گزارش کار پروژه سوم آزمایشگاه سیستم عامل

گروه ۱۳:

سیده زینب پیش بین - ۸۱۰۱۹۹۵۹۷

محمدجواد بشارتی - ۸۱۰۱۹۹۳۸۶

آدرس مخزن github:

https://github.com/JavadBesharati/OS-lab-projects-UT-Spring-2023

شناسه آخرین commit:

6b792b3c17a5b3ced777adbb846989b98acf4bd4

سطح اول: زمانبند نوبت گردشی (Round Robin)

برای این زمانبند تابع زیر در فایل proc.c زده شده است:

```
struct proc*
round_robin(void) // for queue 1 with the highest priority
{
    struct proc *p;
    struct proc *min_p = 0;
    int time = ticks;
    int starvation_time = 0;

    for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
        if (p->state != RUNNABLE || p->queue != 1)
            continue;

        if (p->state != RUNNABLE || p->queue != 1)
            continue;

        int starved_for = time - p->entered_queue;
        if (starved_for > starvation_time) {
            starvation_time = starved_for;
            min_p = p;
        }
    }
    return min_p;
}
```

سطح دوم: زمانبند بخت آزمایی (Lottery)

برای این زمانبند تابع زیر در فایل proc.c زده شده است:

```
404  struct proc*
405  lottery(void)
406  {
407    struct proc* p;
408    int total_tickets = 0;
409
410    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
411      if(p->state != RUNNABLE || p->queue != 2){
412        continue;
413      }
414      total_tickets += p->tickets;
415    }
416
417    int winning_ticket = random_number_generator(1, total_tickets);
418
419    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
420      if(p->state != RUNNABLE || p->queue != 2){
421        continue;
422      }
423      winning_ticket -= p->tickets;
424      if(winning_ticket <= 0){
425        return p;
426      }
427    }
428    return 0;
429  }
```

همچنین برای تولید اعداد تصادفی تابع random_number_generator در فایل proc.c زده شده است:

```
int
random_number_generator(int min, int max)
{
    if (min >= max)
        return max > 0 ? max : -1 * max;
    acquire(&tickslock);
    int random_number, diff = max - min + 1, time = ticks;
    release(&tickslock);
    random_number = (1 + (1 + ((time + 2) % diff ) * (time + 1) * 132) % diff) * (1 + time % max) * (1 + 2 * max % diff);
    random_number = random_number % (max - min + 1) + min;
    return random_number;
}
```

سطح سوم: زمانبند اولین ورود-اولین رسیدگی (FCFS)

برای این زمانبند تابع زیر در فایل proc.c زده شده است:

```
struct proc*
fcfs(void)
{
  struct proc* p;
  struct proc* first_proc = 0;

  int mn = 2e9;

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state != RUNNABLE || p->queue != 3){
      continue;
    }
    if(p->entered_queue < mn){
      mn = p->entered_queue;
      first_proc = p;
    }
  }
```

فراخوانی‌های سیستمی مورد نیاز:

- **تغییر صف پردازه:**

برای این فراخوانی سیستمی تابع زیر در فایل proc.c اضافه شده است:

```
754  void
755  set_proc_queue(int pid, int queue)
756  {
757    struct proc* p;
758
759    acquire(&ptable.lock);
760    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
761      if(p->pid == pid){
762        p->queue = queue;
763      }
764    }
765    release(&ptable.lock);
766  }
```

- **مقدار دهی بلیط بخت آزمایی:**

برای این فراخوانی سیستمی تابع زیر در فایل proc.c اضافه شده است:

```
768  void
769  set_lottery_params(int pid, int ticket_chance){
770    struct proc* p;
771    acquire(&ptable.lock);
772
773    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
774      if(p->pid == pid){
775        p->tickets = ticket_chance;
776      }
777    }
778
779    release(&ptable.lock);
780  }
```

- **چاپ اطلاعات:**

برای این فراخوانی سیستمی تابع زیر در فایل proc.c اضافه شده است:

```
799  void
800  print_all_procs()
801  {
802    struct proc* p;
803    cprintf("name          pid         state        queue    arrive time        ticket      cycle\n");
804    cprintf("..................................................................................\n");
805    acquire(&ptable.lock);
806
807    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
808      if(p->state == UNUSED){
809        continue;
810      }
811      cprintf(p->name);
812
813      for(int i = 0; i < 17 - strlen(p->name); i++){ // 15 instead 11
814        cprintf(" ");
815      }
816
817      cprintf("%d", p->pid);
818
819      for(int i = 0; i < 10 - digit_counter(p->pid); i++){
820        cprintf(" ");
821      }
```

ادامه تابع در ۲ صفحه بعد آمده است:

```
823     char* state = "";
824     if(p->state == 0){
825       state = "UNUSED";
826     }
827     else if(p->state == 1){
828       state = "EMBRYO";
829     }
830     else if(p->state == 2){
831       state = "SLEEPING";
832     }
833     else if(p->state == 3){
834       state = "RUNNABLE";
835     }
836     else if(p->state == 4){
837       state = "RUNNING";
838     }
839     else if(p->state == 5){
840       state = "ZOMBIE";
841     }
842     cprintf(state);
843
844     for(int i = 0; i < 12 - strlen(state); i++){
845       cprintf(" ");
846     }
```

```
847
848       char* queue = "";
849       if(p->queue == 1){
850         queue = "ROUND ROBIN";
851       }
852       else if(p->queue == 2){
853         queue = "LOTTERY";
854       }
855       else if(p->queue == 3){
856         queue = "FCFS";
857       }
858       cprintf(queue);
859
860       for(int i = 0; i < 12 - strlen(queue); i++){
861         cprintf(" ");
862       }
863
864       cprintf("%d", p->entered_queue);
865
866       for(int i = 0; i < 20 - digit_counter(p->entered_queue); i++){
867         cprintf(" ");
868       }
869
870       cprintf("%d", p->tickets);
871
872       for(int i = 0; i < 11 - digit_counter(p->tickets); i++){
873         cprintf(" ");
874       }
875
876       int cycle = ticks - p->entered_queue;
877       cprintf("%d", cycle);
878
879       cprintf("\n");
880     }
881
882     release(&ptable.lock);
883   }
```

برای پیاده‌سازی موارد خواسته شده، پارامترهای زیر نیز در فایل proc.h اضافه شده است:

```
52    int queue;                    // queue number
53    int entered_queue;            // time entered queue
54    int tickets;                  // lottery tickets count
55  };
```

همچنین موارد زیر در فایل proc.c تعریف شده اند:

```
10  #define STARVING_THRESHOLD 8000
11  #define DEFAULT_MAX_TICKETS 30
```

همچنین در تابع allocproc که در فایل proc.c تعریف شده است، خطوط زیر را افزودیم:

```
138   p->entered_queue = ticks;
139   p->queue = 2;
140   p->tickets = random_number_generator(1, DEFAULT_MAX_TICKETS);
```

و در تابع scheduler از فایل proc.c تغییرات زیر اعمال شد:

```
438  void
439  scheduler(void)
440  {
441    struct proc *p;
442    struct cpu *c = mycpu();
443    c->proc = 0;
444
445    for(;;){
446      // Enable interrupts on this processor.
447      sti();
448
449      // Loop over process table looking for process to run.
450      acquire(&ptable.lock);
451
452      fix_queues();
453      p = round_robin();
454
455      if(p == 0){
456        p = lottery();
457      }
458
459      if(p == 0){
460        p = fcfs();
461      }
462
463      if(p == 0){
464        release(&ptable.lock);
465        continue;
466      }
467
468      p->entered_queue = ticks;
469
470      // Switch to chosen process.  It is the process's job
471      // to release ptable.lock and then reacquire it
472      // before jumping back to us.
473      c->proc = p;
474      switchuvm(p);
475      p->state = RUNNING;
476
477      swtch(&(c->scheduler), p->context);
478      switchkvm();
479
480      // Process is done running for now.
481      // It should have changed its p->state before coming back.
482      c->proc = 0;
483
484      release(&ptable.lock);
485
486    }
487  }
```

در فایل sysproc.c هم خطوط زیر افزوده شدند:

```
126  void
127  sys_set_proc_queue(void)
128  {
129    int pid, queue;
130    argint(0, &pid);
131    argint(1, &queue);
132    set_proc_queue(pid, queue);
133  }
134
135  void
136  sys_set_lottery_params(void)
137  {
138    int pid, ticket_chance;
139    argint(0, &pid);
140    argint(1, &ticket_chance);
141    set_lottery_params(pid, ticket_chance);
142  }
143
144  void
145  sys_print_all_procs(void)
146  {
147    print_all_procs();
148  }
```

لازم به ذکر است که از آوردن تصاویر مربوط به اضافه کردن برنامه‌های سطح کاربر و تغییرات مورد نیاز در فایل‌های syscall.c, syscall.h, user.h, Makefile, usys.S و ... به علت تکراری بودن و کاملا مشابه با آزمایش قبلی بودن صرف نظر شده است ولی در صورت نیاز می‌توانید فایل‌های مذکور را بررسی نمایید.

برنامه foo.c:

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

int
main(int argc, char* argv[])
{
    for(int i = 0; i < 3; i++){
        int pid = fork();
        if(pid == 0){
            for(long int j = 0; j < 3000000000; j++){
                int tmp = 3;
                tmp *= 100;
            }
            exit();
        }
    }
    while(wait());
    return 0;
}
```

بررسی صحت عملکرد فراخوانی‌های سیستمی خواسته شده:

```
Group #13:
1. Arshia
2. Zeinab
3. Javad
$ print_all_procs
name           pid         state        queue    arrive time        ticket      cycle
.....................................................................................
init             1         SLEEPING     LOTTERY     2                  27         837
sh               2         SLEEPING     LOTTERY     838                19         1
print_all_procs  3         RUNNING      LOTTERY     839                29         0
$
```

```
$ set_proc_queue 1 3
$ print_all_procs
name           pid         state        queue    arrive time        ticket      cycle
.....................................................................................
init             1         SLEEPING     FCFS        2                  27         5989
sh               2         SLEEPING     LOTTERY     5991               19         0
print_all_procs  5         RUNNING      LOTTERY     5991               7          0
$
```

```
$ set_lottery_params 2 50
$ print_all_procs
name           pid         state        queue    arrive time        ticket      cycle
.....................................................................................
init             1         SLEEPING     FCFS        2                  27         10647
sh               2         SLEEPING     LOTTERY     10648              50         1
print_all_procs  7         RUNNING      LOTTERY     10649              29         0
$
```

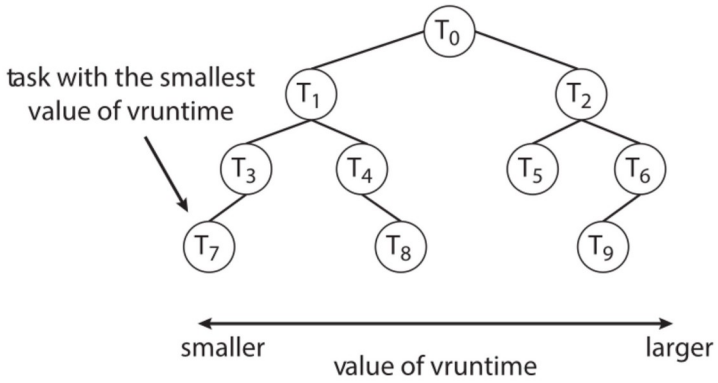
بررسی خروجی print_all_procs **در ازای اجرای** foo **در پس‌زمینه:**

```
Group #13:
1. Arshia
2. Zeinab
3. Javad
$ foo&
$ print_all_procs
name           pid         state        queue    arrive time        ticket      cycle
.....................................................................................
init             1         SLEEPING    LOTTERY     2                  27          1019
sh               2         SLEEPING    LOTTERY     1014               19          7
foo              5         RUNNABLE    LOTTERY     1016               19          5
foo              4         SLEEPING    LOTTERY     461                19          560
foo              6         RUNNABLE    LOTTERY     1015               19          6
foo              7         RUNNABLE    LOTTERY     1020               19          1
print_all_procs  8         RUNNING     LOTTERY     1021               21          0
$
```

پاسخ به سؤالات تشریحی:

۱. با توجه به بدنه تابع sched می بینیم که یک عمل کانتکست سوئیچ رخ داده است. وقتی تابع scheduler داشته باشیم، در این زمان runnable صدا زده می شود که یک پردازه در حالت sched جایگزین پردازه فعلی می شود تا پردازه را انتخاب کرده و به حالت running تبدیل کند .

۲. مطابق توضیحات فصل ۵ کتاب در مورد زمانبندی کاملا منصفانه لینوکس، در لینوکس از یک red black tree برای ساختار این صف استفاده میشود.
به این صورت که از vruntime به عنوان کلید استفاده شده است.)در لینوکس از vruntime برای اولویت بندی استفاده می‌شود که از ترکیب وزن و زمان اجرا به دست می‌آید.
شکل آن در زیر نمایش داده شده است:



۳. در XV٦ فقط یک صف مشترک برای همه داریم که طبق حالت زیر تعریف شده است.

```
struct {
   struct spinlock lock;
   struct proc proc[NPROC];
} ptable;
```

در این حالت وقتی پردازه ای در حال استفاده از پردازنده است باید ptable.lock را گفته باشد .
اما در لینوکس به ازای هر پردازنده یک صف وجود دارد که این صورت نیاز به مکانیزمی جهت load balancing بین پردازنده‌ها داریم. در مقابل وقتی همه پردازه‌ها در صفی مشترک هستند، نیاز به مدیریت کردن دسترسی‌های همزمان است که به کمک lock این بخش مدیریت شده است .

٤. حالتی را در نظر بگیریم که همه پردازه‌ها در حال گرفتن ورودی یا منتظر خروجی دادن هستند، یعنی هیچ پردازه RUNNABLE نداریم. در این حالت اگر وقفه نداشته باشیم و غیرفعال باشد، هیچوقت ورودی و خروجی به پایان نمی‌رسد و برای اینکه به درستی انجام شود، در هر حلقه وقفه برای مدتی فعال میشود، تا این حالت پیش نیاید. در سیستم‌های تک‌هسته‌ای هم نیاز داریم، چون باز همین سناریو ممکن است پیش بیاید و باید وقفه داشته باشیم. یعنی این موضوع وابسته به تعداد هسته‌ها نیست.

٥. در سیستم عامل لینوکس وقفه‌ها به دو دسته top-half و bottom-half تقسیم بندی می شوند.

- top-half: در این دسته وقفه‌ها غیرفعال می‌شوند و حداقل های تنها برقرار هستند. مانند ارتباط با سخت افزار و تغییر flagها در هسته.
- bottom-half: در این دسته‌بندی وقفه‌های دیگر فعال هستند و فعالیت‌های مورد نیاز دیگر وقفه در این دسته اتفاق می‌افتد.

وقفه‌های top-half اولویت بیشتری نسبت به bottom-half دارند. همچنین وقفه‌های bottom-half اولویت بیشتری نسبت به پردازه های دیگر دارند .