سوال ۱)

الف) روش اول که به روش shortest job first هم معروف هست :

خوبی ها:

- با توجه به توضیح خوب اول کوتاه ترین پراسس ها و بعد پراسس های بلند تر رو انجام میده که باعث میشه بازده تا یک حدی بالا بره و تعداد پراسس های بیشتری در زمان کمتری انجام بشن.

بدی ها:

- حتما باید بدونیم که چه مقدار زمان طول میکشه تا یک پراسس کامل انجام بشه (قبل این که بخواد اجرا بشه یا به عبارتی وارد running state بشه که تقریبا غیر ممکن هست)
- Process های طولانی تر زمان waiting بیشتری خواهد داشت و ممکنه بعد یک مدت باعث بشه که به starvation بخورن چون دیرتر از بقیه قرار هست انجام بشن.
- ممکنه به حالت لگ خوردن در اجرای پراسس ها منجر بشن ، خصوصا در اون پراسس های آخری که طولانی تر هستن

ب) روس دوم که به روش round robin هم مشهور هست :

خوبی ها:

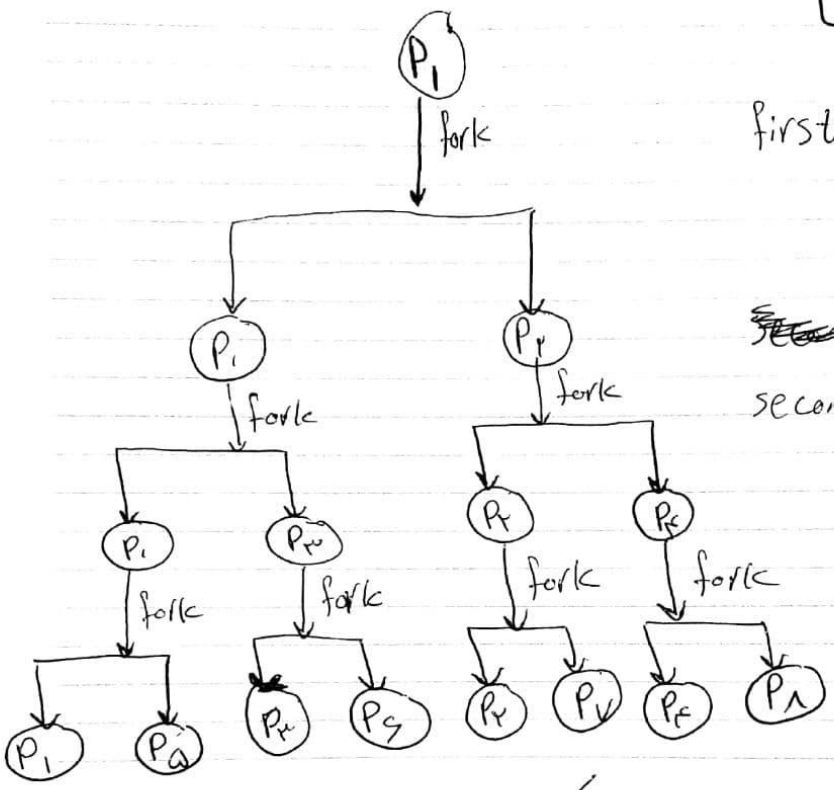
- با توجه به این که هر پراسسی در مدت زمان های مشخصی به cpu دسترسی داره عملا همه ی پراسس ها در یک اولویت یکسان هستن.
- حالت starvation که در sjf رخ میده اینجا رخ نمیده چون حتما همه ی پراسس ها باید یک مت زمان یکسانی اجرا بشن پس هیچ پراسسی جا نمی مونه.

بدی ها:

- با توجه به این که چه مدت زمان t ای در نظر میگیریم ، بازده ممکنه خیلی بیاد پایین خصوصا اگر t زیاد باشه. اگر t خیلی کوتاه باشه باعث میشه زمان زیادی صرف context switching بشه چون تعداد دفعات switch شدن زیاد میشه و عملا نسبت زمان context switching به اجرای پراسس میره بالا که اصلا بهینه نیست و باعث کاهش بهره وری CPU میشه.

سوال دوم)

سؤال ۲



اولین fork ۲ تا process ایجاد می‌کنه و هر دو پراسس
که چه فرق پراسس و چه پراسس فرزند، ادامه‌ی کد رو می‌ران
و هرکدوم fork دوم رو اجرا می‌کنن که ۲ تا پراسس
از هرکدوم تشکیل می‌شه که یعنی ۴ تا process در کل داریم و همه‌ی این
۴ تا ادامه‌ی کد رو انجام می‌دن و fork سوم روی همین و هرکدوم ۲ تا
می‌شن پس در کل می‌شه ۸ تا process

fork و
fork و
fork و

سوال چهارم)

سوال ۴

```
if(fork() && (!fork())) {
    if ( fork() || fork()) {
        fork()
    }
}
```

از if اول که شروع می کنیم اولین بار
که fork می شه ۲ تا حالت فرزند و پدر هست
فرزند که برابر با ۰ می شه و پدر بزرگ تر از ۰ پس
اگر برابر با صفر باشه می شه false و در غیر این صورت می شه true پس
درمورد fork ها باید به این نکته دقت کرد.
توی fork اول از if اول یکی غلطه که باعث می شه از if بیرون بیاد و
S چاپ کنه و fork بعدی درسته که باعث می شه بره شرط And رو چک کنه
که اونم fork هم دو تا هستش و دو نتیجه داره و یکی از if بیرون می یاد و
S چاپ می کنه و یکی دیگه درست هست و وارد if می شه

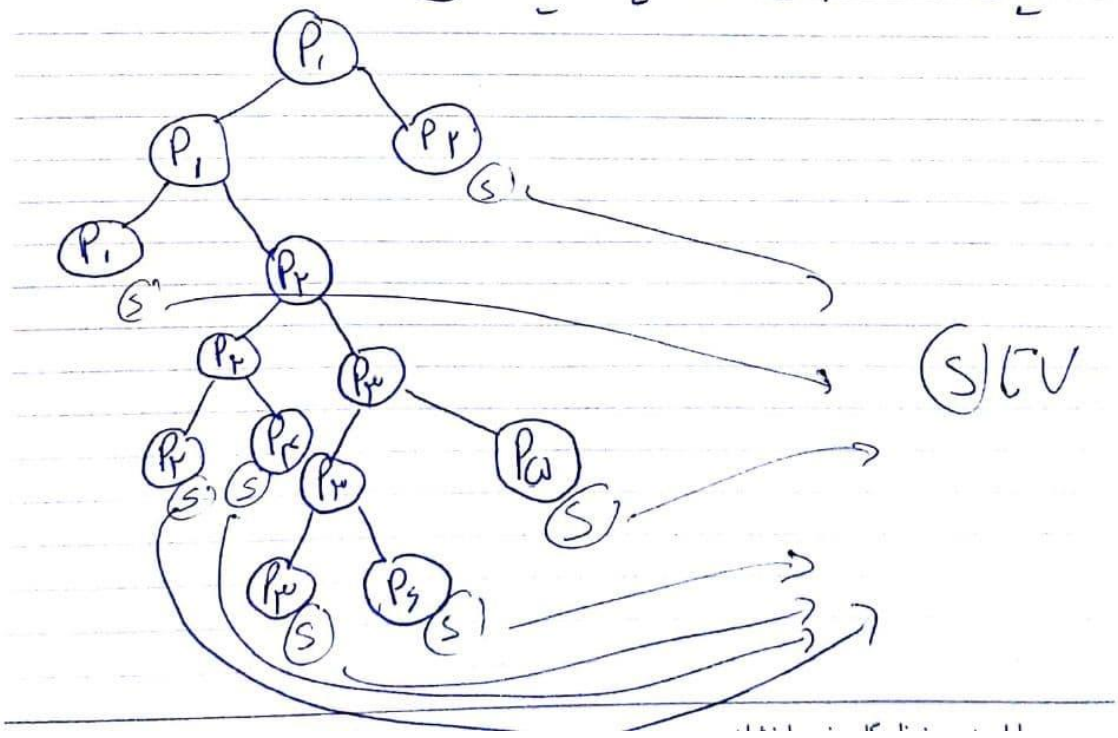
~~[illegible]~~

پس تا این جای کار ۲ تا S دارم.

توی fork اول از if دوم اگر درست باشه ی ره داخل if (چون or هست) و دوباره fork می‌شه و درنتیجه ۲ تا S چاپ می‌کنه و اگر درست نباشه که می‌ره شرط بعدی رو چک کنه و اون جا هم fork می‌شه اگر درست نباشه از if می‌یاد بیرون و S چاپ می‌کنه اگر درست باشه داخل بدنه‌ی if دوباره fork می‌شه و بعد از بیرون اومدن از if چون دوتا هست ۲ بار S چاپ می‌کنه.

پس با همه‌ی این‌ها سیر $2+2+1+2=7$

پس ۷ بار S چاپ می‌کنه



سوال پنجم)

سوال ۵

(a) روش shared memory سریع تر هست قطعاً چون یکبار
از OS می‌خواد که اجازه‌ی دسترسی برای گرفتن یک مموری رو بگیره
مثلاً اگر process یک بار memory دسترسی داره باید process هم از
OS اجازه بگیره تا به memory برسه اول دسترسی پیدا کنه در حالی که
در message passing هر سری باید اجازه از OS گرفته بشه process
ها بفرستن message ها رو دسترسی داشته باشه یا message جدید بفرستن

(b) روش message passing نیاز به رفع conflict نداره نسبت به
shared memory. توی SM اگر cache مربوط به P1 تغییر کنه
باید حتماً توی memory تغییر اعمال بشه تا P2 همه‌ی memory تغییر کرده
تازه اگر بخواد تغییر کرده ممکنه طول بکشه تا از memory روی cache خودش
اون تغییر رو اعمال کنه و همین مراحل ممکنه که باعث بشه مثلاً
C=a+b در P1 اجرا شده باشه و در P2 از C استفاده بشه
که اگر تغییر روی مموری اعمال نشده باشه باعث conflict می‌شه
اما توی message passing به این مشکل نیست چون هر سری تغییر اجازه از
OS اعمال می‌شه و تغییر رو P1 و هم P2 می‌بینه

سوال ششم)

(۶)

a) بله یک thread می‌تونه کار یک process رو بکنه و معمولاً همه‌ی process‌ها
با یک thead می‌تونن شروع بشن حالا این که چند تا thread بعد بشن باز
بستگی داره اما پس یک thread به تنهایی یا به صورت گروهی می‌تونه
کار process رو انجام بده

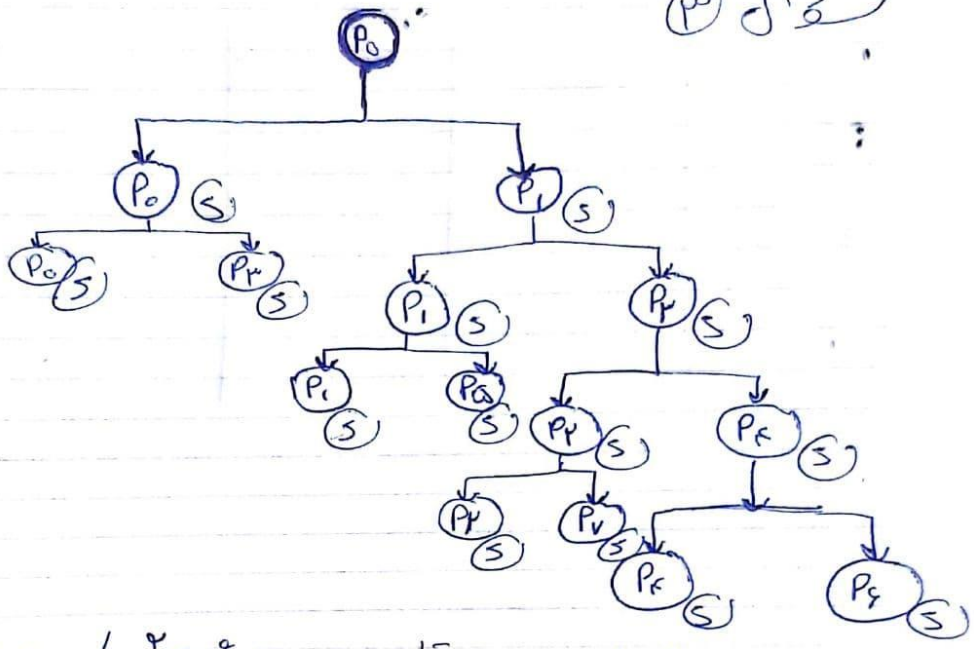
thread از process سبک‌تر هست و PCB ی پردازش رو می‌گیره و
Thread Control block خودش رو داره که ساخت یک thread به مراتب از ساخت
یک process کم‌تر وقت می‌گیره. همون طور که گفتم کاری که یک
process می‌خواد انجام بده معمولاً بین ۱ یا تعدادی thread پخش
می‌شه

b) بله می‌تونن یک space یکسان رو داشته باشن مثل استفاده از shared memory
اصلاً اساساً چون thread‌ها در یک process یکسان address space یکسانی
دارن هزینه‌ی ارتباط بینشون خیلی کم هست.

c) ساخت thread راحت‌تر هست. چون فقط لازمه که
stack ها کپی بشه و بقیه یکسان هست، اما توی process‌ها خوب همه چی فرق
می‌کنه مثل cache جداگانه، heap و ...

سوال سوم)

سوال (۳)



بنابراین ۸ تا process قرار هست داشته باشیم که ۷ تا شون جدید هستن

و با توجه به S هایی که کنار هر کدوم نمایش دادم، سرجمع ۱۴ تا

S چاپ می‌شه