_STRCPY
        return __strcpy(dst, src);
#else
        char* p = dst;
        while ((*p++ = *src++) != '\0')
                ;
        return dst;
#endif /* !__HAVE_ARCH_STRCPY */
}

char* strncpy(char* dst, const char* src, size_t len)
{
        char* p = dst;
        while (len > 0) {
```

```
                if ((*p++ = *src) != '\0') {
                        src++;
                }
                --len;
        }
        return dst;
}

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
}

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);
}

char* strchr(const char* s, char c)
{
        while (*s != '\0') {
                if (*s == c) {
                        return (char*)s;
                }
                ++s;
        }
        return NULL;
}

char* strfind(const char* s, char c)
{
        while (*s != '\0') {
                if (*s == c) {
                        break;
                }
                ++s;
        }
        return (char*)s;
}

/* convert string to long interger */
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
        for (;;) {
                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;
        }
        if (endptr) {
                *endptr = (char*)s;
        }
        return (neg ? -val : val);
}

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 */
}

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 */
}

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 */
}

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;
}
[src/libs/list.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_LIST_H__
#define __LIBS_LIST_H__

#ifndef __ASSEMBLER__
#include <libs/defs.h>

/*
 * simple doubly linked list implementation.
 * */

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_inli
ne));
static inline void list_add_before(list_entry_t *listelm, list_entry_t *elm) __attribute((alwa
ys_inline));
static inline void list_add_after(list_entry_t *listelm, list_entry_t *elm) __attribute((alway
s_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) __at
tribute((always_inline));
static inline void __list_del(list_entry_t * prev, list_entry_t* next) __attribute((always_inl
ine));


static inline void list_init(list_entry_t* elm)
{
```

```
        elm->prev = elm->next = elm;
}

static inline void __list_add(list_entry_t* insert_elm, list_entry_t* prev, list_entry_t* next
)
{
        prev->next = insert_elm;
        next->prev = insert_elm;
        insert_elm->prev = prev;
        insert_elm->next = next;
}

static inline void list_add_before(list_entry_t* listelm, list_entry_t* insert_elm)
{
        __list_add(insert_elm, listelm->prev, listelm);
}

static inline void list_add_after(list_entry_t* listelm, list_entry_t* insert_elm)
{
        __list_add(insert_elm, listelm, listelm->next);
}

static inline void list_add(list_entry_t* listelm, list_entry_t* insert_elm)
{
        list_add_after(listelm, insert_elm);
}

static inline void __list_del(list_entry_t* prev, list_entry_t* next)
{
        prev->next=next;
        next->prev=prev;
}

static inline void list_del(list_entry_t* listelm)
{
        __list_del(listelm->prev, listelm->next);
}

static inline void list_del_init(list_entry_t* listelm)
{
        list_del(listelm);
        list_init(listelm);
}

static inline bool list_empty(list_entry_t* list)
{
        return list->next == list;
}

static inline list_entry_t* list_next(list_entry_t* listelm)
{
        return listelm->next;
}

static inline list_entry_t* list_prev(list_entry_t* listelm)
{
        return listelm->prev;
}

#endif /* !__ASSEMBLER__ */
#endif /* !__LIBS_LIST_H__ */

[src/libs/printfmt.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.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;
}


[src/libs/elf.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_ELF_H__
#define __LIBS_ELF_H__
#include <libs/defs.h>

#define ELF_MAGIC 0x464C457FU

// 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

struct elfhdr {
    u32 e_magic; // must equal ELF_MAGIC
    u8 e_elf[12];
    u16 e_type;   // 1=relocatable, 2=executable, 3=shared object, 4=core image
    u16 e_machine;         // 3=x86, 4=68K, etc.
    u32 e_version;         // file version = 1
    u32 e_entry;           // entry point if executable
    u32 e_phoff;           // program header offset
    u32 e_shoff;           // section header offset
    u32 e_flags;           // architecture-specific flags = 0
    u16 e_ehsize;          // elf header size
    u16 e_phentsize; // program header entry size
    u16 e_phnum;           // program header number
    u16 e_shentsize; // section header entry size
    u16 e_shnum;           // section header number
    u16 e_shstrndx;  // section header name string index
};

struct proghdr {
    u32 p_type;   // loadable code or data, dynamic linking info, etc.
    u32 p_offset; // file segment offset
    u32 p_va;     // virtual address to map segment
    u32 p_pa;     // physical address, not used
    u32 p_filesz; // size of segment in file
    u32 p_memsz;  // size of segment in memory (bigger if contains bss)
    u32 p_flags;  // read/write/execut bits
    u32 p_align;  // required alignment, invariably hardware page size
```

```
};

#endif
[src/libs/rand.c] ++++++++++++++++++++++++++++++
[src/libs/defs.h] ++++++++++++++++++++++++++++++
#ifndef  __DEFS_H__
#define  __DEFS_H__

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

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

#define CHAR_BIT 8
typedef int bool;
typedef char i8;
typedef unsigned char u8;
typedef short i16;
typedef unsigned short u16;
typedef int i32;
typedef unsigned int u32;
typedef long long i64;
typedef unsigned long long u64;

/*
 * Pointers and addresses are 32-bit long.
 * Use pointer types to represent addresses,
 * uintptr_t to represent the numerical values of addresses.
 * */
typedef i32 intptr_t;
typedef u32 uintptr_t;

/*size_t is used for memory object size*/
typedef uintptr_t size_t;

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

/* used fo rpage numbers */
typedef size_t ppn_t;

/*round up + round down + round up div*/
#define ROUNDDOWN(a, n)                 \
        ({                                      \
        size_t __a = (size_t)(a);       \
        (typeof(a))(__a - __a % (n)); \
        })

#define ROUNDUP(a, n)                                           \
        ({                                                          \
        size_t __n = (size_t)(n);                               \
        (typeof(a))(ROUNDDOWN((size_t)a + __n - 1, __n)); \
        })

#define ROUNDUP_DIV(a, n)                       \
        ({                                          \
        size_t __n = (size_t n);                \
        (typeof(a))(((a) + __n - 1) / __n); \
        })

/*
 * Get the struct pointer from a member pointer and member type
```

```
 * */
#define offsetof(type, member) \
        ((size_t)(&((type*)0)->member))

#define to_struct(ptr, type, member) \
        ((type*)((char*)(ptr)-offsetof(type, member)))

#endif  /* __DEFS_H__ */
[src/libs/stat.h] ++++++++++++++++++++++++++++++
[src/libs/hash.c] ++++++++++++++++++++++++++++++
[src/libs/x86.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_X86_H__
#define __LIBS_X86_H__

#include <libs/defs.h>

struct pseudodesc {
        u16 pd_lim;   // limit
        uintptr_t pd_base; // base address
} __attribute__((packed));

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

#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;
                       \
                                                                })

static inline void* __memset(void* s, char c, size_t n) __always_inline;
static inline void* __memmove(void* dst, const void* src, size_t n) __always_inline;
static inline void* __memcpy(void* dst, const void* src, size_t n) __always_inline;
static inline u32 read_ebp(void) __always_inline;


static inline u8 inb(u16 port)
{
        u8 data;
        asm volatile("inb %1, %0"
                        : "=a"(data)
                        : "d"(port)
                        : "memory");
        return data;
}
```

```
static inline u16 inw(u16 port)
{
        u16 data;
        asm volatile("inw %1, %0"
                        : "=a"(data)
                        : "d"(port));
        return data;
}

static inline void insl(u32 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(i16 port, u8 data)
{
        asm volatile("outb %0, %1" ::"a" (data), "d" (port)
                        : "memory");
}

static inline void outw(i16 port, u16 data)
{
        asm volatile("outw %0, %1" ::"a"(data), "d"(port)
                        : "memory");
}

static inline void outsl(u32 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 u32 read_esp(void)
{
        u32 esp;
        asm volatile("movl %%esp, %0" : "=r" (esp));
        return esp;
}
static inline u32 read_ebp(void)
{
        u32 ebp;
        asm volatile("movl %%ebp, %0" : "=r" (ebp));
        return ebp;
}

static inline u32 read_dr(unsigned regnum)
{
        u32 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 inline void write_dr(unsigned regnum, u32 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 breakpoint(void)
{
        asm volatile("int $3");
}

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(u16 sel)
{
        asm volatile("ltr %0" ::"r"(sel)
                     : "memory");
}

static inline u32 read_eflags(void)
{
        u32 eflags;
        asm volatile("pushfl; popl %0"
                     : "=r"(eflags));
        return eflags;
}

static inline void write_eflags(u32 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");
}

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


#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*/

#endif /*__LIBS_X86_H__*/
[src/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__ */

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

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)

#endif
[src/libs/stdlib.h] ++++++++++++++++++++++++++++++
[src/libs/atomic.h] ++++++++++++++++++++++++++++++
#ifndef __LIBS_ATOMIC_H__
#define __LIBS_ATOMIC_H__

#include "defs.h"
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));

static inline void set_bit(int nr, volatile void *addr)
{
```

```
        asm volatile ("btsl %1, %0" : "=m" (*(volatile long *)addr) : "Ir" (nr));
}

static inline void clear_bit(int nr, volatile void *addr)
{
        asm volatile ("btrl %1, %0" :"=m" (*(volatile long *)addr): "Ir" (nr));
}

static inline void change_bit(int nr, volatile void *addr)
{
        asm volatile ("btcl %1, %0" :"=m" (*(volatile long *)addr): "Ir" (nr));
}

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;
}

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;
}

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;
}

#endif /* !__LIBS_ATOMIC_H__ */
[src/libs/skew_heap.h] ++++++++++++++++++++++++++++++
[src/libs/libs_all.h] ++++++++++++++++++++++++++++++
#ifndef  __LIBS_ALL_H__
#define  __LIBS_ALL_H__
#include <libs/atomic.h>
#include <libs/defs.h>
#include <libs/dirent.h>
#include <libs/elf.h>
#include <libs/error.h>
#include <libs/libs_all.h>
#include <libs/list.h>
#include <libs/skew_heap.h>
#include <libs/stat.h>
#include <libs/stdarg.h>
#include <libs/stdio.h>
#include <libs/stdlib.h>
#include <libs/string.h>
#include <libs/udebug.h>
#include <libs/unistd.h>
#include <libs/x86.h>
#endif  /* __LIBS_ALL_H__ */
[src/kern/driver/clock.c] ++++++++++++++++++++++++++++++
#include <kern/trap/trap.h>

volatile size_t ticks = 0;
[src/kern/driver/console.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_DRIVER_CONSOLE_H__
```

```
#define __KERN_DRIVER_CONSOLE_H__

/***** 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


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

#endif
[src/kern/driver/ide.c] ++++++++++++++++++++++++++++++
[src/kern/driver/picirq.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>
#include <kern/driver/picirq.h>

// I/O addresses of the two programmable interrupt controllers
#define IO_PIC1 0x20 // master (IRQs 0-7)
#define IO_PIC2 0xA0 // master (IRQs 0-7)

#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 u16 irq_mask = 0xFFFF & ~(1 << IRQ_SLAVE);
static bool did_init = 0;

static void pic_setmask(u16 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));
}

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 trig, 1 = level trig
        //  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 does not tend to work on 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);
        }
}
[src/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__ */
[src/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__ */

[src/kern/driver/intr.c] ++++++++++++++++++++++++++++++
#include <kern/driver/intr.h>
#include <libs/libs_all.h>


void intr_enable(void)
{
        sti();
}

void intr_disable(void)
{
        cli();
}
[src/kern/driver/clock.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_DRIVER_CLOCK_H__
#define __KERN_DRIVER_CLOCK_H__

#include <libs/defs.h>

extern volatile size_t ticks;


#endif /* !__KERN_DRIVER_CLOCK_H__ */


[src/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__ */
[src/kern/driver/console.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>
#include <kern/debug/assert.h>
#include <kern/driver/console.h>
#include <kern/driver/kbdreg.h>
#include <kern/mm/memlayout.h>
#include <kern/driver/picirq.h>
```

```
#include <kern/trap/trap.h>
#include <kern/sync/sync.h>

static u16* crt_buf;
static u16 crt_pos;
static u16 addr_6845;

static void delay(void)
{
        inb(0x84);
        inb(0x84);
        inb(0x84);
        inb(0x84);
}

static void cga_init(void)
{
        volatile u16* cp = (u16*)(CGA_BUF + KERNBASE);
        u16 was = *cp;
        *cp = (u16)0xA55A;
        if (*cp != 0xA55A) {
                cp = (u16*)(MONO_BUF + KERNBASE);
                addr_6845 = MONO_BASE;
        } else {
                *cp = was;
                addr_6845 = CGA_BASE;
        }

        // extract cursor location
        u32 pos;
        outb(addr_6845, 14);
        pos = inb(addr_6845 + 1) << 8;
        outb(addr_6845, 15);
        pos |= inb(addr_6845 + 1);

        crt_buf = (u16*)cp;
        crt_pos = pos;
}

static bool serial_exists = 0;

void serial_init(void)
{
        // turn off FIFO
        outb(COM1 + COM_FCR, 0);

        // set speed: require DLAB latch
        outb(COM1 + COM_LCR, COM_LCR_DLAB);
        outb(COM1 + COM_DLL, (u8)(115200 / 9600));
        outb(COM1 + COM_DLM, 0);

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

        // no modem control
        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(u16));
                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)
{
        for (int i = 0; !(inb(COM1 + COM_LSR) & COM_LSR_TXRDY) && i < 12800; i++) {
                delay();
        }
        outb(COM1 + COM_TX, c);
}

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');
        }
}

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

#define CONSBUFFSIZE 512

static struct {
        u8 buf[CONSBUFFSIZE];
        u32 rpos;
        u32 wpos;
} cons;

/*
 * called by device interrupt routines to feed input
 * characters into 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 == CONSBUFFSIZE) {
                                cons.wpos = 0;
                        }
                }
        }
}

/* 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;
```

```
}

/* 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 NUMLOACK (1 << 4)
#define SCROLLLOCK (1 << 5)
#define E0ESC (1 << 6)

static u8 shiftcode[256] = {
        [0x1D] CTL,
        [0x2A] SHIFT,
        [0x36] SHIFT,
        [0x38] ALT,
        [0x2A] CTL,
        [0x2A] ALT,
};

static u8 togglecode[256] = {
        [0x3A] CAPSLOCK,
        [0x45] NUMLOACK,
        [0x46] SCROLLLOCK,
};

static u8 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 u8 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 u8 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 u8* charcode[4] = {
        normalmap,
        shiftmap,
        ctlmap,
        ctlmap,
};

static int kbd_proc_data(void)
{
        int c;
        u8 data;
        static u32 shift;
        if ((inb(KBSTATP) & KBS_DIB) == 0) {
                return -1;
        }
        data = inb(KBDATAP);
        if (data == 0xE0) {

        } else if (data & 0x80) {

        } else if (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 sepcial keys: CTRL-ALT-DEL: reboot
        if (!(~shift & (CTL | ALT)) && c == KEY_DEL) {
                cprintf("rebooting...\n");
                outb(0x92, 0x3);
```

```
        }
        return c;
}

static void kbd_intr(void)
{
        cons_intr(kbd_proc_data);
}

static void kbd_init(void)
{
        kbd_intr();
        pic_enable(IRQ_KBD);
}

void cons_init(void)
{
        cga_init();
        serial_init();
        kbd_init();
        if (!serial_exists) {
                cprintf("serial port does not exist!\n");
        }
}

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);
}

int cons_getc(void)
{
        int c = 0;
        bool intr_flag;
        local_intr_save(intr_flag);
        {
                serial_intr();
                //kbd_intr();
                if (cons.rpos != cons.wpos) {
                        c = cons.buf[cons.rpos++];
                        if (cons.rpos == CONSBUFFSIZE) {
                                cons.rpos = 0;
                        }
                }
        }
        local_intr_restore(intr_flag);
        return c;
}
[src/kern/driver/ide.h] ++++++++++++++++++++++++++++++
[src/kern/process/entry.S] ++++++++++++++++++++++++++++++
[src/kern/process/proc.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_PROCESS_PROC_H__
#define __KERN_PROCESS_PROC_H__

#include <libs/defs.h>
#include <libs/list.h>
#include <kern/trap/trap.h>
#include <kern/mm/memlayout.h>
```

```
#include <libs/skew_heap.h>

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

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

struct inode;

enum proc_state {
        PROC_UNINIT = 0,
        PROC_SLEEPING,
        PROC_RUNNABLE,
        PROC_ZOMBIE,
};

struct context {
        u32 eip;
        u32 esp;
        u32 ebx;
        u32 ecx;
        u32 edx;
        u32 esi;
        u32 edi;
        u32 ebp;
};

struct proc_struct {
        char name[PROC_NAME_LEN + 1];
        struct proc_struct *parent;
        struct proc_struct *cptr, *yptr, *optr;
        enum proc_state state;
        int pid;
        int runs;
        uintptr_t kstack;
        volatile bool need_resched;
        struct mm_struct *mm;
        struct context context;
        struct trapframe *tf;
        uintptr_t cr3;
        u32 flags;
        list_entry_t list_link;
        list_entry_t hash_link;
        list_entry_t run_link;
        int exit_code;
        u32 wait_state;
        struct run_queue *rq;
        int time_slice;
        //skew_heap_entry_t lab6_run_pool;
        u32 lab6_stride;
        u32 lab6_priority;
        struct files_struct *filesp;
};

void cpu_idle(void) __attribute__((noreturn));


#endif /*__KERN_PROCESS_PROC_H__*/
[src/kern/process/proc.c] ++++++++++++++++++++++++++++++
#include <proc.h>
//
struct proc_struct *current = NULL;
//
//void cpu_idle(void)
```

```
//{
//      while(1)
//      {
//              if (current->need_resched)
//              {
//                      schedule();
//              }
//      }
//}
[src/kern/process/switch.S] ++++++++++++++++++++++++++++++
[src/kern/init/entry.S] ++++++++++++++++++++++++++++++
#include <kern/mm/mmu.h>
#include <kern/mm/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: bootstack(bootstacktop) ~ +KSTACKSIZE(8KB) (in memlayout.h)
        movl $bootstacktop, %esp
        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)
.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)
        # pad to PDE of KERNBASE
        .space (KERNBASE >> PGSHIFT >> 10 << 2) - (. - __boot_pgdir)
        # map va KERNBASE + (0-4M) to pa 0-4M
```

```
        .long REALLOC(__boot_pt1) + (PTE_P | PTE_U | PTE_W)
        # pad to PGSIZE
        .space PGSIZE - (. - __boot_pgdir)

.set i,0
__boot_pt1:
.rept 1024
        .long i * PGSIZE + (PTE_P | PTE_W)
        .set i, i + 1
.endr

[src/kern/init/init.c] ++++++++++++++++++++++++++++++
/* ***************************************************************************************
 * FILE NAME   : init.c
 * PROGRAMMER  : zhaozz
 * DESCRIPTION : kernel entry
 * DATE        : 2022-01-08 00:38:27
 * ***************************************************************************************/
#include <libs/libs_all.h>
#include <kern/debug/kdebug.h>
#include <kern/debug/kcommand.h>
#include <kern/driver/clock.h>
#include <kern/driver/console.h>
#include <kern/driver/ide.h>
#include <kern/driver/intr.h>
#include <kern/driver/picirq.h>
#include <kern/mm/pmm.h>
#include <kern/mm/swap.h>
#include <kern/mm/vmm.h>
#include <kern/process/proc.h>
#include <kern/trap/trap.h>

//http://patorjk.com/software/taag
static char *welcome ="\n"
" _______     ______     ______   \n"
"/\\___  \\   /\\  __ \\   /\\  ___\\   \n"
"\\/_/  /__  \\ \\ \\/\\ \\  \\ \\___  \\  \n"
"  /\\_____\\  \\ \\_____\\  \\/\\_____\\ \n"
"  \\/_____/   \\/_____/   \\/_____/ \n"
"                                ";

int kern_init(void) __attribute__((noreturn));
extern char bootstacktop[], bootstack[];
extern char edata[], end[];

int kern_init(void)
{
        memset(edata, 0, end - edata);
        cons_init();

        cprintf("%s\n", welcome);
        cprintf("bootstack:0x%x, bootstacktop:0x%x\n", bootstack, bootstacktop);
        cprintf("edata:0x%x, end:0x%x\n", edata, end);

        print_kerninfo();
        print_stackframe();

        pmm_init();
        pic_init();
        idt_init();                 // init interrupt descriptor table


        //vmm_init();
        //sched_init();
        //proc_init();
```

```
        //ide_init();
        //swap_init();
        //fs_init();

        //clock_init();
        //intr_enable();

        //cpu_idle();
        while(1) KCMD_LOOP;
}
[src/kern/syscall/syscall.c] ++++++++++++++++++++++++++++++
[src/kern/syscall/syscall.h] ++++++++++++++++++++++++++++++
[src/kern/sync/monitor.c] ++++++++++++++++++++++++++++++
[src/kern/sync/wait.h] ++++++++++++++++++++++++++++++
[src/kern/sync/wait.c] ++++++++++++++++++++++++++++++
[src/kern/sync/sem.c] ++++++++++++++++++++++++++++++
[src/kern/sync/sync.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_SYNC_SYNC_H__
#define __KERN_SYNC_SYNC_H__

#include <kern/driver/intr.h>
#include <kern/mm/mmu.h>
#include <libs/x86.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__ */
[src/kern/sync/sem.h] ++++++++++++++++++++++++++++++
[src/kern/sync/check_sync.c] ++++++++++++++++++++++++++++++
[src/kern/sync/monitor.h] ++++++++++++++++++++++++++++++
[src/kern/libs/readline.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>


#define BUFSIZE 1024
static char buf[BUFSIZE];

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('\n');
                        buf[i] = '\0';
                        return buf;
                }
        }
}


[src/kern/libs/stdio.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>
#include <kern/debug/assert.h>
#include <kern/driver/console.h>

/* high level console I/O */

static void cputch(int c, int* cnt)
{
        cons_putc(c);
        (*cnt)++;
}

int vcprintf(const char* fmt, va_list ap)
{
        int cnt = 0;
        vprintfmt((void*)cputch, NO_FD, &cnt, fmt, ap);
        return cnt;
}

int cprintf(const char* fmt, ...)
{
        va_list ap;
        int cnt;
        va_start(ap, fmt);
        cnt = vcprintf(fmt, ap);
        va_end(ap);
        return cnt;
}

void cputchar(int c)
{
        cons_putc(c);
}

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

int getchar(void)
{
        int c;
        while ((c = cons_getc()) == 0)
                ; // do nothing
```

```
        return c;
}
[src/kern/trap/trapentry.S] ++++++++++++++++++++++++++++++
#include <kern/mm/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

[src/kern/trap/trap.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_TRAP_TRAP_H__
#define __KERN_TRAP_TRAP_H__

#include <libs/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 {
        u32 reg_edi;
        u32 reg_esi;
        u32 reg_ebp;
        u32 reg_oesp;           /* Useless */
        u32 reg_ebx;
        u32 reg_edx;
        u32 reg_ecx;
        u32 reg_eax;
};

struct trapframe {
        struct pushregs tf_regs;
        u16 tf_gs;
        u16 tf_padding0;
        u16 tf_fs;
        u16 tf_padding1;
        u16 tf_es;
        u16 tf_padding2;
        u16 tf_ds;
        u16 tf_padding3;
        u32 tf_trapno;
        /* below here defined by x86 hardware */
        u32 tf_err;
        uintptr_t tf_eip;
        u16 tf_cs;
        u16 tf_padding4;
        u32 tf_eflags;
```

```
        /* below here only when crossing rings, such as from user to kernel */
        uintptr_t tf_esp;
        u16 tf_ss;
        u16 tf_padding5;
} __attribute__((packed));

void idt_init(void);
void print_trapframe(struct trapframe *tf);
void print_regs(struct trapframe *tf);
bool trap_in_kernel(struct trapframe *tf);

#endif /* !__KERN_TRAP_TRAP_H__ */
```

[src/kern/trap/trap.c] ++++++++++++++++++++++++++++++

```
#include "trap.h"
#include <libs/libs_all.h>
#include <kern/debug/assert.h>
#include <kern/driver/clock.h>
#include <kern/driver/console.h>
#include <kern/debug/kdebug.h>
#include <kern/mm/memlayout.h>
#include <kern/mm/mmu.h>
#include <kern/process/proc.h>
#include <kern/schedule/sched.h>
#include <kern/mm/swap.h>
#include <kern/sync/sync.h>
#include <kern/syscall/syscall.h>
#include <kern/trap/trap.h>
#include <kern/mm/vmm.h>

#define TICK_NUM 100

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,
};

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"
};

/* *
 * 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
};

static volatile int in_swap_tick_event = 0;
extern struct mm_struct* check_mm_struct;

void print_ticks()
{
        cprintf("%d ticks\n", TICK_NUM);
}

/* idt_init - initialize IDT to each of the entry points in kern/trap/vectors.S */
void idt_init(void)
{
        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)
{
        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 == (u16)KERNEL_CS);
```

```
}


void print_trapframe(struct trapframe* tf)
{
        cprintf("trapframe at %p\n", tf);
        print_regs(tf);
        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 trapframe* tf)
{
        cprintf("  edi  0x%08x\n", tf->tf_regs.reg_edi);
        cprintf("  esi  0x%08x\n", tf->tf_regs.reg_esi);
        cprintf("  ebp  0x%08x\n", tf->tf_regs.reg_ebp);
        cprintf("  oesp 0x%08x\n", tf->tf_regs.reg_oesp);
        cprintf("  ebx  0x%08x\n", tf->tf_regs.reg_ebx);
        cprintf("  edx  0x%08x\n", tf->tf_regs.reg_edx);
        cprintf("  ecx  0x%08x\n", tf->tf_regs.reg_ecx);
        cprintf("  eax  0x%08x\n", tf->tf_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 & PTE_U) ? 'U' : 'K',
                        (tf->tf_err & PTE_W) ? 'W' : 'R',
                        (tf->tf_err & PTE_P) ? "protection fault" : "no page found");
}

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());
        return 0;
}


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. r
et=%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:
                //      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;
                        //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)
{
#if 0
        // 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();
                        }
                }
        }
#endif
}
[src/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
[src/kern/debug/panic.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>
#include <kern/debug/kdebug.h>
#include <kern/debug/kcommand.h>
#include <kern/driver/intr.h>

static bool is_panic = 0;

void __panic(const char* file, int line, const char* fmt, ...)
{
        if (is_panic) {
                goto panic_dead;
        }
        is_panic = 1;

        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) {
                kcmd_loop();
        }
}

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;
}
[src/kern/debug/kcommand.h] ++++++++++++++++++++++++++++++
```

```
#ifndef  __KCOMMAND_H__
#define  __KCOMMAND_H__
#include <kern/trap/trap.h>
#include <kern/mm/pmm.h>

void kcmd_loop();
int cmd_help(int argc, char **argv);
int cmd_kerninfo(int argc, char **argv);
int cmd_backtrace(int argc, char **argv);
int cmd_exit(int argc, char **argv);
int cmd_jump(int argc, char **argv);
int cmd_mem(int argc, char **argv);
int cmd_print_pg(int argc, char **argv);
int cmd_print_free_pages(int argc, char **argv);
int cmd_call(int argc, char **argv);

#define KCMD_LOOP { \
        uinfo(""); \
        kcmd_loop(); \
}

#endif  /* __KCOMMAND_H__ */
[src/kern/debug/kcommand.c] ++++++++++++++++++++++++++++++
#include <libs/stdio.h>
#include <libs/string.h>
#include <kern/mm/mmu.h>
#include <kern/mm/default_pmm.h>
#include <kern/trap/trap.h>
#include <kern/debug/kcommand.h>
#include <kern/debug/kdebug.h>

bool is_kernel_panic(void);

#define COMMAND_MAX    200
#define MAXARGS         16
#define WHITESPACE      " \t\n\r"

enum CMD_RETURN_CODE {
        CMD_EXIT = -1,
        CMD_SUCCEED = 0,
        CMD_FAILED,
        CMD_NOT_SUPPORT,
        CMD_BAD_ARGS,
        CMD_NULL,
};

struct command {
        const char *name;
        const char *desc;
        int argc;  //-1: not check; other: must equal to argc
        int(*func)(int argc, char **argv);
};

static struct command commands[COMMAND_MAX] = {
        {"help", "Display this list of commands", 1, cmd_help},
        {"kerninfo", "Display information about the kernel", 1, cmd_kerninfo},
        {"backtrace", "Print backtrace of stack frame", 1, cmd_backtrace},
        {"exit", "exit console", 1, cmd_exit},
        {"jump", "jump addr", 2, cmd_jump},
        {"call", "call addr", 2, cmd_call},
        {"mem", "print memory", 1, cmd_mem},
        {"page", "print page table", 1, cmd_print_pg},
        {"free_page", "print free pages", 1, cmd_print_free_pages},
        {0, 0, 0, 0},
};
```

```
int get_commands_len()
{
        int i=0;
        while(commands[i].name != 0) ++i;
        return i;
}

void append_command(struct command cmdone)
{
        int len = get_commands_len();
        commands[len] = cmdone;
        commands[len+1].name = 0;
        commands[len+1].argc = 0;
        commands[len+1].desc = 0;
        commands[len+1].func = 0;
}

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;
}

static int runcmd(char *buf)
{
        char *argv[MAXARGS];
        int argc = parse(buf, argv);
        if (argc == 0) {
                return CMD_NULL;
        }
        int i;
        for (i = 0; i < get_commands_len(); i ++) {
                if (strcmp(commands[i].name, argv[0]) == 0) {
                        if (argc != commands[i].argc)
                        {
                                return CMD_BAD_ARGS;
                        }
                        return commands[i].func(argc, argv);
                }
        }
        return CMD_NOT_SUPPORT;
}


void kcmd_loop()
{
```

```
        cprintf("Type 'help' for a list of commands.\n");
        char *buf;
        static int index = 0;
        char promt_buf[64] = {0};
        while (1) {
                snprintf(promt_buf, 64, "[ksh:%d]$ ", ++index);
                if ((buf = readline(promt_buf)) != NULL) {
                        enum CMD_RETURN_CODE ret_code = runcmd(buf);
                        if (ret_code == CMD_EXIT)
                        {
                                cprintf("exit!\n");
                                return;
                        } else if (ret_code == CMD_NOT_SUPPORT) {
                                cprintf("not support!\n");
                        } else if (ret_code == CMD_BAD_ARGS) {
                                cprintf("bad args!\n");
                        } else {
                                continue;
                        }
                }
        }
}

/* mon_help - print the information about mon_* functions */
int cmd_help(int argc, char **argv)
{
        int i;
        for (i = 0; i < get_commands_len(); i ++) {
                cprintf("%s - %s\n", commands[i].name, commands[i].desc);
        }
        return CMD_SUCCEED;
}

int cmd_kerninfo(int argc, char **argv)
{
        print_kerninfo();
        return CMD_SUCCEED;
}

/* *
 * mon_backtrace - call print_stackframe in kern/debug/kdebug.c to
 * print a backtrace of the stack.
 * */
int cmd_backtrace(int argc, char **argv)
{
        print_stackframe();
        return CMD_SUCCEED;
}

int cmd_exit(int argc, char **argv)
{
        return CMD_EXIT;
}

int cmd_jump(int argc, char **argv)
{
        cprintf("jump to %s\n", argv[1]);
        u32 addr = str2n(argv[1]);
        asm volatile("jmp *%0\n" ::"r"(addr));
        return CMD_SUCCEED;
}

int cmd_call(int argc, char **argv)
{
        cprintf("call to %s\n", argv[1]);
```

```
        u32 addr = str2n(argv[1]);
        asm volatile("call *%0\n" ::"r"(addr));
        return CMD_SUCCEED;
}

int cmd_mem(int argc, char **argv)
{
        print_mem();
        return CMD_SUCCEED;
}

int cmd_print_pg(int argc, char **argv)
{
        print_pg();
        return CMD_SUCCEED;
}

int cmd_print_free_pages(int argc, char **argv)
{
        print_free_pages();
        return CMD_SUCCEED;
}


[src/kern/debug/kdebug.h] ++++++++++++++++++++++++++++++
#ifndef  __KDEBUG_H__
#define  __KDEBUG_H__

void print_kerninfo(void);
void print_stackframe(void);

#endif  /* __KDEBUG_H__ */
[src/kern/debug/assert.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_DEBUG_ASSERT_H__
#define __KERN_DEBUG_ASSERT_H__

#include <kern/driver/console.h>
#include <libs/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)

#define static_assert(x) \
        switch (x) {          \
                case 0:               \
                case (x):;            \
        }

#endif /* __KERN_DEBUG_ASSERT_H__ */
[src/kern/debug/kdebug.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>
#include <kern/debug/assert.h>
#include <kern/debug/kdebug.h>
#include <kern/debug/kcommand.h>
```

```
#include <kern/mm/memlayout.h>
#include <kern/process/proc.h>
#include <kern/debug/stab.h>
#include <kern/sync/sync.h>
#include <kern/mm/vmm.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__;
        }
#if 0
        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)sta
bs, 0)) {
                        return -1;
                }
                if (!user_mem_check(mm, (uintptr_t)stabstr, stabstr_end - stabstr, 0)) {
                        return -1;
                }
        }
#endif

        // 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", ((u32)end - (u32)kern_init + 102
3) / 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);
        }
}

u32 read_eip(void)
{
        u32 eip;
        asm volatile("movl 4(%%ebp), %0" : "=r" (eip));
        return eip;
}

/* *
 * 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_stack(u32 addr, u32 len)
{
        while(len--)
        {
                cprintf("addr:0x%08x ", addr);
                cprintf("val:0x%08x \n", *((u32*)addr));
                addr += sizeof(addr);
        }
}

void print_stackframe(void)
{
        u32 t_ebp = read_ebp();
        u32 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);
                u32* args = (u32*)t_ebp + 2;
                for (j = 0; j < 4; j++) {
                        cprintf("0x%08x ", args[j]);
                }
                print_debuginfo(t_eip - 1);
                t_eip = ((u32 *)t_ebp)[1];
                t_ebp = ((u32 *)t_ebp)[0];
        }
}

[src/kern/debug/stab.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_DEBUG_STAB_H__
#define __KERN_DEBUG_STAB_H__

#include <libs/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 {
    u32 n_strx;         // index into string table of name
    u8 n_type;          // type of symbol
    u8 n_other;         // misc info (usually empty)
    u16 n_desc;         // description field
    uintptr_t n_value;      // value of symbol
};

#endif /* !__KERN_DEBUG_STAB_H__ */


[src/kern/mm/swap.c] ++++++++++++++++++++++++++++++
#include "swap.h"
volatile int swap_init_ok = 0;

[src/kern/mm/kmalloc.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_SLAB_H__
#define __KERN_MM_SLAB_H__

#include <libs/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__ */
[src/kern/mm/default_pmm.h] ++++++++++++++++++++++++++++++
#ifndef  __DEFAULT_PMM_H__
#define  __DEFAULT_PMM_H__

#include <kern/mm/pmm.h>

extern const struct pmm_manager default_pmm_manager;
extern free_area_t free_area;
void print_free_pages();

#endif  /* __DEFAULT_PMM_H__ */
[src/kern/mm/swap.h] ++++++++++++++++++++++++++++++
#ifndef  __SWAP_H__
#define  __SWAP_H__

extern volatile int swap_init_ok;


#endif  /* __SWAP_H__ */
[src/kern/mm/vmm.c] ++++++++++++++++++++++++++++++
/* ************************************************************************************************
```

```
 * FILE NAME   : vmm.c
 * PROGRAMMER  : zhaozz
 * DESCRIPTION : kernel vmm implement
 * DATE        : 2022-01-08 00:38:00
 * ********************************************************************************/


[src/kern/mm/pmm.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>
#include <kern/driver/console.h>
#include <kern/driver/intr.h>
#include <kern/debug/assert.h>
#include <kern/mm/default_pmm.h>
#include <kern/mm/kmalloc.h>
#include <kern/mm/memlayout.h>
#include <kern/mm/mmu.h>
#include <kern/mm/pmm.h>
#include <kern/sync/sync.h>
#include <kern/mm/vmm.h>
#include <kern/mm/swap.h>

/*
 * Task State Segment
 * the Task Register(TR) holds a segment selector that points to a
 * valid TSS segment descriptor which resides in the GDT.
 * Therefore, to use a TSS the following must be done in 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
 * 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 physical page array
struct page_frame* g_page_frame_base;
// amount of physical memory(in pages)
size_t g_npages = 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;

// physical memory management
const struct pmm_manager* g_pmm_manager;

/*
 * pde(page table 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.
 * */
pde_t* const vpd = (pde_t*)PGADDR(PDX(VPT), PDX(VPT), 0);
pte_t* const vpt = (pte_t*)VPT; // 1111 1010 1100 0000 0000 0000 0000 0000

/*
 * Gloable Descriptor Table:
 * The kernel and user segment are identical(except for the DPL).
 * To load the %ss register, the CPL must equal the DPL.
 *   - 0x0 : unused(always faults -- for trapping NULL for 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 segment register 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));
}

void load_esp0(uintptr_t esp0)
{
        ts.ts_esp0 = esp0;
}

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 field 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);
}

static void init_pmm_manager(void)
{
        g_pmm_manager = &default_pmm_manager;
        cprintf("memory management: %s\n", g_pmm_manager->name);
        g_pmm_manager->init();
}
```

```
// call pmm->init_memmap to build Page struct for free memory
static void init_memmap(struct page_frame* base, size_t n)
{
        g_pmm_manager->init_memmap(base, n);
}

// call pmm->alloc_pages to allocate a continuous n*PAGESIZE memory
struct page_frame* alloc_pages(size_t n)
{
        struct page_frame* page = NULL;
        bool intr_flag;

        for (;;) {
                local_intr_save(intr_flag);
                {
                        page = g_pmm_manager->alloc_pages(n);
                }
                local_intr_restore(intr_flag);
                // todo ...
                if (page != NULL || n > 1 || swap_init_ok == 0)
                        break;
                //extern struct mm_struct *check_mm_struct;
                //swap_out(check_mm_struct, n, 0);
        }
        return page;
}

void free_pages(struct page_frame* base, size_t n)
{
        bool intr_flag;
        local_intr_save(intr_flag);
        {
                g_pmm_manager->free_pages(base, n);
        }
        local_intr_restore(intr_flag);
}

size_t nr_free_pages(void)
{
        size_t ret;
        bool intr_flag;
        local_intr_save(intr_flag);
        {
                ret = g_pmm_manager->nr_free_pages();
        }
        local_intr_restore(intr_flag);
        return ret;
}

// initialize the physical memory management
static void page_init(void)
{
        struct e820map* memmap = (struct e820map*)(0x8000 + KERNBASE);
        u64 maxpa = 0;
        cprintf("e820map:\n");
        int i;
        for (i = 0; i < memmap->nr_map; i++) {
                u64 begin = memmap->map[i].addr, end = begin + memmap->map[i].size;
                if(i == 0 && i != memmap->nr_map - 1)
                        cprintf("â\224\234â\224\200â\224\200memory: size:%08llx, [%08llx, %08l
lx], type = %d - %s.\n",
                                        memmap->map[i].size, begin, end - 1, memmap->map[i].ty
pe, E820MAP_TYPE(memmap->map[i].type));
                else if (i == memmap->nr_map - 1)
```

```
                        cprintf("â\224\224â\224\200â\224\200memory: size:%08llx, [%08llx, %08l
lx], type = %d - %s.\n",
                                        memmap->map[i].size, begin, end - 1, memmap->map[i].ty
pe, E820MAP_TYPE(memmap->map[i].type));
                else
                        cprintf("â\224\234â\224\200â\224\200memory: size:%08llx, [%08llx, %08l
lx], type = %d - %s.\n",
                                        memmap->map[i].size, begin, end - 1, memmap->map[i].ty
pe, E820MAP_TYPE(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[];
        g_npages = maxpa / PGSIZE;
        g_page_frame_base = (struct page_frame*)ROUNDUP((void*)end, PGSIZE);
        for (i = 0; i < g_npages; i++) {
                SET_PAGE_RESERVED(g_page_frame_base + i);
        }

        uintptr_t freemem = PADDR((uintptr_t)g_page_frame_base + sizeof(struct page_frame) * g
_npages);

        for (i = 0; i < memmap->nr_map; i ++) {
                u64 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);
                                }
                        }
                }
        }

}

/*
 * setup & enable the paging machanism
 * 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, u32 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;
        }
}

/*
 * allocate one page using pmm->alloc_pages(1)
 * return: the kernel virtual address of this allocated page
 * note: this function is used to get the memory for PDT and PT
 * */
void* boot_alloc_page(void)
{
        struct page_frame* p = alloc_page();
        if (p == NULL) {
                panic("boot_alloc_page failed.\n");
        }
        return page2kva(p);
}

void pmm_init(void)
{
        // we've enabled paging
        boot_cr3 = PADDR(boot_pgdir);

        // alloc/free the physical memory(4KB)
        // a framework of physical memory manager (struct pmm_manager) is defined in pmm.h
        init_pmm_manager();

        // detect physical memory space, reserve already used memory,
        // use pmm->init_memmap to create free page list
        page_init();
        // use pmm->check to verify the correctness of the alloc/freee function in a pmm
        check_alloc_page();
        check_pgdir();

        static_assert(KERNBASE % PTSIZE == 0 && KERNTOP % PTSIZE == 0);
        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 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 memory map is established.
        // check the correctness of the basic virtual memory map.
        check_boot_pgdir();

        print_pg();
        kmalloc_init();
}

// get pte and return the kernel virtual address of this pte for la
// if the PT contains this pte didn't exist, alloc a page for PT
// pgdir: the kernel virtual base address of PDT
// la:    the linear address need to map
// create:if alloc a page for PT
// return: the kernel virtual address of this pte
pte_t* get_pte(pde_t* pgdir, uintptr_t la, bool create)
```

```
{
        pde_t* pdep = &pgdir[PDX(la)];
        if (!(*pdep & PTE_P)) {
                struct page_frame* 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;
        }
        pte_t* ret = &((pte_t*)KADDR(PDE_ADDR(*pdep)))[PTX(la)];
        return ret;
}

// get related Page struct for linear address la using PDT pgdir
struct page_frame* 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;
}

// free an Page struct 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)
{
        if (*ptep & PTE_P)
        {
                struct page_frame *page = pte2page(*ptep);
                if (page_ref_dec(page) == 0)
                {
                        free_page(page);
                }
                *ptep = 0;
                tlb_invalidate(pgdir, la);
        }
}

//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_frame *page, uintptr_t la, u32 perm) {
        udebug("pgdir=0x%x, la=0x%x\r\n", pgdir, la);
```

```
        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_frame *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);
        }
}

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 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.
 * 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));

        do{
                // call get_pte to find process A's pte according to the addr start
                pte_t *ptep = get_pte(from, start, 0);
                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, just alloc a PT.
                // todo...
        } while(1);
        return 0;
}

static void check_alloc_page(void)
{
        g_pmm_manager->check();
        cprintf("check_alloc_page succeed!\n");
}

static void check_pgdir(void)
{
        assert(g_npages <= KMEMSIZE / PGSIZE);
        assert(boot_pgdir != NULL && (u32)PGOFF(boot_pgdir) == 0);
        assert(get_page(boot_pgdir, 0x0, NULL) == NULL);

        struct page_frame *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);

        //TODO... crush here
        assert(page_ref(p1) == 2);
```

```
        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 succeed!\n");
}

static void check_boot_pgdir(void)
{
        pte_t* ptep;
        for (int i = 0; i < g_npages; 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_frame* 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 succeed!\n");
}

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;
}

static int get_pgtable_items(size_t left, size_t right, size_t start, uintptr_t* table, size_t
* left_store, 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;
}


void print_mem()
{
        struct e820map* memmap = (struct e820map*)(0x8000 + KERNBASE);
        cprintf("e820map:\n");
        int i;
        for (i = 0; i < memmap->nr_map; i++) {
                u64 begin = memmap->map[i].addr, end = begin + memmap->map[i].size;
                cprintf("â\224\234â\224\200â\224\200memory: size:%08llx(%8lldKB), [%08llx, %08
llx], type = %d - %s\n",
                                (u64)memmap->map[i].size,
                                (u64)(memmap->map[i].size/1024),
                                begin, end - 1, memmap->map[i].type, E820MAP_TYPE(memmap->map[
i].type));
        }
}

void print_pg(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(%8dKB) %s\n", right - left,
                                left * PTSIZE, right * PTSIZE,
                                (right - left) * PTSIZE, (right-left) *(PTSIZE/1024), 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(%8dKB) %s\n", r - l, l * PGSIZ
E, r * PGSIZE,
                                        (r - l) * PGSIZE, (r-l)*PGSIZE/1024, perm2str(perm));
                }
        }
        cprintf("--------END--------\n");
}
[src/kern/mm/default_pmm.c] ++++++++++++++++++++++++++++++
#include "memlayout.h"
#include <libs/libs_all.h>
#include <kern/mm/pmm.h>
#include <kern/mm/default_pmm.h>

free_area_t g_free_area;

#define g_free_list (g_free_area.free_list)
#define g_nr_free (g_free_area.nr_free)

static void default_init(void)
{
        list_init(&g_free_list);
        g_nr_free = 0;
```

```
}

static void default_init_memmap(struct page_frame *base, size_t n) {
        assert(n > 0);
        struct page_frame *p = base;
        for (; p != base + n; p ++) {
                assert(PAGE_RESERVED(p));
                p->flags = p->property = 0;
                set_page_ref(p, 0);
        }
        base->property = n;
        SET_PAGE_PROPERTY(base);
        g_nr_free += n;
        list_add_before(&g_free_list, &(base->page_link));
}

static struct page_frame * default_alloc_pages(size_t n)
{
        assert(n > 0);
        if (n > g_nr_free) {
                return NULL;
        }
        struct page_frame *page = NULL;
        list_entry_t *le = &g_free_list;
        // TODO: optimize (next-fit)
        while ((le = list_next(le)) != &g_free_list) {
                struct page_frame *p = le2page(le, page_link);
                if (p->property >= n) {
                        page = p;
                        break;
                }
        }
        if (page != NULL) {
                if (page->property > n) {
                        struct page_frame *p = page + n;
                        p->property = page->property - n;
                        SET_PAGE_PROPERTY(p);
                        list_add_after(&(page->page_link), &(p->page_link));
                }
                list_del(&(page->page_link));
                g_nr_free -= n;
                CLEAR_PAGE_PROPERTY(page);
        }
        return page;
}

static void default_free_pages(struct page_frame *base, size_t n)
{
        assert(n > 0);
        struct page_frame *p = base;
        for (; p != base + n; p ++) {
                assert(!PAGE_RESERVED(p) && !PAGE_PROPERTY(p));
                p->flags = 0;
                set_page_ref(p, 0);
        }
        base->property = n;
        SET_PAGE_PROPERTY(base);
        list_entry_t *le = list_next(&g_free_list);
        while (le != &g_free_list) {
                p = le2page(le, page_link);
                le = list_next(le);
                // TODO: optimize
                if (base + base->property == p) {
                        base->property += p->property;
                        CLEAR_PAGE_PROPERTY(p);
```

```
                        list_del(&(p->page_link));
                }
                else if (p + p->property == base) {
                        p->property += base->property;
                        CLEAR_PAGE_PROPERTY(base);
                        base = p;
                        list_del(&(p->page_link));
                }
        }
        g_nr_free += n;
        le = list_next(&g_free_list);
        while (le != &g_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 g_nr_free;
}

static void basic_check(void)
{
        struct page_frame *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) < g_npages * PGSIZE);
        assert(page2pa(p1) < g_npages * PGSIZE);
        assert(page2pa(p2) < g_npages * PGSIZE);

        list_entry_t free_list_store = g_free_list;
        list_init(&g_free_list);
        assert(list_empty(&g_free_list));

        unsigned int nr_free_store = g_nr_free;
        g_nr_free = 0;

        assert(alloc_page() == NULL);

        free_page(p0);
        free_page(p1);
        free_page(p2);
        assert(g_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(&g_free_list));
```

```
        struct page_frame *p;
        assert((p = alloc_page()) == p0);
        assert(alloc_page() == NULL);

        assert(g_nr_free == 0);
        g_free_list = free_list_store;
        g_nr_free = nr_free_store;

        free_page(p);
        free_page(p1);
        free_page(p2);
}

static void default_check(void)
{
        int count = 0, total = 0;
        list_entry_t *le = &g_free_list;
        while ((le = list_next(le)) != &g_free_list) {
                struct page_frame *p = le2page(le, page_link);
                assert(PAGE_PROPERTY(p));
                count ++, total += p->property;
        }
        assert(total == nr_free_pages());

        basic_check();

        struct page_frame *p0 = alloc_pages(5), *p1, *p2;
        assert(p0 != NULL);
        assert(!PAGE_PROPERTY(p0));

        list_entry_t free_list_store = g_free_list;
        list_init(&g_free_list);
        assert(list_empty(&g_free_list));
        assert(alloc_page() == NULL);

        unsigned int nr_free_store = g_nr_free;
        g_nr_free = 0;

        free_pages(p0 + 2, 3);
        assert(alloc_pages(4) == NULL);
        assert(PAGE_PROPERTY(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(PAGE_PROPERTY(p0) && p0->property == 1);
        assert(PAGE_PROPERTY(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(g_nr_free == 0);
        g_nr_free = nr_free_store;

        g_free_list = free_list_store;
```

```
        free_pages(p0, 5);

        le = &g_free_list;
        while ((le = list_next(le)) != &g_free_list) {
                struct page_frame *p = le2page(le, page_link);
                count --, total -= p->property;
        }
        assert(count == 0);
        assert(total == 0);
}

void print_free_pages()
{
        list_entry_t *le = &g_free_list;
        struct page_frame *p = le2page(le, page_link);
        do {
                p = le2page(le, page_link);
                uclean(
                                "------------------\n"
                                "page           :0x%x\n"
                                "ref            :0x%x\n"
                                "flags          :0x%x\n"
                                "property       :0x%x\n"
                                "zone_num       :0x%x\n"
                                "page_link      :0x%x\n"
                                "pra_page_link  :0x%x\n"
                                "pra_vaddr      :0x%x\n",
                                p               ,
                                p->ref          ,
                                p->flags        ,
                                p->property     ,
                                p->zone_num     ,
                                p->page_link    ,
                                p->pra_page_link,
                                p->pra_vaddr    );
        } while ((le = list_next(le)) != &g_free_list);
}

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,
};
[src/kern/mm/vmm.h] ++++++++++++++++++++++++++++++
/* ***************************************************************************************
 * FILE NAME   : vmm.h
 * PROGRAMMER  : zhaozz
 * DESCRIPTION : kernel vmm
 * DATE        : 2022-01-08 00:37:46
 * ***************************************************************************************/
#ifndef  __VMM_H__
#define  __VMM_H__


#include <libs/defs.h>
#include <libs/list.h>
#include <kern/mm/memlayout.h>
#include <kern/sync/sync.h>
#include <kern/process/proc.h>
#include <kern/sync/sem.h>
```

```
struct mm_struct;

struct vma_struct {
        struct mm_struct *vm_mm;
        uintptr_t vm_start;
        uintptr_t vm_end;
        uintptr_t vm_flags;
        list_entry_t list_link;
};

#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

struct mm_struct {
        list_entry_t mmap_list;
        struct vma_struct *mmap_cache;
        pde_t *pgdir;
        int map_count;
        void *sm_priv;
        int mm_count;
//      semaphore_t mm_sem;
        int locked_by;
};


#endif  /* __VMM_H__ */
[src/kern/mm/swap_fifo.c] ++++++++++++++++++++++++++++++
[src/kern/mm/memlayout.h] ++++++++++++++++++++++++++++++
#ifndef __KERN_MM_MEMLAYOUT_H__
#define __KERN_MM_MEMLAYOUT_H__

/* global segment number */
#define SEG_KTEXT 1
#define SEG_KDATA 2
#define SEG_UTEXT 3
#define SEG_UDATA 4
#define SEG_TSS   5

/* global descriptor number */
#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 seletoc

#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)


#define KERNBASE  0xC0000000
#define KMEMSIZE  0x38000000   // the maximum amount of physical memory : 896MB
#define KERNTOP   (KERNBASE + KMEMSIZE)

#define VPT 0xFAC00000  // virtual page table. entry PDX[VPT] in the PD (page directory), PD m
aps all the PTEs for the entir virtual address space (4MB region starting at VPT)
```

```
#define KSTACKPAGE 2
#define KSTACKSIZE  (KSTACKPAGE * PGSIZE)

#define USERTOP  0xB0000000
#define USTACKTOP USERTOP
#define USTACKPAGE 256
#define USTACKSIZE (USTACKPAGE * PGSIZE)

#define USERBASE 0x00200000
#define UTEXT 0x00800000
#define USTAB USERBASE

#define USER_ACCESS(start , end) \
        (USERBASE <= (start) && (start) < (end) && (end) <= USERTOP)

#define KERN_ACCESS(start, end) \
        (KERNBASE <= (start) && (start) < (end) && (end) <= KERNTOP)

/* *
 * 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.
 * */

#ifndef __ASSEMBLER__
#include <libs/defs.h>
#include <libs/atomic.h>
#include <libs/list.h>

typedef uintptr_t pde_t;
typedef uintptr_t pte_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 {
                u64 addr;
                u64 size;
                u32 type;  // 1:memory, 2:reserved(ROM, memory-mapped device), 3:ACPI Reclaim
memory, 4:ACPI NVS memory
        } __attribute__((packed)) map[E820MAX];
};

#define E820MAP_TYPE(type)  ({ \
                char *p_ret = ""; \
                if (type == 1) \
                p_ret = "memory"; \
                else if (type == 2) \
                p_ret = "reserved(ROM, memory-mapped device)"; \
                else if (type == 3) \
                p_ret = "ACPI Reclaim memory"; \
                else if (type == 4) \
                p_ret = "ACPI NVS memory"; \
                else \
                p_ret = "not defined!"; \
                p_ret; \
                })

/*
 * sturct page - page descriptor structures(physical page).
 * */
struct page_frame {
        int             ref;                // page frame's reference counter
        u32             flags;              // array of flags that describe the status of the
page frame
        unsigned int    property;           // used in buddy system, stores the order (the X i
n 2^X) of the continuous memory block
        int             zone_num;           // used in buddy system, the No. of zone which the
 page belongs 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
#define PG_property 1
#define SET_PAGE_RESERVED(page) set_bit(PG_reserved, &((page)->flags))
#define CLEAR_PAGE_RESERVED(page) clear_bit(PG_reserved, &((page)->flags))
#define PAGE_RESERVED(page) test_bit(PG_reserved, &((page)->flags))
#define SET_PAGE_PROPERTY(page) set_bit(PG_property, &((page)->flags))
#define CLEAR_PAGE_PROPERTY(page) clear_bit(PG_property, &((page)->flags))
#define PAGE_PROPERTY(page) test_bit(PG_property, &((page)->flags))

#define le2page(le, member) \
        to_struct((le), struct page_frame, member)

typedef struct{
        list_entry_t free_list;
        unsigned int nr_free;
} free_area_t;
```

```
#endif  /* !__ASSEMBLER__ */
#endif /* !__KERN_MM_MEMLAYOUT_H__ */
[src/kern/mm/pmm.h] ++++++++++++++++++++++++++++++
#ifndef  __PMM_H__
#define  __PMM_H__

#include <libs/defs.h>
#include <kern/mm/mmu.h>
#include <kern/mm/memlayout.h>
#include <libs/atomic.h>
#include <kern/debug/assert.h>

struct pmm_manager {
        const char* name;
        void (*init)(void);
        void (*init_memmap)(struct page_frame* base, size_t n);
        struct page_frame* (*alloc_pages)(size_t n);
        void (*free_pages)(struct page_frame* base, size_t n);
        size_t (*nr_free_pages)(void);
        void (*check)(void);
};

extern const struct pmm_manager* pmm_manager;
extern pde_t* boot_pgdir;
extern uintptr_t boot_cr3;

extern struct page_frame* g_page_frame_base;
extern size_t g_npages;

extern char bootstack[], bootstacktop[];

void pmm_init(void);
struct page_frame* alloc_pages(size_t n);
void free_pages(struct page_frame* 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_frame* 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_frame* page, uintptr_t la, u32 perm);

void load_esp0(uintptr_t esp0);
void tlb_invalidate(pde_t* pgdir, uintptr_t la);
struct page_frame* pgdir_alloc_page(pde_t* pgdir, uintptr_t la, u32 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_pg(void);
void print_mem();

/*
 * takes a kernel virtual address(above KERNBASE)
 * returns the corresponding physical 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;                                     \
                })

/*
 * takes a physical address
 * returns the corresponding kernel virtual address
 * */
#define KADDR(pa) ({                                            \
                uintptr_t __m_pa = (pa);                                \
                size_t __m_ppn = PPN(__m_pa);                           \
                if (__m_ppn >= g_npages) {                                  \
                panic("KADDR called with invalid pa %08lx", __m_pa); \
                }                                                       \
                (void*)(__m_pa + KERNBASE);                             \
                })

static inline ppn_t page2ppn(struct page_frame* page)
{
        return page - g_page_frame_base;
}

static inline uintptr_t page2pa(struct page_frame* page)
{
        return page2ppn(page) << PGSHIFT;
}

static inline struct page_frame* pa2page(uintptr_t pa)
{
        if (PPN(pa) >= g_npages) {
                panic("pa2page called with invalid pa");
        }
        return &g_page_frame_base[PPN(pa)];
}

static inline void* page2kva(struct page_frame* page)
{
        return KADDR(page2pa(page));
}

static inline struct page_frame* pte2page(pte_t pte)
{
        if (!(pte & PTE_P)) {
                panic("pte2page call with invalid pte");
        }
        return pa2page(PTE_ADDR(pte));
}

static inline struct page_frame* pde2page(pde_t pde)
{
        return pa2page(PDE_ADDR(pde));
}

static inline int page_ref(struct page_frame* page)
{
        return page->ref;
}

static inline void set_page_ref(struct page_frame* page, int val)
{
        page->ref = val;
}

static inline int page_ref_inc(struct page_frame* page)
{
```

```
        page->ref += 1;
        return page->ref;
}

static inline int page_ref_dec(struct page_frame* page)
{
        page->ref -= 1;
        return page->ref;
}

static inline struct page_frame * kva2page(void *kva) {
        return pa2page(PADDR(kva));
}


#endif  /* __PMM_H__ */
[src/kern/mm/kmalloc.c] ++++++++++++++++++++++++++++++
#include <libs/libs_all.h>
#include <kern/mm/memlayout.h>
#include <kern/debug/assert.h>
#include <kern/mm/kmalloc.h>
#include <kern/sync/sync.h>
#include <kern/mm/pmm.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;
};

#define SLOB_UNIT sizeof(struct slob_block)
#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;
};

static struct slob_block arena = { .next = &arena, .units = 1 };
static struct slob_block *slobfree = &arena;
static struct bigblock *bigblocks;


static void* __slob_get_free_pages(gfp_t gfp, int order)
{
        struct page_frame * 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((void *)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 );

        struct slob_block *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 = (struct slob_block *)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 = (struct slob_block *)__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)
{
        struct slob_block *cur, *b = (struct slob_block *)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)
{
        struct slob_block *m;
        struct bigblock *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(struct bigblock), 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(struct bigblock));
        return 0;
}
```

```
void * kmalloc(size_t size)
{
        return __kmalloc(size, 0);
}


void kfree(void *block)
{
        struct bigblock *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(struct bigblock));
                                return;
                        }
                }
                spin_unlock_irqrestore(&block_lock, flags);
        }

        slob_free((struct slob_block *)block - 1, 0);
        return;
}


unsigned int ksize(const void *block)
{
        struct bigblock *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 ((struct slob_block *)block - 1)->units * SLOB_UNIT;
}

[src/kern/mm/swap_fifo.h] ++++++++++++++++++++++++++++++
[src/kern/mm/mmu.h] ++++++++++++++++++++++++++++
#ifndef  __MMU_H__
#define  __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 <libs/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 = (u32)(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 = (u32)(off) >> 16;      \
}

/* Set up a call gate descriptor */
#define SETCALLGATE(gate, ss, off, dpl)              \
{                                                \
        (gate).gd_off_15_0 = (u32)(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 = (u32)(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 {
        u32 ts_link;  // old ts selector
        uintptr_t ts_esp0; // stack pointers and segment selectors
        u16 ts_ss0;   // after an increase in privilege level
        u16 ts_padding1;
        uintptr_t ts_esp1;
        u16 ts_ss1;
        u16 ts_padding2;
        uintptr_t ts_esp2;
        u16 ts_ss2;
        u16 ts_padding3;
        uintptr_t ts_cr3; // page directory base
        uintptr_t ts_eip; // saved state from last task switch
        u32 ts_eflags;
        u32 ts_eax; // more saved state (registers)
        u32 ts_ecx;
        u32 ts_edx;
        u32 ts_ebx;
        uintptr_t ts_esp;
        uintptr_t ts_ebp;
        u32 ts_esi;
        u32 ts_edi;
        u16 ts_es; // even more saved state (segment selectors)
        u16 ts_padding4;
        u16 ts_cs;
        u16 ts_padding5;
        u16 ts_ss;
        u16 ts_padding6;
        u16 ts_ds;
        u16 ts_padding7;
        u16 ts_fs;
        u16 ts_padding8;
        u16 ts_gs;
        u16 ts_padding9;
        u16 ts_ldt;
        u16 ts_padding10;
        u16 ts_t;     // trap on task switch
        u16 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 kernel 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  /* __MMU_H__ */
[src/kern/schedule/default_sched.c] ++++++++++++++++++++++++++++++
[src/kern/schedule/sched.c] ++++++++++++++++++++++++++++++
[src/kern/schedule/sched.h] ++++++++++++++++++++++++++++++
[src/kern/schedule/default_sched.h] ++++++++++++++++++++++++++++++
[src/boot/bootasm.S] ++++++++++++++++++++++++++++++
#include <asm.h>

.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

.global 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
    movw %ax, %ds

#enable A20:
seta20.1:
    inb $0x64, %al
    testb $0x2, %al
    jnz seta20.1

    movb $0xd1, %al
    outb %al, $0x64

seta20.2:
    inb $0x64, %al
    testb $0x2, %al
    jnz seta20.2

    mov $0xdf, %al
    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
protcseg:
    # set up the protedted-mode data segment registers
    movw $PROT_MODE_DSEG, %ax
    movw %ax,%ds
    movw %ax,%es
    movw %ax,%fs
    movw %ax,%gs
    movw %ax,%ss

    # 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

spin:
    jmp spin


# bootstrap GDT
# p2align == power-of-two bytes alignment (2*2=4-bytes boundary)
.p2align 2
gdt:
    SEG_NULLASM
    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

# pdf 330: GDTR.Limit:Base <- m16:32
gdtdesc:
    .word 0x17        # sizeof(gdt) - 1
    .long gdt        #address gdt
[src/boot/asm.h] ++++++++++++++++++++++++++++++
#ifndef __TOYOS_ASM_H__
#define __TOYOS_ASM_H__

/*pdf = "INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986"
 *pdf 108: Figure 6-1 */

#define SEG_NULLASM \
    .word 0, 0;     \
    .byte 0, 0, 0, 0

#define SEG_ASM(type, base, limit)                      \
    .word (((limit) >> 12) & 0xffff), (base)&0xffff;  \
    .byte(((base) >> 16) & 0xff), (0x90 | (type)),    \
        (0xC0 | (((limit) >> 28) & 0xf)), (((base) >> 24) & 0xff)

#define STA_X 0x8
#define STA_E 0x4
#define STA_C 0x4
#define STA_W 0x2
#define STA_R 0x2
```

```
#define STA_A 0x1

#endif
[src/boot/bootmain.c] ++++++++++++++++++++++++++++++
#include <libs/defs.h>
#include <libs/x86.h>
#include <libs/elf.h>

#define SECTSIZE 512
#define ELFHDR ((struct elfhdr*)0x10000) // scratch space

static void waitdisk(void)
{
        while ((inb(0x1F7) & 0xC0) != 0x40);
}

/* read_sect -  read a single sector at @secnum into @dst*/
static void read_sect(void* dst, u32 secnum)
{
        waitdisk();
        outb(0x1F2, 1);
        outb(0x1F3, secnum & 0xFF);
        outb(0x1F4, (secnum >> 8) & 0xFF);
        outb(0x1F5, (secnum >> 16) & 0xFF);
        outb(0x1F6, ((secnum >> 24) & 0xF) | 0xE0);
        outb(0x1F7, 0x20);  // cmd 0x20 - read sectors

        waitdisk();
        insl(0x1F0, dst, SECTSIZE / 4);
        waitdisk();
}

/*
 * read_seg - read @count bytes at @offset from kernel into virtual address @va,
 * might copy more than asked
 * */
static void read_seg(uintptr_t va, u32 count, u32 offset)
{
        uintptr_t end_va = va + count;
        va -= offset % SECTSIZE;
        u32 secnum = (offset / SECTSIZE) + 1;
        for (; va < end_va; va += SECTSIZE, secnum++) {
                read_sect((void*)va, secnum);
        }
}

void bootmain(void)
{
        read_seg((uintptr_t)ELFHDR, SECTSIZE * 8, 0);
        if (ELFHDR->e_magic != ELF_MAGIC) {
                goto bad;
        }
        struct proghdr *ph, *eph;
        ph = (struct proghdr*)((uintptr_t)ELFHDR + ELFHDR->e_phoff);
        eph = ph + ELFHDR->e_phnum;
        for (; ph < eph; ph++) {
                read_seg(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);
```

```
        while (1);
}
```