به نام خدا

گزارش پروژه سوم آزمایشگاه سیستم عامل

پاییز ۱۴۰۱

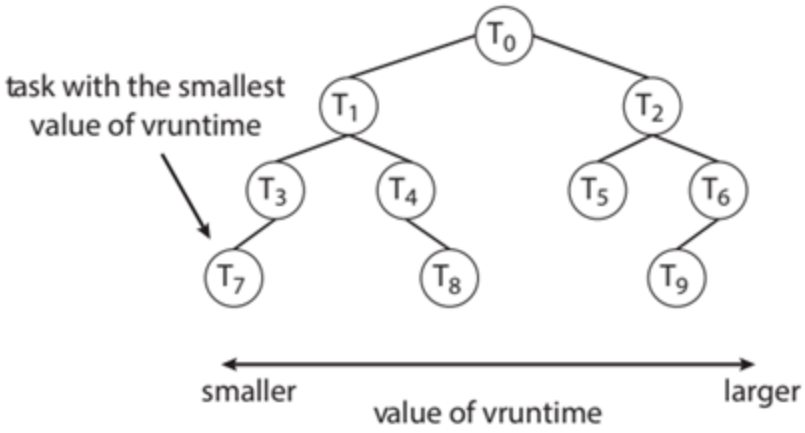
گروه 3: علی هدائی(810199513)، پویا صادقی(810199447)، علی عطاللهی(810199461)

.....

پاسخ به سوالات تشریحی ⭐

1_ زمانی که حالت یک پردازه RUNNABLE میشود این تابع صدا زده میشود. سپس ابتدا شرایط پردازنده بررسی میشود برای مثال با استفاده از تابع holding بررسی میشود که ptable قفل نباشد و حالت پردازه RUNNING نباشد(RUNNABLE باشد) و بعد flag ها بررسی می شود و در صورت وجود مشکل با panic آن را اطلاع میدهد. سپس عمل context switch انجام میشود و context فعلی ذخیره میشود و سپس scheduler اجرا میشود تا پردازه را انتخاب کند و از حالت RUNNABLE به RUNNING تبدیل کند.

2_ برای پیاده سازی CFS در لینوکس به جای استفاده از صف استاندارد از red-black tree استفاده میشود و هر runnable task را وارد این درخت میکند که در واقع یک balanced binary search tree است که کلید آن مقدار vruntime است.(ترکیب وزن و زمان اجرا است.) که شکل آن به صورت زیر است:



زمانی که یک تسک runnable شود وارد این درخت میشود و زمانیکه نباشد از درخت حذف میشود و تسکی که زمان پردازشی کمتری داشته باشد (vruntime کمتر) سمت چپ درخت قرار میگیرند و هرچه بیشتر باشد به سمت راست میرود. سمت چپ ترین node ، بیشترین اولویت را در CFS scheduler دارد و آن را اجرا میکند.

3_ در سیستم عامل xv6 یک صف مشترک برای همه پردازنده ها داریم که تعریف آن در زیر آمده است:

```
10  struct {
11    struct spinlock lock;
12    struct proc proc[NPROC];
13  } ptable;
```

در این ساختمان داده از یک صف از پردازه ها و یک قفل برای مدیریت کردن دسترسی های همزمان استفاده شده است. ولی در سیستم عامل لینوکس هر پردازنده یک صف مخصوص به خود را دارد.

مزیت صف مشترک این است که نیازی مدیریت load بین پردازنده ها نداریم چون همه پردازه ها در یک صف هستند.

نقص آن مشکل در دسترسی همزمان به صف است که برای حل کردن این مشکل از قفل کردن استفاده میکنیم.

4_ وقتی که حالت هیچ پردازه ای RUNNABLE نیست و همه پردازه ها در حال ورودی گرفتن یا آماده خروجی دادن هستند، اگر وقفه وجود نداشته باشد و فعال نباشد، عمل ورودی و خروجی هرگز تمام نمی شود و برای اینکه این اتفاق نیفتد در هر حلقه وقفه برای مدتی فعال میشود تا این حالت اتفاق نیفتد. در سیستم های تک هسته ای نیز این اتفاق ممکن است رخ دهد.

5_ به این دو سطح در سیستم عامل لینوکس FLIH(first level interrupt handler) و SLIH(second level interrupt handler) گفته می شود همچنین به آنها upper half و lower half نیز گفته میشود.

وظیفه FLIH مدیریت وقفه های مهم و ضروری در کمترین زمان است. بدین صورت که یا به پاسخ وقفه می پردازد یا اطلاعات ضروری که در زمان وقوع وقفه موجود است را ذخیره میکند و SLIH را برای آن زمان بندی میکند. برای انجام این روال یک context switch انجام میشود و کد مربوط به مدیریت کننده ی وقفه بارگذاری و اجرا میشود.

SLIH وظیفه رسیدگی به وقفه هایی دارد که زمان بیشتری نیاز دارند. که این کار مانند یک پردازه انجام میشود بدین صورت که یا یک thread در سطح کرنل برای هندلر دارند یا توسط thread pool مدیریت میشوند و سپس در صف قرار میگیرند تا اجرا شوند. و مانند پردازه ها schedule میشوند.

برای رفع مشکل گرسنگی در سیستم های بی درنگ از aging استفاده میکنند بدین صورت هرچه پردازه با اولویت کمتر بیشتر بماند به مرور زمان اولویتش افزایش پیدا میکند و بالاخره اجرا میشود.

.....

سطوح زمان بندی (صف های پیاده سازی شده)

سطح اول: زمانبند نوبت گردشی 🐱

: proc.c

```
10  + #define STARVING_THRESHOLD 8000
```

```
117 +   p->entered_queue = ticks;
118 +   p->queue = 2;
119 +
```

```
329 + fix_queues(void) {
330 +     struct proc *p;
331 +     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
332 +         if (p->state == RUNNABLE)
333 +             if (ticks - p->entered_queue >= STARVING_THRESHOLD) {
334 +                 p->queue = 1;
335 +                 p->entered_queue = ticks;
336 +             }
337 +     }
338 + }
```

```
340 + struct proc* round_robin(void) { // for queue 1 with the highest priority
341 +     struct proc *p;
342 +     struct proc *min_p = 0;
343 +     int time = ticks;
344 +     int starvation_time = 0;
345 +     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
346 +         if (p->state != RUNNABLE || p->queue != 1)
347 +             continue;
348 +         int starved_for = time - p->entered_queue;
349 +         if (starved_for > starvation_time) {
350 +             starvation_time = starved_for;
351 +             min_p = p;
352 +         }
353 +     }
354 +     return min_p;
355 + }
```

```
370 +         fix_queues();
371 +
372 +         p = round_robin();
373 +
374 +         if (p == 0) {
375 +             release(&ptable.lock);
376 +             continue;
377 +         }
378 +         p->entered_queue = ticks;
379 +
380 +         // Switch to chosen process.  It is the process's job
381 +         // to release ptable.lock and then reacquire it
382 +         // before jumping back to us.
383 +         c->proc = p;
384 +         switchuvm(p);
385 +         p->state = RUNNING;
386 +
387 +         swtch(&(c->scheduler), p->context);
388 +         switchkvm();
389 +
390 +         // Process is done running for now.
391 +         // It should have changed its p->state before coming back.
392 +         c->proc = 0;
393 +         release(&ptable.lock);
```

: proc.h

```
52 +   int queue;                    // queue number
53 +   int entered_queue;            // time entered queue
54    };
```

سطح دوم: زمان بند بخت آزمایی (Lottery) 

: proc.c

```
11  + #define DEFAULT_MAX_TICKETS 10
```

```
26  + int
27  + generate_random_number(int min, int max)
28  + {
29  +     if (min >= max)
30  +         return max;
31  +     int rand_num;
32  +     acquire(&tickslock);
33  +     rand_num = (ticks + 2) * (ticks + 1) * (2 * ticks + 3) * 1348 * (ticks % max);
34  +     release(&tickslock);
35  +     rand_num = rand_num % (max - min + 1) + min;
36  +     return rand_num;
37  + }
```

```
133 +    p->tickets = generate_random_number(1, DEFAULT_MAX_TICKETS);
```

```
372 + struct proc* lottery(void) { // for queue #2 and entrance queue
373 +     struct proc *p;
374 +     int total_tickets = 0;
375 +     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
376 +         if (p->state != RUNNABLE || p->queue != 2)
377 +             continue;
378 +         total_tickets += p->tickets;
379 +     }
380 +     int winning_ticket = generate_random_number(1, total_tickets);
381 +     for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
382 +         if (p->state != RUNNABLE || p->queue != 2)
383 +             continue;
384 +         winning_ticket -= p->tickets;
385 +         if (winning_ticket <= 0)
386 +             return p;
387 +     }
388 +     return 0;
389 + }
390 +
```

```
359  393        struct proc *p;
     @@ -371,6 +405,9 @@ scheduler(void) {
371  405
372  406            p = round_robin();
373  407
     408  +         if (p == 0)
     409  +             p = lottery();
     410  +
374  411            if (p == 0) {
375  412                release(&ptable.lock);
376  413                continue;
     @@ -650,4 +687,3 @@ get_callers(int syscall_number)
650  687        }
651  688
```

: proc.h

```
     54  +   int tickets;                // number of lottery tickets
1    55    };
```

سطح سوم: زمان بند اول بهترین کار (BJF) 

در این بخش پروسس های مربوط به این صف را با استفاده get_rank رنک آنها را بدست آورده و در صورتی که رنک کمتری داشتند، برای اجرا برگردانده می‌شوند تا اجرا بشوند.

همچنین به ازای هر واحد زمانی 0.1 واحد به مقدار executed_cycle اضافه می‌شود.

: proc.c

```
+ struct proc*
+ bjf(void)
+ {
+   struct proc* p;
+   struct proc* min_p = 0;
+   float min_rank = MIN_BJF_RANK;
+
+   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
+     if (p->state != RUNNABLE || p->queue != 3)
+       continue;
+     if (get_rank(p) < min_rank){
+       min_p = p;
+       min_rank = get_rank(p);
+     }
+   }
+
+   return min_p;
+ }
```

```
+ float
+ get_rank(struct proc* p)
+ {
+   return
+     p->priority * p->priority_ratio
+     + p->entered_queue * p->arrival_time_ratio
+     + p->executed_cycle * p->executed_cycle_ratio;
+ }
```

```
@@ -464,6 +520,7 @@ yield(void)
{
  acquire(&ptable.lock);  //DOC: yieldlock
  myproc()->state = RUNNABLE;
+ myproc()->executed_cycle += 0.1;
  sched();
  release(&ptable.lock);
}
```

: proc.h

```
+ int priority_ratio;
+ int arrival_time_ratio;
+ int executed_cycle_ratio;
+ float executed_cycle;
+ int priority;
};
```

.....

فراخوانی های سیستمی مورد نیاز

شامل موارد زیر می‌شود :

تغییر صف پردازه 😺

مقداردهی بلیط بخت‌آزمایی

مقدار دهی پارامتر BJF در سطح پردازه

مقدار دهی پارامتر BJF در سطح سیستم

تمام این سیستم‌کال‌ها در فایل‌های مربوط به خود اضافه شدند. همچنین یک فایل برای زدن آنها ساخته شده است.

: defs.h

```
128 + void            set_proc_queue(int, int);
129 + void                    set_lottery_params(int, int);
130 + void            set_a_proc_bjf_params(int, int, int, int);
131 + void            set_all_bjf_params(int, int, int);
128 132
```

: proc.c

```
697 + void
698 + set_proc_queue(int pid, int queue)
699 + {
700 +   struct proc *p;
701 +
702 +   acquire(&ptable.lock);
703 +   for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
704 +   {
705 +     if (p->pid == pid)
706 +       p->queue = queue;
707 +   }
708 +   release(&ptable.lock);
709 + }
```

```
711 + void
712 + set_lottery_params(int pid, int ticket_chance){
713 +   struct proc *p;
714 +
715 +   acquire(&ptable.lock);
716 +   for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
717 +   {
718 +     if (p->pid == pid)
719 +       p->tickets = ticket_chance;
720 +   }
721 +   release(&ptable.lock);
722 + }
```

```
724 + void
725 + set_a_proc_bjf_params(int pid, int priority_ratio, int arrival_time_ratio, int execu
726 + {
727 +   struct proc *p;
728 +
729 +   acquire(&ptable.lock);
730 +   for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
731 +   {
732 +     if (p->pid == pid)
733 +     {
734 +       p->priority_ratio = priority_ratio;
735 +       p->arrival_time_ratio = arrival_time_ratio;
736 +       p->executed_cycle_ratio = executed_cycle_ratio;
737 +     }
738 +   }
739 +   release(&ptable.lock);
740 + }
```

```
742 + void
743 + set_all_bjf_params(int priority_ratio, int arrival_time_ratio, int executed_cycle_ratio)
744 + {
745 +   struct proc *p;
746 +
747 +   acquire(&ptable.lock);
748 +   for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
749 +   {
750 +       p->priority_ratio = priority_ratio;
751 +       p->arrival_time_ratio = arrival_time_ratio;
752 +       p->executed_cycle_ratio = executed_cycle_ratio;
753 +   }
754 +   release(&ptable.lock);
755 + }
```

: proc.h

```
52  + int queue;              // queue number
53  + int entered_queue;          // time entered queue
54  + int tickets;
55  + int priority_ratio;
56  + int arrival_time_ratio;
57  + int executed_cycle_ratio;
58  + float executed_cycle;
59  + int priority;
60  };
```

: set_all_bjf_params.c

```
@@ -0,0 +1,14 @@
+ #include "types.h"
+ #include "stat.h"
+ #include "user.h"
+ #include "fcntl.h"
+
+
+
+ int
+ main(int argc, char *argv[])
+ {
+   set_all_bjf_params(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));
+
+   exit();
+ }
```

: set_a_proc_bjf_params.c

```
@@ -0,0 +1,14 @@
+ #include "types.h"
+ #include "stat.h"
+ #include "user.h"
+ #include "fcntl.h"
+
+
+
+ int
+ main(int argc, char *argv[])
+ {
+   set_a_proc_bjf_params(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]), atoi(argv[4]));
+
+   exit();
+ }
```

: set_lottery_params.c

```
@@ -0,0 +1,14 @@
+ #include "types.h"
+ #include "stat.h"
+ #include "user.h"
+ #include "fcntl.h"
+
+
+
+ int
+ main(int argc, char *argv[])
+ {
+   set_lottery_params(atoi(argv[1]), atoi(argv[2]));
+
+   exit();
+ }
```

: set_proc_queue.c

```
@@ -0,0 +1,14 @@
+ #include "types.h"
+ #include "stat.h"
+ #include "user.h"
+ #include "fcntl.h"
+
+
+
+ int
+ main(int argc, char *argv[])
+ {
+   set_proc_queue(atoi(argv[1]), atoi(argv[2]));
+
+   exit();
+ }
```

: syscall.c

```
109  109    extern int sys_get_parent_pid(void);
     110  + extern int sys_set_proc_queue(void);
     111  + extern int sys_set_lottery_params(void);
     112  + extern int sys_set_a_proc_bjf_params(void);
     113  + extern int sys_set_all_bjf_params(void);
110  114
111  115
112  116    static int (*syscalls[])(void) = {
          @@ -133,8 +137,12 @@ static int (*syscalls[])(void) = {
133  137    [SYS_close]                   sys_close,
134  138    [SYS_find_largest_prime_factor] sys_find_largest_prime_factor,
135  139    [SYS_get_callers]             sys_get_callers,
136       - [SYS_change_file_size]        sys_change_file_size,
     140  + [SYS_change_file_size]        sys_change_file_size,
137  141    [SYS_get_parent_pid]          sys_get_parent_pid,
     142  + [SYS_set_proc_queue]          sys_set_proc_queue,
     143  + [SYS_set_lottery_params]      sys_set_lottery_params,
     144  + [SYS_set_a_proc_bjf_params]   sys_set_a_proc_bjf_params,
     145  + [SYS_set_all_bjf_params]      sys_set_all_bjf_params,
```

: syscall.h

```
27  + #define SYS_set_proc_queue            26
28  + #define SYS_set_lottery_params        27
29  + #define SYS_set_a_proc_bjf_params     28
30  + #define SYS_set_all_bjf_params        29
```

: sysproc.c

```
120  + void
121  + sys_set_proc_queue(void)
122  + {
123  +   int pid, queue;
124  +   argint(0, &pid);
125  +   argint(1, &queue);
126  +   set_queue(pid, queue);
127  + }
```

```
129 + void
130 + sys_set_lottery_params(void)
131 + {
132 +   int pid, ticket_chance;
133 +   argint(0, &pid);
134 +   argint(1, &ticket_chance);
135 +   set_lottery_params(pid, ticket_chance);
136 + }
137 +
```

```
138 + void
139 + sys_set_a_proc_bjf_params(void)
140 + {
141 +   int pid, priority_ratio, arrival_time_ratio, executed_cycle_ratio;
142 +   argint(0, &pid);
143 +   argint(1, &priority_ratio);
144 +   argint(2, &arrival_time_ratio);
145 +   argint(3, &executed_cycle_ratio);
146 +   set_a_proc_bjf_params(pid, priority_ratio, arrival_time_ratio, executed_cycle_rati
147 + }
148 +
```

```
149 + void
150 + sys_set_all_bjf_params(void)
151 + {
152 +   int priority_ratio, arrival_time_ratio, executed_cycle_ratio;
153 +   argint(0, &priority_ratio);
154 +   argint(1, &arrival_time_ratio);
155 +   argint(2, &executed_cycle_ratio);
156 +   set_all_bjf_params(priority_ratio, arrival_time_ratio, executed_cycle_ratio);
157 + }
```

: user.h

```
30 + void set_proc_queue(int, int);
31 + void set_lottery_params(int, int);
32 + void set_a_proc_bjf_params(int, int, int, int);
33 + void set_all_bjf_params(int, int, int);
```

: usys.S

```
36 + SYSCALL(set_proc_queue)
37 + SYSCALL(set_lottery_params)
38 + SYSCALL(set_a_proc_bjf_params)
39 + SYSCALL(set_all_bjf_params)
```

: makefile

```
190 +        _set_a_proc_bjf_params\
191 +        _set_all_bjf_params\
192 +        _set_lottery_params\
193 +        _set_proc_queue\
194 +        _foo\
195 +        _print_procs\
```

```
268 +        set_a_proc_bjf_params.c set_all_bjf_params.c set_lottery_params.c set_proc_queue.c
```

تست سیستم‌کال‌ها :

```
$ print_procs
name       pid       state       queue       arrival_time       tickets     p
atio      e_ratio       a_ratio       rank       exec_cycle
.....
.....
init       1         SLEEPING    Lottery     7                  27         1
         1             1             8.80        8
sh         2         SLEEPING    Lottery     1029               29         1
         1             1             1030.19     2
print_procs4         RUNNING     Lottery     1031               21         1
         1             1             1032.19     2
$ set_proc_queue 2 3
$ print_procs
name       pid       state       queue       arrival_time       tickets     p
atio      e_ratio       a_ratio       rank       exec_cycle
.....
.....
init       1         SLEEPING    Lottery     7                  27         1
         1             1             8.80        8
sh         2         SLEEPING    BJF         3073               29         1
         1             1             3074.39     4
print_procs6         RUNNING     Lottery     3075               29         1
         1             1             3076.10     1
$
```

```
.....
.....
init       1         SLEEPING    Lottery     7                  27         1
         1             1             8.80        8
sh         2         SLEEPING    Lottery     1029               29         1
         1             1             1030.19     2
print_procs4         RUNNING     Lottery     1031               21         1
         1             1             1032.19     2
$ set_proc_queue 2 3
$ print_procs
name       pid       state       queue       arrival_time       tickets     p
atio      e_ratio       a_ratio       rank       exec_cycle
.....
.....
init       1         SLEEPING    Lottery     7                  27         1
         1             1             8.80        8
sh         2         SLEEPING    BJF         3073               29         1
         1             1             3074.39     4
print_procs6         RUNNING     Lottery     3075               29         1
         1             1             3076.10     1
$ set_all_bjf_params 10 10 10
$
```

چاپ اطلاعات 🐶

همانطور که در پروژه قبل انجام دادیم لازم است برای اضافه کردن این سیستم کال فایل هایی را تغییر دهیم:

: Syscall.h

```
31  #define SYS_print_all_procs           30
```

: Syscall.c

```
[SYS_set_a_proc_bjf_params]     sys_set_a_proc_bjf_params,
[SYS_set_all_bjf_params]        sys_set_all_bjf_params,
[SYS_print_all_procs]           sys_print_all_procs,
```

: Defs.h

```
130  void            set_a_proc_bjf_params(int, int, int, int);
131  void            set_all_bjf_params(int, int, int);
132  void            print_all_procs(void);
```

: Sysproc.c

```
159  void
160  sys_print_all_procs(void)
161  {
162    print_all_procs();
163  }
```

: User.h

```
32  void set_a_proc_bjf_params(int, int, i
33  void set_all_bjf_params(int, int, int)
34  void print_all_procs(void);
35
```

:Usys.s

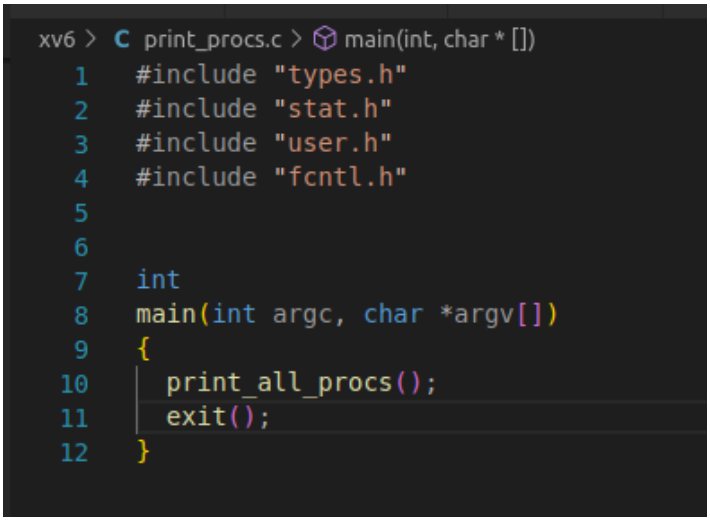
```
39  SYSCALL(set_all_bjf_params)
40  SYSCALL(print_all_procs)
41
```

حال قسمت اصلی آن در proc.c پیاده سازی شده است که در آن پراپرتی های مختلف استراکت proc را با استفاده از cprintf چاپ میکنیم همچنین توابعی مانند printfloat که اعداد اعشاری را تا دو رقم اعشار نمایش میدهد و get_lenght برای بدست آوردن تعداد ارقام عدد برای نمایش بهتر خروجی زده شده است.

```
807  }
808  void
809  print_all_procs()
810  {
811      struct proc *p;
812      cprintf("name       pid       state       queue       arrival_time       tickets     priority_ratio     rank       exec_cycle\n");
813      cprintf("..........................................................................................................................\n");
814      acquire(&ptable.lock);
815      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
816        if (p->state == UNUSED)
817          continue;
818
819
820      cprintf(p->name);
821      for(int i = 0; i < 11 - strlen(p->name); i++) cprintf(" ");
822
823      cprintf("%d", p->pid);
824      for(int i = 0; i < 10 - get_lenght(p->pid); i++) cprintf(" ");
825
826      char* state;
827      if (p->state == 0)
828        state="UNUSED";
829      else if (p->state == 1)
830        state="EMBRYO";
831      else if (p->state == 2)
832        state="SLEEPING";
833      else if (p->state == 3)
834        state="RUNNABLE";
835      else if (p->state == 4)
836        state="RUNNING";
837      else if (p->state == 5)
838        state="ZOMBIE";
839      cprintf(state);
840      for(int i = 0; i < 12 - strlen(state); i++) cprintf(" ");
841
842      char* queue;
843      if (p->queue == 1)
844        queue="RoundRobin";
845      else if (p->queue == 2)
846        queue="Lottery";
847      else if (p->queue == 3)
848        queue="BJF";
849      cprintf(queue);
850      for(int i = 0; i < 12 - strlen(queue); i++) cprintf(" ");
851
852      cprintf("%d", p->entered_queue);
853      for(int i = 0; i < 20 - get_lenght(p->entered_queue); i++) cprintf(" ");
```

```
854
855      int tickets = p->tickets;
856      cprintf("%d", p->tickets);
857      if(tickets < 0)
858        for(int i = 0; i < 12 - get_lenght(tickets*(-1)) - 1; i++) cprintf(" ");
859      else
860        for(int i = 0; i < 12 - get_lenght(tickets); i++) cprintf(" ");
861
862
863
864      cprintf("%d", p->priority_ratio);
865      for(int i = 0; i < 19 - get_lenght(p->priority_ratio); i++) cprintf(" ");
866
867
868      printfloat(get_rank(p));                    float get_rank(struct proc *p)
869      for(int i = 0; i < 11 - get_lenght((int)get_rank(p))-2; i++) cprintf(" ");
870
871      float executed_cycle = p->executed_cycle*10;
872      if(executed_cycle - (int)(executed_cycle) <= 0.5)
873        cprintf("%d", (int)(executed_cycle));
874      else
875        cprintf("%d", (int)(executed_cycle)+1);
876
877      cprintf("\n");
878    }
879    release(&ptable.lock);
880
881  }
```

و سپس برای استفاده از آن در شل برنامه سطح کاربرش می نویسیم و میک فایل را تغییر میدهیم(به UPROGS و EXTRA اضافه میکنیم):



.....

برنامه سطح کاربر 🐶

در نهایت برای تست کلی آن کدی به نام foo زده شد که در آن چند پردازه که عملیات محاسباتی باید انجام دهند نوشته شده است.

```c
xv6 > C foo.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 temp = 3;
                temp*=100;
            }
            exit();
        }
    }
    while (wait());
    return 0;
}
```

و سپس foo را همانطور که گفته شد در پس زمینه اجرا میکنیم و سپس خروجی print_procs را در زیر مشاهده می کنیم که نشاندهنده درست کار کردن صف ها است:

```
$ print_procs
name       pid       state       queue       arrival_time       tickets     p_ratio      e_ratio      a_ratio      rank       exec_cycle
.....................................................................................................................................
init       1         SLEEPING    Lottery     3                  27          1            1            1            4.40       4
sh         2         SLEEPING    RoundRobin  20435              9           1            1            1            20436.30   3
foo        5         RUNNABLE    Lottery     20431              11          1            1            1            21523.67   10917
foo        4         SLEEPING    Lottery     409                9           1            1            1            410.10     1
foo        6         RUNNING     Lottery     20435              11          1            1            1            22437.75   20018
foo        7         RUNNABLE    Lottery     20434              11          1            1            1            21345.82   9108
print_procs12        RUNNING     Lottery     20435              7           1            1            1            20436.00   0
$
```