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\
```