گزارش فاز اول پروژه سیستم‌های عامل

اشکان شکیبا (۹۹۳۱۰۳۰)

سوال اول

ابتدا تابع kinit حافظه را آزاد می‌کند. سپس با اجرای تابع kvminit یک پیج تیبل ساخته می‌شود تا کرنل بتواند از فضای آدرس‌های مختلف استفاده کند. در ادامه تابع kvminithart برای هر هسته پردازنده اجرا می‌شود و پیج تیبل را راه‌اندازی و در نهایت فلاش می‌کند. پس از آن تابع procinit فراخوانی شده و جدول پردازه‌ها را راه‌اندازی می‌کند. سپس تابع trapinit یک لاک برای ساخت trapها می‌سازد. در ادامه تابع trapinithart اجرا می‌شود که وظیفه مدیریت فراخوانی‌های سیستمی و وقفه‌ها را دارد. پس از آن تابع plicinit کنترل‌گر وقفه‌ها را راه‌اندازی می‌کند. سپس تابع plicinithart اجرا می‌شود که مربوط به وقفه‌های دستگاه‌هاست. در ادامه تابع binit اجرا شده و یک بافر کش مرتب‌شده می‌سازد. پس از آن تابع iinit با لاک کردن inodeها دسترسی به آنها را مدیریت می‌کند. سپس تابع fileinit با لاک کردن ftable فایل‌ها را نگهداری می‌کند. در ادامه با اجرا شدن تابع virtio_disk_init درایورها ساخته می‌شوند.

پس از همه این مراحل، نوبت به اجرای تابع userinit می‌رسد. با اجرای این تابع ابتدا متغیر p که اشاره‌گری از نوع پردازه است ساخته شده و آدرس پردازه مورد استفاده در آن ذخیره می‌شود. سپس با فراخوانی تابع allocproc فضای مورد نیاز از حافظه allocate شده و متغیر initproc برابر با اولین پردازه مقداردهی می‌شود. پس از آن تابع uvmfirst اجرا شده و یک پیج از حافظه

به کاربر اختصاص می‌یابد (در واقع یک پیج از حافظه به پردازه داده می‌شود). در ادامه مقادیر اشاره‌گرهای برنامه و استک برابر صفر قرار داده می‌شوند. سپس پردازه نام‌گذاری شده و محل کار آن برابر با root سیستم قرار داده می‌شود. پس از آن نیز وضعیت پردازه به در حال اجرا تغییر داده می‌شود تا وارد روند اجرا شود. در انتها لاک مربوطه release می‌شود و فرایند پایان می‌یابد.

سوال دوم

تابع syscall که در فایل syscall.c قرار دارد، وظیفه دریافت و اجرای فراخوانی‌های سیستمی را دارد که به شکل entry به usys.pl افزوده شده‌اند. در هر بار اجرای این تابع، اطلاعات فراخوانی شامل نام و شماره آن (که پیش‌تر در فایل syscall.h ثبت شده) در رجیستر a7 قرار می‌گیرد. سپس با توجه به این اطلاعات، تابع مربوط به فراخوانی که پیش‌تر در فایل sysproc.c ذخیره شده اجرا می‌شود که خود آن نیز با جست‌وجو و اجرای تابعی که در فایل defs.h معرفی کرده‌ایم، تابع اصلی را از فایل proc.c فراخوانی کرده و خروجی آن را به عنوان خروجی فراخوانی سیستمی بازگردانی می‌کند.

اولین فراخوانی سیستمی (getProcTick)

مرحله اول: افزودن entry به فایل usys.pl

```
entry("fork");
entry("exit");
entry("wait");
entry("pipe");
entry("read");
entry("write");
entry("close");
entry("kill");
entry("exec");
entry("open");
entry("mknod");
entry("unlink");
entry("fstat");
entry("link");
entry("mkdir");
entry("chdir");
entry("dup");
entry("getpid");
entry("sbrk");
entry("sleep");
entry("uptime");
entry("getProcTick");
```

مرحله دوم: افزودن پروتوتایپ تابع مربوطه به فایل user.h

```
// system calls
int fork(void);
int exit(int) __attribute__((noreturn));
int wait(int*);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(const char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
```

```
char* sbrk(int);
int sleep(int);
int uptime(void);
int getProcTick(int);
```

مرحله سوم: افزودن نام و شماره فراخوانی به فایل syscall.h

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_getProcTick  22
```

مرحله چهارم: افزودن پروتوتایپ و نام فراخوانی به فایل syscall.c

```
// Prototypes for the functions that handle system calls.
extern uint64 sys_fork(void);
extern uint64 sys_exit(void);
extern uint64 sys_wait(void);
extern uint64 sys_pipe(void);
extern uint64 sys_read(void);
extern uint64 sys_kill(void);
extern uint64 sys_exec(void);
extern uint64 sys_fstat(void);
extern uint64 sys_chdir(void);
extern uint64 sys_dup(void);
extern uint64 sys_getpid(void);
extern uint64 sys_sbrk(void);
extern uint64 sys_sleep(void);
extern uint64 sys_uptime(void);
extern uint64 sys_open(void);
```

```
extern uint64 sys_write(void);
extern uint64 sys_mknod(void);
extern uint64 sys_unlink(void);
extern uint64 sys_link(void);
extern uint64 sys_mkdir(void);
extern uint64 sys_close(void);
extern uint64 sys_getProcTick(void);

// An array mapping syscall numbers from syscall.h
// to the function that handles the system call.
static uint64 (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_getProcTick]   sys_getProcTick,
};
```

مرحله پنجم: پیاده‌سازی تابع فراخوانی در فایل sysproc.c

```
uint64
sys_getProcTick(void)
{
    int pid;
    argint(0, &pid);
    return getProcTick(pid);
}
```

مرحله ششم: افزودن فیلد ticks به استراکت proc در فایل proc.h

```
// Per-process state
struct proc {
```

```
  struct spinlock lock;

  // p->lock must be held when using these:
  enum procstate state;        // Process state
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  int xstate;                  // Exit status to be returned to parent's
wait
  int pid;                     // Process ID

  // wait_lock must be held when using this:
  struct proc *parent;         // Parent process

  // these are private to the process, so p->lock need not be held.
  uint64 kstack;               // Virtual address of kernel stack
  uint64 sz;                   // Size of process memory (bytes)
  pagetable_t pagetable;       // User page table
  struct trapframe *trapframe; // data page for trampoline.S
  struct context context;      // swtch() here to run process
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)

  uint ticks;
};
```

مرحله هفتم: مقداردهی فیلد ticks در انتهای تابع allocproc در فایل proc.c

```
// Set up new context to start executing at forkret,
// which returns to user space.
memset(&p->context, 0, sizeof(p->context));
p->context.ra = (uint64)forkret;
p->context.sp = p->kstack + PGSIZE;
p->ticks = ticks;
```

مرحله هشتم: پیاده‌سازی تابع اصلی در فایل proc.c

```
int
getProcTick(int pid) {
    struct proc *p;
    for (p = proc; p < &proc[NPROC]; p++) {
        acquire(&p->lock);
        if (pid == p->pid) {
            int result = ticks - (p->ticks);
            release(&p->lock);
            return result;
        }
        release(&p->lock);
    }
```

```
    return -1;
}
```

مرحله نهم: افزودن پروتوتایپ تابع اصلی به فایل defs.h

```
// proc.c
int             cpuid(void);
void            exit(int);
int             fork(void);
int             growproc(int);
void            proc_mapstacks(pagetable_t);
pagetable_t     proc_pagetable(struct proc *);
void            proc_freepagetable(pagetable_t, uint64);
int             kill(int);
int             killed(struct proc*);
void            setkilled(struct proc*);
struct cpu*     mycpu(void);
struct cpu*     getmycpu(void);
struct proc*    myproc();
void            procinit(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(uint64);
void            wakeup(void*);
void            yield(void);
int             either_copyout(int user_dst, uint64 dst, void *src, uint64
len);
int             either_copyin(void *dst, int user_src, uint64 src, uint64
len);
void            procdump(void);
int             getProcTick(int);
```

مرحله دهم: ساخت فایل تست با نام getProcTickTest.c در دایرکتوری user و پیاده‌سازی مراحل تست در تابع main آن

```
#include "../kernel/types.h"
#include "../kernel/stat.h"
#include "user.h"

int main(int argc, char *argv[]) {
    int pid = atoi(argv[1]);
    fprintf(2, "%d\n", getProcTick(pid));
    exit(0);
}
```

مرحله یازدهم: افزودن فایل تست به Makefile

```
UPROGS=\
    $U/_cat\
    $U/_echo\
    $U/_forktest\
    $U/_grep\
    $U/_init\
    $U/_kill\
    $U/_ln\
    $U/_ls\
    $U/_mkdir\
    $U/_rm\
    $U/_sh\
    $U/_stressfs\
    $U/_usertests\
    $U/_grind\
    $U/_wc\
    $U/_zombie\
    $U/_getProcTickTest\
```

دومین فراخوانی سیستمی (sysinfo)

مرحله اول: افزودن entry به فایل usys.pl

```
entry("fork");
entry("exit");
entry("wait");
entry("pipe");
entry("read");
entry("write");
entry("close");
entry("kill");
entry("exec");
entry("open");
entry("mknod");
entry("unlink");
entry("fstat");
entry("link");
entry("mkdir");
entry("chdir");
entry("dup");
entry("getpid");
entry("sbrk");
entry("sleep");
entry("uptime");
entry("getProcTick");
entry("sysinfo");
```

مرحله دوم: افزودن پروتوتایپ تابع مربوطه به فایل user.h

```
// system calls
int fork(void);
int exit(int) __attribute__((noreturn));
int wait(int*);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(const char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
```

```
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int getProcTick(int);
int sysinfo(int);
```

مرحله سوم: افزودن نام و شماره فراخوانی به فایل syscall.h

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_getProcTick  22
#define SYS_sysinfo  23
```

مرحله چهارم: افزودن پروتوتایپ و نام فراخوانی به فایل syscall.c

```
// Prototypes for the functions that handle system calls.
extern uint64 sys_fork(void);
extern uint64 sys_exit(void);
extern uint64 sys_wait(void);
extern uint64 sys_pipe(void);
extern uint64 sys_read(void);
extern uint64 sys_kill(void);
extern uint64 sys_exec(void);
extern uint64 sys_fstat(void);
extern uint64 sys_chdir(void);
extern uint64 sys_dup(void);
extern uint64 sys_getpid(void);
extern uint64 sys_sbrk(void);
```

```
extern uint64 sys_sleep(void);
extern uint64 sys_uptime(void);
extern uint64 sys_open(void);
extern uint64 sys_write(void);
extern uint64 sys_mknod(void);
extern uint64 sys_unlink(void);
extern uint64 sys_link(void);
extern uint64 sys_mkdir(void);
extern uint64 sys_close(void);
extern uint64 sys_getProcTick(void);
extern uint64 sys_sysinfo(void);

// An array mapping syscall numbers from syscall.h
// to the function that handles the system call.
static uint64 (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_getProcTick]   sys_getProcTick,
[SYS_sysinfo]   sys_sysinfo,
};
```

مرحله پنجم: پیاده‌سازی تابع فراخوانی در فایل sysproc.c

```
uint64
sys_sysinfo(void) {
    uint64 info;
    argaddr(0, &info);
    return sysinfo(info);
}
```

مرحله ششم: ساخت فایل sysinfo.h در دایرکتوری kernel و تعریف استراکت sysinfo در آن

```
struct sysinfo {
    long uptime; // Seconds since boot
    unsigned long totalram; // Total usable main memory size
    unsigned long freeram; // Available memory size
    unsigned short procs; // Number of current processes
};
```

مرحله هفتم: افزودن sysinfo.h به ابتدای فایل proc.c

```
#include "types.h"
#include "param.h"
#include "memlayout.h"
#include "riscv.h"
#include "spinlock.h"
#include "proc.h"
#include "defs.h"
#include "sysinfo.h"
```

مرحله هشتم: پیاده‌سازی توابع freeram و totalram در فایل kalloc.c

```
long
freeram(void) {
    struct run *r;
    int free_ram = 0;

    acquire(&kmem.lock);
    r = kmem.freelist;
    while(r) {
        free_ram++;
        r = r->next;
    }
    release(&kmem.lock);

    return free_ram * 4096;
}

long
totalram(void) {
    long total_ram;

    acquire(&kmem.lock);
    total_ram = PHYSTOP - KERNBASE;
    release(&kmem.lock);
```

```
    return total_ram;
}
```

مرحله نهم: افزودن توابع freeram و totalram به فایل defs.h

```
// kalloc.c
void*           kalloc(void);
void            kfree(void *);
void            kinit(void);
long            freeram(void);
long            totalram(void);
```

مرحله دهم: پیاده‌سازی تابع processes در فایل proc.c

```
int
processes(void) {
    struct proc *p;
    int result = 0;
    for (p = proc; p < &proc[NPROC]; p++) {
        acquire(&p->lock);
        if (p->state != UNUSED)
            result++;
        release(&p->lock);
    }
    return result;
}
```

مرحله یازدهم: پیاده‌سازی تابع اصلی در فایل proc.c

```
int
sysinfo(uint64 addr) {
    struct sysinfo si;
    struct proc *p = myproc();

    si.uptime = (long) ticks / 100;
    si.freeram = freeram();
    si.totalram = totalram();
    si.procs = processes();

    if (copyout(p->pagetable, addr, (char *) &si, sizeof(si)) < 0)
        return -1;
    else
        return 0;
}
```

مرحله دوازدهم: افزودن پروتوتایپ تابع اصلی به فایل defs.h

```
// proc.c
int             cpuid(void);
void            exit(int);
int             fork(void);
int             growproc(int);
void            proc_mapstacks(pagetable_t);
pagetable_t     proc_pagetable(struct proc *);
void            proc_freepagetable(pagetable_t, uint64);
int             kill(int);
int             killed(struct proc*);
void            setkilled(struct proc*);
struct cpu*     mycpu(void);
struct cpu*     getmycpu(void);
struct proc*    myproc();
void            procinit(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(uint64);
void            wakeup(void*);
void            yield(void);
int             either_copyout(int user_dst, uint64 dst, void *src, uint64
len);
int             either_copyin(void *dst, int user_src, uint64 src, uint64
len);
void            procdump(void);
int             getProcTick(int);
int             sysinfo(uint64);
```

مرحله سیزدهم: ساخت فایل تست با نام sysinfoTest.c در دایرکتوری user و پیاده‌سازی مراحل تست در تابع main آن

```
#include "../kernel/types.h"
#include "../kernel/stat.h"
#include "../kernel/sysinfo.h"
#include "user.h"

int main(int argc, char **argv) {
    struct sysinfo si;
    sysinfo((uint64) &si);
    printf("uptime: %l\n"
           "total memory: %l\n"
           "free memory: %l\n"
           "active processes: %d\n",
           si.uptime,
           si.totalram,
           si.freeram,
```

```
            si.procs);
    exit(0);
}
```

مرحله چهاردهم: افزودن فایل تست به Makefile

```
UPROGS=\
   $U/_cat\
   $U/_echo\
   $U/_forktest\
   $U/_grep\
   $U/_init\
   $U/_kill\
   $U/_ln\
   $U/_ls\
   $U/_mkdir\
   $U/_rm\
   $U/_sh\
   $U/_stressfs\
   $U/_usertests\
   $U/_grind\
   $U/_wc\
   $U/_zombie\
   $U/_getProcTickTest\
   $U/_sysinfoTest\
```