**گزارش فاز دوم پروژه سیستم‌های عامل**

**اشکان شکیبا (۹۹۳۱۰۳۰)، علی هاشم‌پور (۹۹۳۱۰۸۲)**

**سوال اول**

الگوریتم زمان‌بندی پیش‌فرض در xv6، شکلی از الگوریتم round robin است.

**سوال دوم**

ابتدا تابع scheduler در فایل proc.c اجرا شده و یک پردازه می‌سازد و آن را به پردازنده می‌دهد. سپس تابع devintr با بررسی زمان اجرای هر پردازه، پس از اتمام زمان اجرای آن یک timer interrupt می‌دهد. پس از آن در تابع usertrap فایل trap.c، شرط برابری which\_dev با ۲ برقرار شده و تابع yield فراخوانی می‌شود که خود تابع sched را فراخوانی می‌کند. این تابع وظیفه انجام context switch را دارد، به گونه‌ای که پردازه در حال اجرا در پردازنده را تغییر داده و در صورتی که اجرای پردازه به پایان نرسیده باشد، آن را به ready queue می‌افزاید.

**پیاده‌سازی**

تعریف enum مربوط به حالت زمان‌بند در فایل proc.c

enum SchedulerType {  
 *DEFAULT*,  
 *FCFS*};  
  
enum SchedulerType scheduler\_type = *DEFAULT*;

تکمیل استراکت proc و تعریف استراکت proc\_times در فایل proc.h

uint ticks;  
 long termination\_time;  
 long running\_time;  
 long ready\_time;  
 long sleeping\_time;  
};  
  
struct proc\_times {  
 long burst\_time;  
 long turnaround\_time;  
 long waiting\_time;  
};

مقداردهی اولیه فیلدهای time در تابع allocproc فایل proc.c

p->ticks = ticks;  
p->termination\_time=0;  
p->running\_time=0;  
p->ready\_time=0;  
p->sleeping\_time=0;

مقداردهی به termination\_time پردازه در تابع exit فایل proc.c

acquire(&p->lock);  
p->termination\_time = ticks;  
  
p->xstate = status;  
p->state = *ZOMBIE*;

تعریف توابع round\_robin\_scheduler و fcfs\_scheduler و تغییر تابع scheduler در فایل proc.c

void  
round\_robin\_scheduler(struct cpu \*c) {  
 struct proc \*p;  
 for (p = proc; p < &proc[NPROC]; p++) {  
 acquire(&p->lock);  
 if (p->state == *RUNNABLE*) {  
 p->state = *RUNNING*;  
 c->proc = p;  
 swtch(&c->context, &p->context);  
 c->proc = 0;  
 }  
 release(&p->lock);  
 }  
}  
  
void fcfs\_running(struct cpu \*c) {  
 struct proc \*p;  
 struct proc \*first\_process = 0;  
 for (p = proc; p < &proc[NPROC]; p++) {  
 if (p->state != *RUNNABLE*) {  
 continue;  
 }  
 if ((p->pid == 1 || p->pid == 2)) {  
 continue;  
 }  
 if (first\_process == 0 || p->ticks < first\_process->ticks) {  
 first\_process = p;  
 }  
 }  
 if (first\_process != 0) {  
 acquire(&first\_process->lock);  
 if (first\_process->state == *RUNNABLE*) {  
 first\_process->state = *RUNNING*;  
 c->proc = first\_process;  
 swtch(&c->context, &first\_process->context);  
 c->proc = 0;  
 }  
 release(&first\_process->lock);  
 } else {  
 if (first\_process == 0) {  
 round\_robin\_scheduler(c);  
 }  
 }  
}  
  
void  
scheduler(void) {  
 struct cpu \*c = mycpu();  
  
 c->proc = 0;  
 for (;;) {  
 intr\_on();  
 if (scheduler\_type == *DEFAULT*) {  
 round\_robin\_scheduler(c);  
 } else {  
 fcfs\_running(c);  
 }  
 }  
}

تعریف توابع update\_process\_times، childWait و toggleScheduler در فایل proc.c

void  
update\_process\_times() {  
 struct proc \*p;  
 for (p = proc; p < &proc[NPROC]; p++) {  
 acquire(&p->lock);  
 if (p->state == *RUNNING*) {  
 p->running\_time++;  
 }  
 if (p->state == *RUNNABLE*) {  
 p->ready\_time++;  
 }  
 if (p->state == *SLEEPING*) {  
 p->sleeping\_time++;  
 }  
 release(&p->lock);  
 }  
}  
  
int  
childWait(uint64 addr, uint64 proc\_time) {  
 struct proc \*child;  
 int child\_pid, have\_children;  
 struct proc \*p = myproc();  
 acquire(&wait\_lock);  
 for (;;) {  
 have\_children = 0;  
 for (child = proc; child < &proc[NPROC]; child++) {  
 if (child->parent == p) {  
 acquire(&child->lock);  
 have\_children = 1;  
 if (child->state == *ZOMBIE*) {  
 child\_pid = child->pid;  
 struct proc\_times child\_times;  
 child\_times.burst\_time = child->running\_time;  
 child\_times.turnaround\_time = child->termination\_time - child->ticks;  
 child\_times.waiting\_time = child->sleeping\_time + child->ready\_time;  
 if (addr != 0 && copyout(p->pagetable, addr, (char \*) &child->xstate, sizeof(child->xstate)) < 0) {  
 release(&child->lock);  
 release(&wait\_lock);  
 return -1;  
 }  
 if (proc\_time != 0 && copyout(p->pagetable, proc\_time, (char \*) &child\_times, sizeof(child\_times)) < 0) {  
 release(&child->lock);  
 release(&wait\_lock);  
 return -1;  
 }  
 freeproc(child);  
 release(&child->lock);  
 release(&wait\_lock);  
 return child\_pid;  
 }  
 release(&child->lock);  
 }  
 }  
 if (!have\_children || killed(p)) {  
 release(&wait\_lock);  
 return -1;  
 }  
 sleep(p, &wait\_lock);  
 }  
}  
  
int  
toggleScheduler() {  
 if (scheduler\_type == *DEFAULT*) {  
 scheduler\_type = *FCFS*;  
 return 1;  
 } else {  
 scheduler\_type = *DEFAULT*;  
 return 0;  
 }  
}

افزودن پروتوتایپ توابع به فایل defs.h

int getProcTick(int);  
int sysinfo(uint64);  
void update\_process\_times();  
int childWait(uint64,uint64);  
int toggleScheduler();

فراخوانی update\_process\_times در تابع clockintr در فایل trap.c

void  
clockintr()  
{  
 acquire(&tickslock);  
 ticks++;  
 update\_process\_times();  
 wakeup(&ticks);  
 release(&tickslock);  
}

افزودن entry سیستم کال‌های childWait و toggleScheduler به فایل usys.pl

entry("uptime");  
entry("getProcTick");  
entry("sysinfo");  
entry("childWait");  
entry("toggleScheduler");

افزودن پروتوتایپ توابع مربوط به سیستم کال‌های childWait و toggleScheduler به فایل user.h

int uptime(void);  
int getProcTick(int);  
int sysinfo(int);  
int childWait(uint64, uint64);  
int toggleScheduler(void);

افزودن نام و شماره فراخوانی سیستم کال‌های childWait و toggleScheduler به فایل syscall.h

#define SYS\_close 21  
#define SYS\_getProcTick 22  
#define SYS\_sysinfo 23  
#define SYS\_childWait 24  
#define SYS\_toggleScheduler 25

افزودن پروتوتایپ و نام فراخوانی سیستم کال‌های childWait و toggleScheduler به فایل syscall.c

extern uint64 sys\_close(void);  
extern uint64 sys\_getProcTick(void);  
extern uint64 sys\_sysinfo(void);  
extern uint64 sys\_childWait(void);  
extern uint64 sys\_toggleScheduler(void);

[SYS\_close] sys\_close,  
[SYS\_getProcTick] sys\_getProcTick,  
[SYS\_sysinfo] sys\_sysinfo,  
[SYS\_childWait] sys\_childWait,  
[SYS\_toggleScheduler] sys\_toggleScheduler,

پیاده‌سازی تابع فراخوانی در فایل sysproc.c

uint64  
sys\_childWait(void) {  
 uint64 u1;  
 uint64 u2;  
 argaddr(0, &u1);  
 argaddr(1, &u2);  
 return childWait(u1, u2);  
}  
  
  
uint64  
sys\_toggleScheduler(void) {  
 return toggleScheduler();  
}

ساخت فایل تست با نام schedulerTest.c در دایرکتوری user و پیاده‌سازی مراحل تست در تابع main آن

#include "../kernel/types.h"  
#include "../kernel/stat.h"  
#include "user.h"  
  
struct proc\_times {  
 long burst\_time;  
 long turnaround\_time;  
 long waiting\_time;  
};  
  
int main() {  
 int i, n, pid;  
 int processes\_count = 32;  
  
 toggleScheduler();  
 for (i = 0; i < processes\_count; i++) {  
 pid = fork();  
 if (pid < 0)  
 return -1;  
 else if (pid == 0) {  
 for (i = 0; i < 100000000; i++)  
 n++;  
 exit(0);  
 }  
 }  
  
 for (i = 0; i < processes\_count; i++) {  
 struct proc\_times properties;  
 pid = childWait(0, (uint64) &properties);  
 printf("Process %d: {\n Burst Time: %d,\n Waiting Time: %d,\n Turnaround Time: %d\n}\n",  
 pid, properties.burst\_time, properties.waiting\_time, properties.turnaround\_time);  
 }  
  
 exit(0);  
}

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*\* $U/\_schedulerTest*\*