دانشکده‌ی مهندسی کامپیوتر

سیستم‌های عامل
تمرین‌های سری دوم

علی حیدری، محمدجواد میرشکاری

۱۲ اردیبهشت ۱۳۹۸

فهرست مطالب

۱ فرایند ۱
۱.۱ ۱
۲.۱ ۲
۳.۱ ۳
۴.۱ ۴
۵.۱ ۴
۶.۱ ۵
۷.۱ ۶
۸.۱ ۷

۲ زمان‌بندی ۸
۱.۲ ۸
۲.۲ ۱۰
۳.۲ ۱۲
۴.۲ ۲۲
۵.۲ ۲۲
۶.۲ ۲۲
۷.۲ ۲۳

۱ فرایند

۱.۱

برنامه‌ی `process-run.py` را با پارامترهای `-l 5:100,5:100` اجرا کنید. مصرف CPU باید چگونه باشد؟ (به عبارت دیگر درصد زمانی که CPU در حال استفاده است چقدر است؟) چرا شما آن را می دانید؟ از پارامترهای `-c` و `-p` استفاده کنید تا از درستی آن آگاه شوید.

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 5:100,5:100
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu
  cpu

Process 1
  cpu
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

با توجه به این که هیچ فرایندی از جنس I/O نداریم پس در هیچ زمانی CPU در حالت انتظار نخواهد بود و همه‌ی فرایندها یکی پس از دیگری بدون وقفه اجرا خواهند شد. بنابراین CPU در تمام مدت در حال استفاده است. (درصد استفاده از CPU: ۱۰۰ ٪)

```
ali@DESKTOP:~$ python2.7 process-run.py -l 5:100,5:100 -c -p
Time      PID: 0     PID: 1        CPU        IOs
  1     RUN:cpu      READY          1
  2     RUN:cpu      READY          1
  3     RUN:cpu      READY          1
  4     RUN:cpu      READY          1
  5     RUN:cpu      READY          1
  6        DONE    RUN:cpu          1
  7        DONE    RUN:cpu          1
  8        DONE    RUN:cpu          1
  9        DONE    RUN:cpu          1
 10        DONE    RUN:cpu          1

Stats: Total Time 10
Stats: CPU Busy 10 (100.00%)
Stats: IO Busy  0 (0.00%)
```

۲.۱

حال با پارامترهای `./process-run.py -l 4:100,1:0` اجرا کنید. این پارامترها یک فرایند با چهار دستورالعمل را تعریف می‌کند (همه‌ی آن‌ها از CPU استفاده می‌کنند)، و دیگری به آسانی یک عمل I/O را تعریف می‌کند که تا پایان یافتن آن در انتظار باشد. چه مقدار زمان طول می‌کشد تا هر دو فرایند کامل شوند؟ از پارامترهای `-c` و `-p` استفاده کنید تا از درستی آن آگاه شوید.

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 4:100,1:0
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu
```

```
Process 1
  io

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

فرایند اول طی ۴ مرحله که با CPU کار دارد کار خود را انجام می‌دهد و به حالت DONE می‌رود سپس فرایند دوم که دارای تنها یک دستور و آن هم فراخوانی I/O است اجرا می‌شود. هر فراخوانی I/O به صورت پیش‌فرض ۵ کلاک طول می‌کشد که در این مدت این فرایند بلاک شده و به حالت WAITING می‌رود. پس از ۹ کلاک و در کلاک ۱۰ام همه‌ی فرایندها به حالت DONE می‌رسند.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 4:100,1:0 -c -p
Time      PID: 0     PID: 1        CPU        IOs
  1      RUN:cpu      READY          1
  2      RUN:cpu      READY          1
  3      RUN:cpu      READY          1
  4      RUN:cpu      READY          1
  5         DONE     RUN:io          1
  6         DONE    WAITING                    1
  7         DONE    WAITING                    1
  8         DONE    WAITING                    1
  9         DONE    WAITING                    1
 10*        DONE       DONE

Stats: Total Time 10
Stats: CPU Busy 5 (50.00%)
Stats: IO Busy  4 (40.00%)
```

۳.۱

ترتیب فرایندها را تغییر دهید: `./process-run.py -l 1:0,4:100` . چه اتفاقی می‌افتد؟ آیا تغییر ترتیب اهمیت دارد؟ چرا؟ (همانند قبل از پارامترهای `-c` و `-p` استفاده کنید تا از درستی آن آگاه شوید.)

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 1:0,4:100
Produce a trace of what would happen when you run these processes:
Process 0
  io

Process 1
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

با عوض کردن نوبت فرایندها اتفاق جالبی می‌افتد با توجه به این که هنگامی که یک فرایند درخواست I/O دارد آن فرایند BLOCK شده و به حالت WAITING می‌رود. در این حالت CPU خالی است و می‌تواند فرایند دیگری را اجرا کند پس فرایند دیگر وارد CPU شده و از حالت READY به حالت RUNNING درمی‌آید. در این حالت نسبت به قبل انجام فرایندها ۴ کلاک کم‌تر طول کشید و ۳۳/۳۳ ٪ رشد استفاده از CPU را داشتیم.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 1:0,4:100 -c -p
```

```
Time      PID: 0     PID: 1        CPU        IOs
  1       RUN:io       READY         1
  2      WAITING    RUN:cpu         1          1
  3      WAITING    RUN:cpu         1          1
  4      WAITING    RUN:cpu         1          1
  5      WAITING    RUN:cpu         1          1
  6*        DONE       DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy  4 (66.67%)
```

۴.۱

اکنون برخی از پارامترهای دیگر را بررسی خواهیم کرد. یک پارامتر مهم `-S` است. که تعیین می‌کند سیستم چگونه هنگام فراخوانی I/O واکنش نشان دهد. با تنظیم پارامتر SWITCH ON END سیستم هنگام انجام فرایند I/O به فرایند دیگری نمی‌رود، در عوض منتظر می‌ماند تا فرایند به طور کامل به اتمام برسد. هنگامی که با دو فرایند (`-l 1:0,4:100 -c -S SWITCH_ON_END`)، یکی فراخوانی I/O و دیگری کار با CPU اجرا کنید چه اتفاقی می‌افتد؟

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 1:0,4:100 -c -S SWITCH_ON_END
Time      PID: 0     PID: 1        CPU        IOs
  1       RUN:io       READY         1
  2      WAITING       READY                   1
  3      WAITING       READY                   1
  4      WAITING       READY                   1
  5      WAITING       READY                   1
  6*        DONE    RUN:cpu         1
  7         DONE    RUN:cpu         1
  8         DONE    RUN:cpu         1
  9         DONE    RUN:cpu         1
```

اگر CPU نتواند هنگامی که فرایند در حال اجرا درخواست I/O می‌کند آن را رها کرده و به سراغ اجرای فرایند دیگری برود در زمان اجرای I/O فرایند دیگری اجرا نمی‌شود و این باعث می‌شود که دوباره دو کار پس از ۹ کلاک و در کلاک ۱۰ام تمام شوند و عملکرد بهینه‌ی CPU از دست برود.

۵.۱

حال فرایندهای مشابهی را اجرا کنید اما با تغییر حالت سوئیچ تنظیم به تعویض به فرایند دیگر هنگام وجود فراخوانی I/O در حالت WAITING (`-l 1:0,4:100 -c -S SWITCH ON IO`). چه اتفاقی می‌افتد؟ از پارامترهای `-c` و `-p` استفاده کنید تا از درستی آن آگاه شوید.

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 1:0,4:100 -S SWITCH_ON_IO
Produce a trace of what would happen when you run these processes:
Process 0
  io

Process 1
  cpu
  cpu
  cpu
```

```
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

هنگامی که CPU می‌تواند وقتی که فرایند در حال اجراست و درخواست I/O می‌کند آن را BLOCK کند و در حالت WAITING قرار دهد و در این زمان به اجرای یک فرایند دیگر بپردازد باعث می‌شود که وقت CPU برای انتظار I/O گرفته نشود که آپشن SWITCH_ON_IO این امکان را می‌دهد. در این حالت در مجموع فرایندها بعد از ۵ کلاک و در کلاک ۶ام پایان می‌یابند.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 1:0,4:100 -c -S SWITCH_ON_IO -c -p
Time      PID: 0     PID: 1        CPU        IOs
  1       RUN:io      READY          1
  2      WAITING    RUN:cpu          1          1
  3      WAITING    RUN:cpu          1          1
  4      WAITING    RUN:cpu          1          1
  5      WAITING    RUN:cpu          1          1
  6*        DONE       DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy  4 (66.67%)
```

۶.۱

یک رفتار مهم دیگر این است که هنگام اتمام فراخوانی I/O چکار کند. با `-I IO RUN_LATER` ، زمانی که فراخوانی I/O کامل شود، لزوما فرایندی که آن را فراخوانی کرده بلافاصله اجرایش ادامه نیابد بلکه اجرای هر چه در حال اجرا بود ادامه یابد. چه اتفاقی می‌افتد هنگامی که شما این ترکیب از فرایندها را اجرا کنید؟

(`./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO RUN_LATER -c -p` را اجرا کنید)

آیا منابع سیستم به طور موثر مورد استفاده قرار می گیرد؟

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I
    IO_RUN_LATER -c -p
Time      PID: 0     PID: 1     PID: 2     PID: 3        CPU        IOs
  1       RUN:io      READY      READY      READY          1
  2      WAITING    RUN:cpu      READY      READY          1          1
  3      WAITING    RUN:cpu      READY      READY          1          1
  4      WAITING    RUN:cpu      READY      READY          1          1
  5      WAITING    RUN:cpu      READY      READY          1          1
  6*       READY    RUN:cpu      READY      READY          1
  7        READY       DONE    RUN:cpu      READY          1
  8        READY       DONE    RUN:cpu      READY          1
  9        READY       DONE    RUN:cpu      READY          1
 10        READY       DONE    RUN:cpu      READY          1
 11        READY       DONE    RUN:cpu      READY          1
 12        READY       DONE       DONE    RUN:cpu          1
 13        READY       DONE       DONE    RUN:cpu          1
 14        READY       DONE       DONE    RUN:cpu          1
 15        READY       DONE       DONE    RUN:cpu          1
 16        READY       DONE       DONE    RUN:cpu          1
 17       RUN:io       DONE       DONE       DONE          1
 18      WAITING       DONE       DONE       DONE                     1
 19      WAITING       DONE       DONE       DONE                     1
 20      WAITING       DONE       DONE       DONE                     1
 21      WAITING       DONE       DONE       DONE                     1
```

```
 22*     RUN:io        DONE        DONE        DONE           1
 23     WAITING        DONE        DONE        DONE                      1
 24     WAITING        DONE        DONE        DONE                      1
 25     WAITING        DONE        DONE        DONE                      1
 26     WAITING        DONE        DONE        DONE                      1
 27*       DONE        DONE        DONE        DONE

Stats: Total Time 27
Stats: CPU Busy 18 (66.67%)
Stats: IO Busy  12 (44.44%)
```

در این هنگام اجرای فرایندهای دارای فراخوانی I/O به آخر موکول می‌شود و این باعث می‌شود که فرایندهایی که با CPU کار دارند اول اجرا شده و در آخر هنگامی که فرایندهای دارای فراخوانی I/O بخواهند اجرا شوند در زمانی که فرایندها BLOCK شده و به حالت WAITING می‌رود فرایند دیگری وجود نداشته باشد تا CPU با آن مشغول شود و زمان CPU تلف می‌شود. در این حالت سیستم بهینه عمل نمی‌کند.

۷.۱

حال همان فرایندها را اجرا کنید اما با پارامتر
که بلافاصله فرایندی را اجرا می‌کند که فراخوانی I/O دارد. این رفتار چه تفاوتی دارد؟ چرا ممکن است اجرای دوباره‌ی یک فرآیند که فقط یک I/O را به پایان رسانده است ایده خوبی باشد؟

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I
    IO_RUN_IMMEDIATE -c -p
Time      PID: 0      PID: 1      PID: 2      PID: 3         CPU         IOs
  1       RUN:io       READY       READY       READY           1
  2      WAITING     RUN:cpu       READY       READY           1           1
  3      WAITING     RUN:cpu       READY       READY           1           1
  4      WAITING     RUN:cpu       READY       READY           1           1
  5      WAITING     RUN:cpu       READY       READY           1           1
  6*      RUN:io       READY       READY       READY           1
  7      WAITING     RUN:cpu       READY       READY           1           1
  8      WAITING        DONE     RUN:cpu       READY           1           1
  9      WAITING        DONE     RUN:cpu       READY           1           1
 10      WAITING        DONE     RUN:cpu       READY           1           1
 11*      RUN:io        DONE       READY       READY           1
 12      WAITING        DONE     RUN:cpu       READY           1           1
 13      WAITING        DONE     RUN:cpu       READY           1           1
 14      WAITING        DONE        DONE     RUN:cpu           1           1
 15      WAITING        DONE        DONE     RUN:cpu           1           1
 16*        DONE        DONE        DONE     RUN:cpu           1
 17         DONE        DONE        DONE     RUN:cpu           1
 18         DONE        DONE        DONE     RUN:cpu           1

Stats: Total Time 18
Stats: CPU Busy 18 (100.00%)
Stats: IO Busy  12 (66.67%)
```

با استفاده از این روش که اولویت اجرا با فرایندهای حاوی دستورات فراخوانی I/O هستند، باشد باعث می‌شود که هنگامی که دستور فراخوانی I/O اجرا می‌شود و فرایند BLOCK شده به حالت WAITING می‌رود فرایندهایی که با CPU کار دارند اجرا شوند و CPU بی‌کار نماند که باعث بهبود عملکرد CPU و تمام شدن زودتر همه‌ی کارها (در مجموع) می‌شود.

۸.۱

حال با فرایندهای تصادفی ساختگی اجرا کنید: `-s 1 -l 3:50,3:50` یا `-s 2 -l 3:50,3:50` یا `-s 3 -l 3:50,3:50` . اگر می‌توانید پیش‌بینی کنید که چه اتفاقی می‌افتد. چه اتفاقی می‌افتد هنگامی که شما از پارامتر `-I IO_RUN_IMMEDIATE` استفاده کنید در مقابلِ `-I IO_RUN_LATER` ؟ چه اتفاقی می‌افتد هنگامی که شما از پارامتر `-S SWITCH_ON_IO` استفاده کنید در مقابلِ `-S SWITCH_ON_END` ؟

پاسخ.

```
ali@DESKTOP:~$ python2.7 process-run.py -s 1 -l 3:50,3:50 -c -p -I IO_RUN_IMMEDIATE
    -S SWITCH_ON_IO
Time      PID: 0     PID: 1        CPU        IOs
  1     RUN:cpu      READY          1
  2      RUN:io      READY          1
  3     WAITING   RUN:cpu          1          1
  4     WAITING   RUN:cpu          1          1
  5     WAITING   RUN:cpu          1          1
  6     WAITING      DONE                    1
  7*     RUN:io      DONE          1
  8     WAITING      DONE                    1
  9     WAITING      DONE                    1
 10     WAITING      DONE                    1
 11     WAITING      DONE                    1
 12*       DONE      DONE

Stats: Total Time 12
Stats: CPU Busy 6 (50.00%)
Stats: IO Busy  8 (66.67%)
```

```
ali@DESKTOP:~$ python2.7 process-run.py -s 1 -l 3:50,3:50 -c -p -I IO_RUN_LATER -S
    SWITCH_ON_IO
Time      PID: 0     PID: 1        CPU        IOs
  1     RUN:cpu      READY          1
  2      RUN:io      READY          1
  3     WAITING   RUN:cpu          1          1
  4     WAITING   RUN:cpu          1          1
  5     WAITING   RUN:cpu          1          1
  6     WAITING      DONE                    1
  7*     RUN:io      DONE          1
  8     WAITING      DONE                    1
  9     WAITING      DONE                    1
 10     WAITING      DONE                    1
 11     WAITING      DONE                    1
 12*       DONE      DONE

Stats: Total Time 12
Stats: CPU Busy 6 (50.00%)
Stats: IO Busy  8 (66.67%)
```

```
ali@DESKTOP:~$ python2.7 process-run.py -s 1 -l 3:50,3:50 -c -p -I IO_RUN_IMMEDIATE
    -S SWITCH_ON_END
Time      PID: 0     PID: 1        CPU        IOs
  1     RUN:cpu      READY          1
  2      RUN:io      READY          1
  3     WAITING      READY                  1
  4     WAITING      READY                  1
  5     WAITING      READY                  1
  6     WAITING      READY                  1
  7*     RUN:io      READY          1
  8     WAITING      READY                  1
```

```
  9     WAITING      READY                    1
 10     WAITING      READY                    1
 11     WAITING      READY                    1
 12*       DONE    RUN:cpu          1
 13        DONE    RUN:cpu          1
 14        DONE    RUN:cpu          1

Stats: Total Time 14
Stats: CPU Busy 6 (42.86%)
Stats: IO Busy  8 (57.14%)
```

```
ali@DESKTOP:~$ python2.7 process-run.py -s 1 -l 3:50,3:50 -c -p -I IO_RUN_LATER -S
    SWITCH_ON_END
Time      PID: 0     PID: 1        CPU        IOs
  1     RUN:cpu      READY          1
  2      RUN:io      READY          1
  3     WAITING      READY                    1
  4     WAITING      READY                    1
  5     WAITING      READY                    1
  6     WAITING      READY                    1
  7*     RUN:io      READY          1
  8     WAITING      READY                    1
  9     WAITING      READY                    1
 10     WAITING      READY                    1
 11     WAITING      READY                    1
 12*       DONE    RUN:cpu          1
 13        DONE    RUN:cpu          1
 14        DONE    RUN:cpu          1

Stats: Total Time 14
Stats: CPU Busy 6 (42.86%)
Stats: IO Busy  8 (57.14%)
```

اتفاقی که هنگامی که با پارامترهای `-I IO_RUN_IMMEDIATE` رخ می‌دهد این است که بعد از اجرای یک فرایند آن فرایندی مورد اولویت قرار می‌گیرد که درخواست I/O داشته باشد که این همان‌طور که در سوال ۷.۱ بحث شد باعث بهبود راندمان CPU خواهد شد، اما هنگامی که با پارامترهای `-I IO_RUN_LATER` اجرا می‌شود اجرای فرایندهای حاوی دستورات I/O به آخر موکول می‌شود که همان‌طور که در سوال ۶.۱ بحث شد باعث افت راندمان CPU خواهد شد.
اگر دستور با پارامترهای `-S SWITCH_ON_IO` اجرا شود CPU این امکان را دارد که هنگامی که یک فرایند درخواست I/O دارد آن را BLOCK کند و به اجرای فرایند دیگری مشغول شود که همان‌طور که در سوال ۵.۱ بحث شد باعث بهبود راندمان CPU خواهد شد اما گر با `-S SWITCH_ON_END` اجرا شود دیگر CPU امکان BLOCK کردن یک فرایند هنگام فراخوانی I/O را از دست می‌دهد و همان‌طور که در سوال ۴.۱ بحث شد باعث افت راندمان CPU خواهد شد.