**تکلیف سری اول سیستم عامل**
**سارا برادران ۹۶۲۴۱۹۳**

---

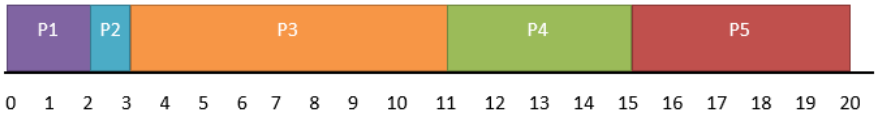
**سوال ۱**

**الف )** در قسمت اول کوانتوم زمانی ۱ میلی ثانیه است لذا زمانبند به ترتیب cpu را به اندازه ۱ میلی ثانیه در اختیار هر پروسس قرار می دهد. تعداد کل پروسس ها ۱۱ می باشد (۱۰ پروسس io bounded و ۱ پروسس cpu bounded ) و زمان تعویض متن ۰.۱ میلی ثانیه است پس در کل $11 \times 0.1 = 1.1$ میلی ثانیه در یک چرخه صرف تعویض متن می گردد به علاوه $11 \times 1 = 11$ میلی ثانیه زمان مفید استفاده از cpu است پس بهره وری $\frac{11}{11+1.1} = 90.9$ درصد می باشد.

**ب )** در قسمت دوم کوانتوم زمانی ۱۰ میلی ثانیه است لذا زمانبند به ترتیب cpu را به اندازه ۱۰ میلی ثانیه در اختیار هر پروسس قرار می دهد اما پروسس های io bounded پس از گذشت ۱ میلی ثانیه از کل زمانی که cpu را در اختیار دارند دستور io صادر می کنند پس مجددا زمانبند پس از گذشت ۱ میلی ثانیه cpu را در اختیار پروسس دیگری قرار خواهد داد اما پروسس cpu bounded از تمام زمان کوانتوم فراهم شده استفاده خواهد کرد پس در یک چرخه از RR می توان گفت $1 \times 10 + 10 \times 1 = 20$ میلی ثانیه cpu کار مفید انجام می دهد و کل زمان یک چرخه برابر $11 \times 0.1 + 10 \times 1 + 1 \times 10 = 21.1$ می گردد پس بهره وری $\frac{20}{21.1} = 94.7$ درصد می باشد.

**سوال ۲**

- **FCFS**




جدول ۱: FCFS

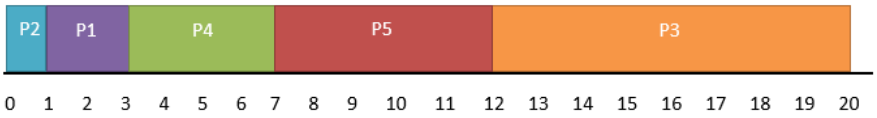
| پروسس | زمان انتظار | زمان بازگشت |
|---|---|---|
| p1 | ۰ | ۲ |
| p2 | ۲ | ۳ |
| p3 | ۳ | ۱۱ |
| p4 | ۱۱ | ۱۵ |
| p5 | ۱۵ | ۲۰ |

Average Response Time : $\frac{0+2+3+11+15}{5} = \frac{31}{5} = 6.2$

Average Turnaround Time : $\frac{2+3+11+15+20}{5} = \frac{51}{5} = 10.2$

---

**SJF •**




جدول ۲: SJF

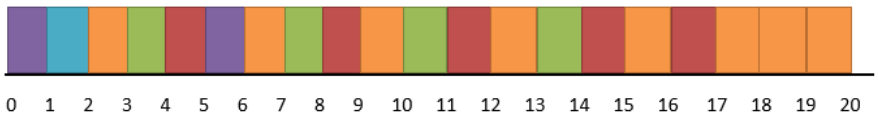
| پروسس | زمان انتظار | زمان بازگشت |
|---|---|---|
| p1 | ۱ | ۳ |
| p2 | ۰ | ۱ |
| p3 | ۱۲ | ۲۰ |
| p4 | ۳ | ۷ |
| p5 | ۷ | ۱۲ |

Average Response Time : $\frac{0+1+3+7+12}{5} = \frac{23}{5} = 4.6$

Average Turnaround Time : $\frac{1+3+7+12+20}{5} = \frac{43}{5} = 8.6$

---

**RR •**




جدول ۳: RR

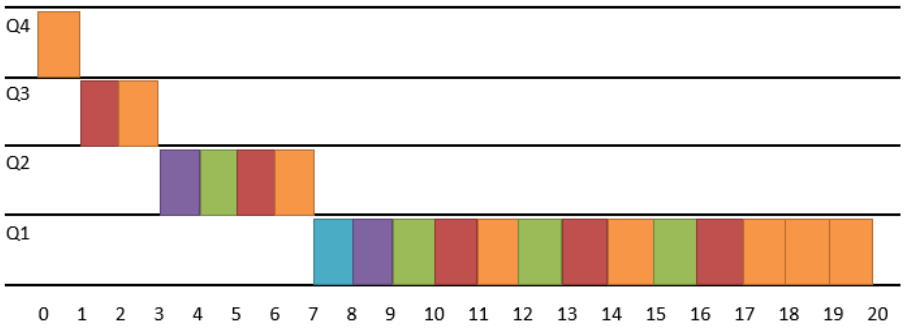
| پروسس | زمان انتظار | زمان بازگشت |
|---|---|---|
| p1 | ۰ | ۶ |
| p2 | ۱ | ۲ |
| p3 | ۲ | ۲۰ |
| p4 | ۳ | ۱۴ |
| p5 | ۴ | ۱۷ |

Average Response Time : $\frac{0+1+2+3+4}{5} = \frac{10}{5} = 2$

Average Turnaround Time : $\frac{6+2+20+14+17}{5} = \frac{59}{5} = 11.8$

---

- **MLFQ**




جدول ۴: MLFQ

| پروسس | زمان انتظار | زمان بازگشت |
|---|---|---|
| p1 | ۳ | ۹ |
| p2 | ۷ | ۸ |
| p3 | ۰ | ۲۰ |
| p4 | ۴ | ۱۶ |
| p5 | ۱ | ۱۷ |

$$\text{Average Response Time : } \frac{0+1+3+4+7}{5} = \frac{15}{5} = 3$$

$$\text{Average Turnaround Time : } \frac{8+9+16+17+20}{5} = \frac{70}{5} = 14$$

**نتیجه** ⇐ همانطور که از محاسبات مشخص است زمانبند RR دارای کمترین میانگین زمان انتظار است.

**سوال ۳**

**FIFO** ⇐ الگوریتم بکار رفته در این مثال FIFO نمی تواند باشد زیرا الگوریتم FIFO منطبق بر این شرط است که اگر cpu در اختیار کار x قرار گرفت تا زمان خاتمه کار x همچنان cpu در اختیار این کار باشد و صرفا پس از خاتمه کار بتوان cpu را به کار دیگری اختصاص داد از آنجایی که در مثال مربوطه بدون اینکه کار A خاتمه یابد cpu پس از ۱۰ واحد زمانی در اختیار کار B قرار گرفته و مجددا بدون اینکه کار B خاتمه یابد cpu پس از ۱۰ واحد زمانی در اختیار کار C قرار گرفته لذا شرط مربوط به الگوریتم FIFO برقرار نبوده پس زمانبند مورد نظر نمی تواند FIFO باشد.

**STCF** ⇐ الگوریتم بکار رفته در این مثال STCF نیز نمی تواند باشد زیرا الگوریتم STCF منطبق بر این شرط است که تنها زمانی که یک کار جدید وارد می شود کار قبلی متوقف شده و زمانبد از میان کار های موجود cpu را به کاری اختصاص خواهد داد که زمان کمتری تا خاتمه نیاز دارد. از آنجایی که در مثال مربوطه پس از ۱۰ واحد زمانی cpu انجام کار A را متوقف و کار B را شروع کرده است پس در صورت استفاده از STCF می توان نتیجه گرفت در ۱۰ واحد زمانی پس از شروع کار A کار B وارد شده است اما مجددا در صورت استفاده از این الگوریتم در لحظه وورد کار B زمانبد از میان کار A و B باید یک کار را برای اختصاص cpu انتخاب کرده باشد و در آن لحظه کار A تنها ۱۰ واحد زمانی تا خاتمه نیاز داشته است در حالی که کار B که جدیدا وارد شده است تا خاتمه ۱۵ واحد زمانی

نیاز داشته است و لذا زمانبد STCF باید مجددا cpu را در اختیار کار A قرار داده باشد نه کار B پس زمانبد مورد نظر نمی تواند STCF باشد.

**RR** ⇐ زمانبد مورد نظر می تواند یک زمانبند RR با کوانتوم ۱۰ واحد زمانی باشد باشد چراکه این زمانبد زمان را به تکه هایی تقسیم کرده و هر بازه زمانی را در اختیار یک کار قرار خواهد داد و این چرخه را تا زمانی تک تک کارها به اتمام برسد ادامه خواهد داد دقیقا مشابه سناریو ای که در مثال مذکور وجود دارد.

**MLFQ** ⇐ زمانبد مورد نظر MLFQ نیز می تواند باشد زیرا اگر فرض کنیم در ابتدا هر سه کار A, B, C در اولویت یکسان قرار داشته باشند و زمانبد بازه های زمانی(کوانتوم) ۱۰ واحدی را به هر کار اختصاص دهد آن گاه چون کار A تمام بازه زمانی را مصرف می کند برای گام بعد به یک اولویت پایین تر منتقل می شود سپس کار B انجام شده و به همبن ترتیب چون تمام بازه را مصرف کرده برای گام بعد به یک اولویت پایین تر منتقل می شود سپس کار C انجام شده و از آنجایی که کار C در بازه ۱۰ واحد زمانی خاتمه می یابد در گام بعد هیچ کاری در صف اولویت بالایی باقی نمی ماند پس زمانبد به ترتیب cpu را به کارهای موجود در صف یک اولویت پایین تر اختصاص می دهد و به این ترتیب ابتدا باقی مانده کار A انجام شده و سپس باقی مانده کار B بر روی cpu انجام می شود و این سناریو می تواند منطبق بر رفتار زمانبد MLFQ باشد.

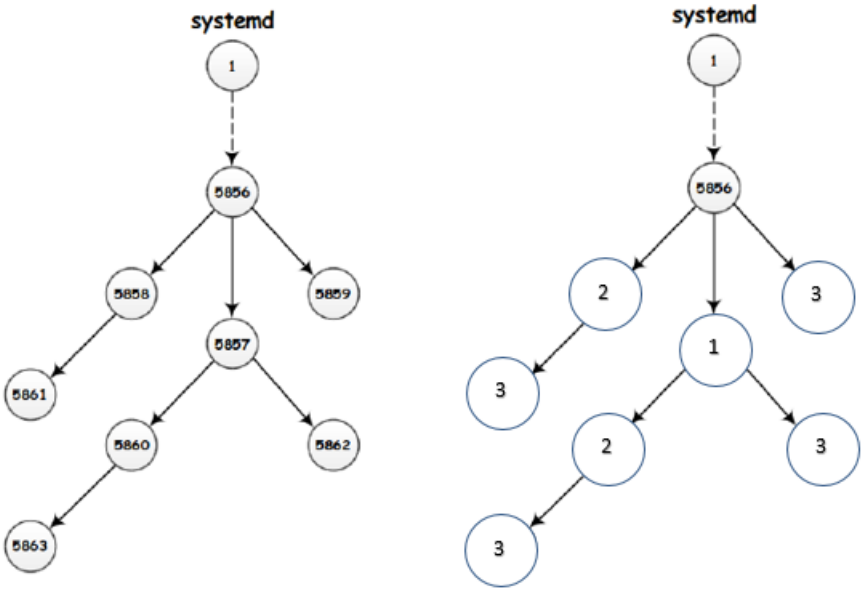
**Lottery** ⇐ زمانبد مورد نظر Lottery نیز می تواند باشد زیرا می توان فرض کرد که زمانبند هر ۱۰ واحد زمانی یکبار به صورت رندوم یک عدد انتخاب کرده و cpu را در اختیار پروسس نظیر آن عدد قرار می دهد آن گاه می توان گفت عدد رندوم اول در بازه نظیر بلیط های پروسس A ، عدد رندوم دوم در بازه نظیر بلیط های پروسس B ، عدد رندوم سوم در بازه نظیر بلیط های پروسس C ، عدد رندوم چهارم مجددا در بازه نظیر بلیط های پروسس A و نهایتا عدد رندوم پنجم در بازه نظیر بلیط های پروسس B بوده است.

**سوال ۴** برنامه نوشته شده در قالب فایل program.c ضمیمه شده است. در حقیقت در این برنامه توسط ۶ دستور fork تعداد ۶ پروسس فرزند به ترتیب ایجاد می گردد از آنجایی که در هر پروسس فرزند پس از ایجاد ، برنامه ls توسط دستور exec جایگزین می شود لذا فرزندان به ادامه اجرای برنامه فعلی نمی پردازند لذا دستورات fork بعدی منجر به ایجاد چندین فرزند نمی شود زیرا ضمن رسیدن به هر دستور fork تنها یک پروسس و آن هم همان پروسس اولیه و والد است که در حال اجرای برنامه نوشته شده می باشد.

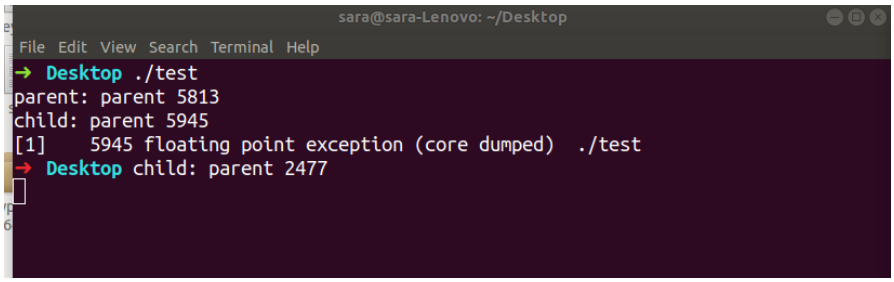
```
this child runs <execl>
I am the clild(1) with pid = 3337
lost+found  myuser  sara  user1  ww-1

this child runs <execlp>
I am the clild(2) with pid = 3338
lost+found  myuser  sara  user1  ww-1

this child runs <execv>
I am the clild(3) with pid = 3339
lost+found  myuser  sara  user1  ww-1

this child runs <execvp>
I am the clild(4) with pid = 3341
lost+found  myuser  sara  user1  ww-1

this child runs <execle>
I am the child(5) with pid = 3342
lost+found  myuser  sara  user1  ww-1

this child runs <execve>
I am the clild(6) with pid = 3343
lost+found  myuser  sara  user1  ww-1

I am parent with pid = 3336
```

شکل ۱: نتیجه اجرای برنامه سوال ۴ (اجرای دستور exec به شیوه های مختلف)

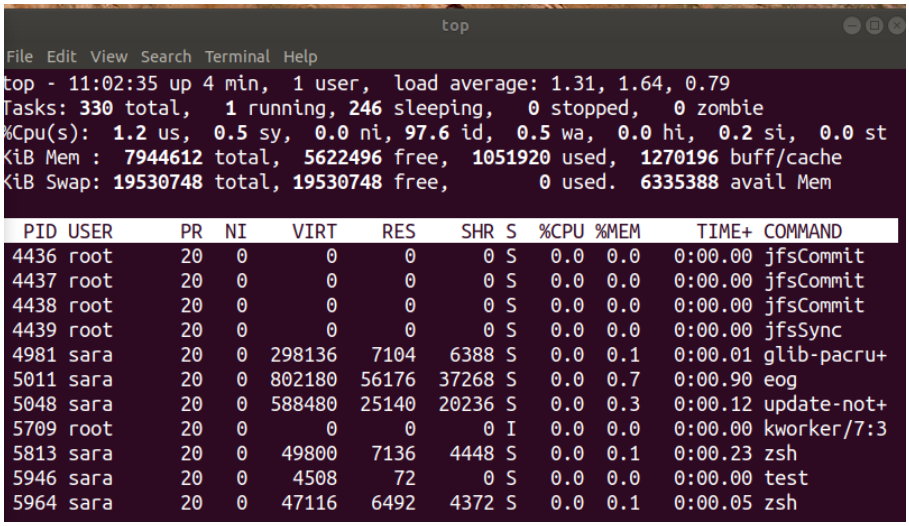
**سوال ۵** درخت حاصله پس از اجرای سه دستور fork بدست خواهد آمد شکل زیر نشان می دهد در هر fork چه فرزندانی ایجاد می گردند.



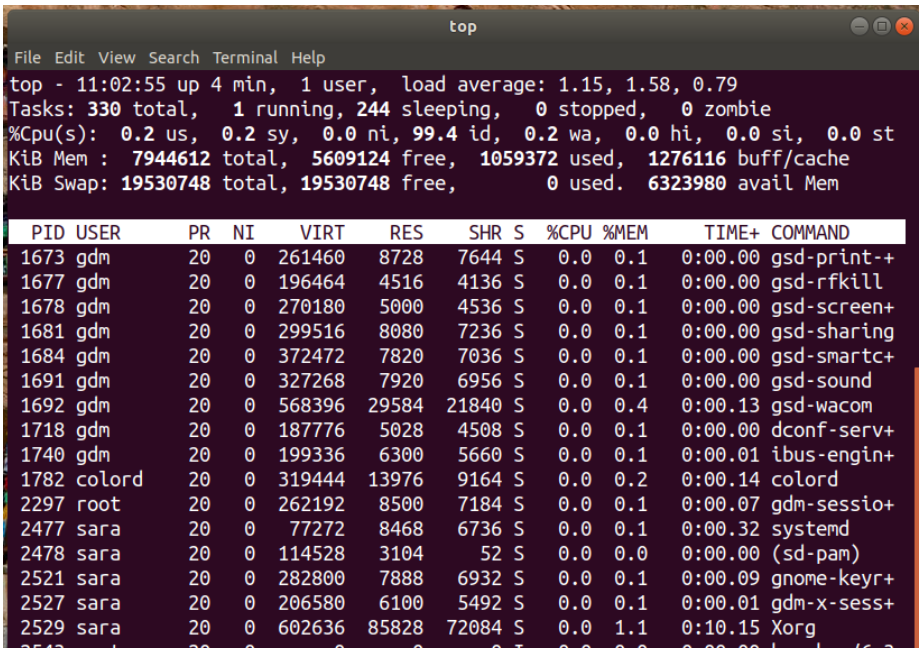
**سوال ۶** ضمن اجرای fork یک پروسس فرزند ایجاد می گردد که همزمان با پروسس والد ازین پس به اجرا برنامه می پردازد. پس از ایجاد پروسس فرزند تابع fork برای پروسس فرزند مقدار ۰ را بازمی گرداند پس برنامه ای که توسط پروسس فرزند در حال اجرا است وارد بلاک if خواهد شد و pid والد خود را چاپ می کند.(۵۹۴۵) به طور همزمان از آنجایی که fork مقدار pid فرزند را در برنامه والد باز می گرداند لذا برنامه در حال اجرا توسط پروسس والد وارد بلاک else می گردد و pid مربوط به والد پروسس parent چاپ می شود(۵۸۱۳) مطابق شکل۳ این والد همان پروسس zsh (z shell) است که برنامه اولیه در آن به اجرا در آمده است. در گام بعدی پروسس فرزند به sleep(4) برخورد کرده و پروسس والد به sleep(2) برخورد کرده است و به طور معمول پروسس والد زودتر از sleep عبور کرده و به دلیل عملیات تقسیم بر صفر موجود یک exception رخ داده و پروسس والد terminate می گردد زمانی که پروسس والد کشته می شود systemd والد جدید پروسس فرزند شده و پس از خاتمه sleep(4) مشاهده می شود که پروسس فرزند pid پروسس systemd را به عنوان والد جدید خود چاپ می نماید.(۲۴۷۷) نتیجه : اگر والد یک پروسس پیش از خاتمه پروسس فرزند خاتمه یابد پروسس systemd والد جدید خواهد بود. با توجه به شکل های ۳ و ۴ که از خروجی برنامه top می باشد می توان مشاهده کرد pid = 2477 متعلق به پروسس systemd و pid = 5813 متعلق به پروسس zsh می باشد.




شکل ۲: در ابتدا پروسس ۵۹۴۵ که همان پروسس اولیه ای است که به اجرا در آمده والد پروسس فرزند ایجاد شده بوده و والد این پروسس اولیه پروسس ۵۸۱۳ می باشد که همان شل است ، پس از خاتمه پروسس ۵۹۴۵ پروسس ۲۴۷۷ که همان پروسس systemd است والد جدید پروسس فرزند خواهد شد




شکل ۳: در برنامه top مشاهده می شود که شناسه پروسس ۵۸۱۳ متعلق به zsh می باشد.




شکل ۴: در برنامه top مشاهده می شود که شناسه پروسس ۲۴۷۷ متعلق به systemd می باشد.

### سوال ۷

**توضیحات مربوط به سوال ۵ کتاب** همانطور که در تصویر زیر واضح است اجرای برنامه با تنظیم SWITCH-ON-IO منجر به این خواهد شد که شبیه ساز رفتار زمانبندی را تداعی سازد که ضمن رسیدن یک پروسس به دستورات مربوط به I/O از میان برنامه های آماده به اجرا (READY state) یک برنامه را انتخاب کرده و تا زمان خاتمه عملیات I/O پروسس قبلی ، cpu را به اجرای آن اختصاص دهد به این صورت از بیکار ماندن cpu جلوگیری می شود.

```
→ Desktop ./process-run.py -l 1:0,4:100 -c -p -S SWITCH_ON_IO
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%)

→ Desktop
```

**توضیحات مربوط به سوال ۴ کتاب** به علاوه همانطور که در تصویر فوق واضح است بر خلاف سوال ۵ اجرای برنامه با تنظیم SWITCH-ON-END منجر به این خواهد شد که شبیه ساز رفتار زمانبندی را تداعی سازد که تنها در صورت اتمام یک پروسس cpu را به دیگر برنامه های آماده اجرا (READY state) اختصاص دهد. و اگر یک پروسس حین اجرا به دستورات مربوط به I/O برخورد نماید cpu تا زمان خاتمه عملیات I/O بیکار و بلا استفاده خواهند ماند. همانطور که در جزئیات اجرا مشخص است استفاده از این روش زمانبندی موجب افزایش زمان اجرای دو برنامه و کاهش بهره وری cpu خواهد شد.

```
→ Desktop ./process-run.py -l 1:0,4:100 -c -p -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

Stats: Total Time 9
Stats: CPU Busy 5 (55.56%)
Stats: IO Busy  4 (44.44%)

→ Desktop
```