**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 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*\*