

Operating systems

Homework 1

Seyyed Adel mirsharji

9730653

System configuration:

- Memory: 7.5Gib
- Processor: Intel(R) Core(TM) i5-3230M CPU @ 2.60Ghz x 4
- Graphics: Intel(R) HD Graphics 4000(IVB GT2)
- Operating system: Ubuntu 20.04.1 LTS
- OS-type: 64-bit

(A) پیاده سازی :

در مرحله اول فایلی که در آن تعداد فرزندان هر سطح و دستورات نوشته شده (Command.txt) باز میشود و خط به خط خوانده میشود و هر خط در یک آرایه دو بعدی (line) ذخیره میشود.

پس از بدست آوردن تعداد فرزندان سطح یک و دو و دستورات سه پایپ ساخته میشود:

1. P1 برای نوشتن دستورات توسط فرآیند اصلی و خواندن آنها در فرزند سطح یک.
2. P2 برای نوشتن نتیجه در سطح یک و خواندن آنها در فرآیند اصلی.
3. P3 برای نوشتن نتیجه اجرای دستور (به صورت پیشفرض دستورات (sleep) در نظر گرفته شده اند) در فرزند سطح دو و خواندن آنها در فرزند سطح یک.

در مرحله بعد دستورات توسط فرآیند اصلی در پایپ اول نوشته میشوند و سپس به تعداد گفته شده فرزندان سطح یک (fork) میزنیم و فرزندان سطح یک ساخته میشوند.

تنها نکته این است که در ابتدا مقداری غیر صفر به pid که عدد برگردانده شده توسط fork است میدهیم که با شرط مخالف صفر بودن آن در for loop فرزند فقط برای فرآیند اصلی ایجاد شود و از ایجاد فرزندان سطح دو فعلا خودداری شود چون پس از اولین fork مقدار pid برای فرآیند اصلی (pid) فرزند ایجاد شده و مخالف صفر است و برای فرزند برابر صفر.

پس تا اینجا فرزندان سطح یک ایجاد شدند.

در مرحله بعد در فرآیند اصلی هر فرزند که نتیجه اجرای دستورات داده شده به آنرا در پایپ بنویسد و به اتمام برسد در فرآیند اصلی از پایپ خوانده و در command line چاپ میشود و فرآیند اصلی منتظر نتیجه دستورات بعدی در فرزند بعدی میشود. پس با هر بار به اتمام رسید دستورات و کار فرزند نتیجه در پایپ نوشته و در فرآیند اصلی و خوانده میشود. در فرزندان سطح یک به تعداد مورد نظر دستور از پایپی که فرآیند اصلی در آن دستورات را نوشته دستورات خوانده میشوند و در ادامه برای هر دستور یک فرزند سطح دوم ایجاد شده و دستور را اجرا میکند. در اینجا باز هم فرزند سطح یک منتظر میماند و هر فرزند سطح دو که دستور را اجرا کند و نتیجه را در پایپ بنویسد فرزند سطح یک آنرا از پایپ خوانده و در پایپ برای فرآیند اصلی مینویسد و با اتمام تمام فرزندان در سطح دو خود نیز به پایان میرسد و فرآیند اصلی هم بعد از اتمام تمامی فرزندان اتمام یافته و برنامه بسته میشود.

اما نحوه اجرای دستورات در فرزند سطح دوم به اینصورت است که با فراخوانی `execv` فایل اجرایی (EXEC.out) را اجرا و ارگومانهای مورد نیاز آنرا میدهد. حال فایل EXEC.out یک فایل اجرایی از فایل EXEC.c است که در آن یک دستور `usleep` قرارداده شده (دراصل `usleep` به میکروثانیه توقف میکند اما با ضرب زمان ورودی در 1000 توقف به صورت میلی ثانیه انجام میشود) تا به اندازه زمانهای مورد نظر برای هر دستور در فایل دستورات بر حسب میلی ثانیه

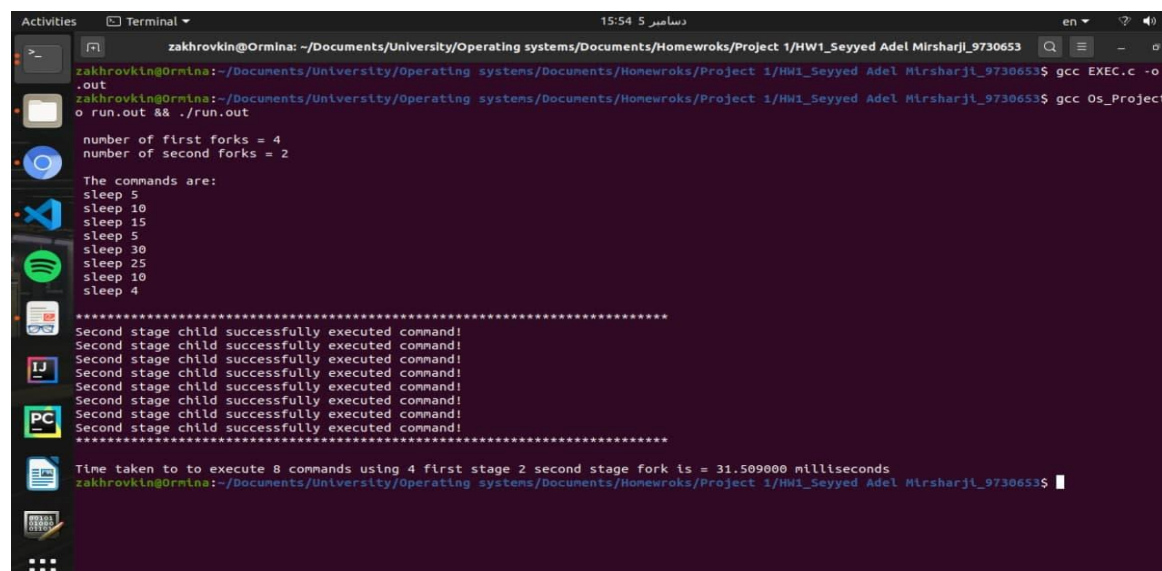
توقف کند و سپس نتیجه اجرا دستور را در پایپ برای والدش که فرزند سطح یک است بنویسد. برای انجام اینکار باید زمان برحسب میکروثاینه و عدد درگاه نوشتن پایپ سوم (writing end) به EXEC.out بصورت آرگومان داده شود (منظور از عدد درگاه نوشتن پایپ سوم همان P3[1] است) تا نتیجه اجرای دستور در پایپ نوشته شود.

علت استفاده از EXEC.out این است که دستور sleep در command line یا به عبارت دیگر در جریان stdout چیزی قرار نمیدهد و برای برگرداندن نتیجه اجرای دستور توسط پایپ به والد پیاده سازی میتواند به این شکل باشد. اما اگر دستوری مثل (ls) که در command line چاپ میکند باشد میتوان با استفاده از dup2 و خواندن و نوشتن در جریان stdout نتیجه را از سطح دو به یک برگرداند و نیازی به EXEC.out نباشد.

(B) مقایسه مدت زمان اجرا:

```
1 4 2
2 sleep 5
3 sleep 10
4 sleep 15
5 sleep 5
6 sleep 30
7 sleep 25
8 sleep 10
9 sleep 4
```

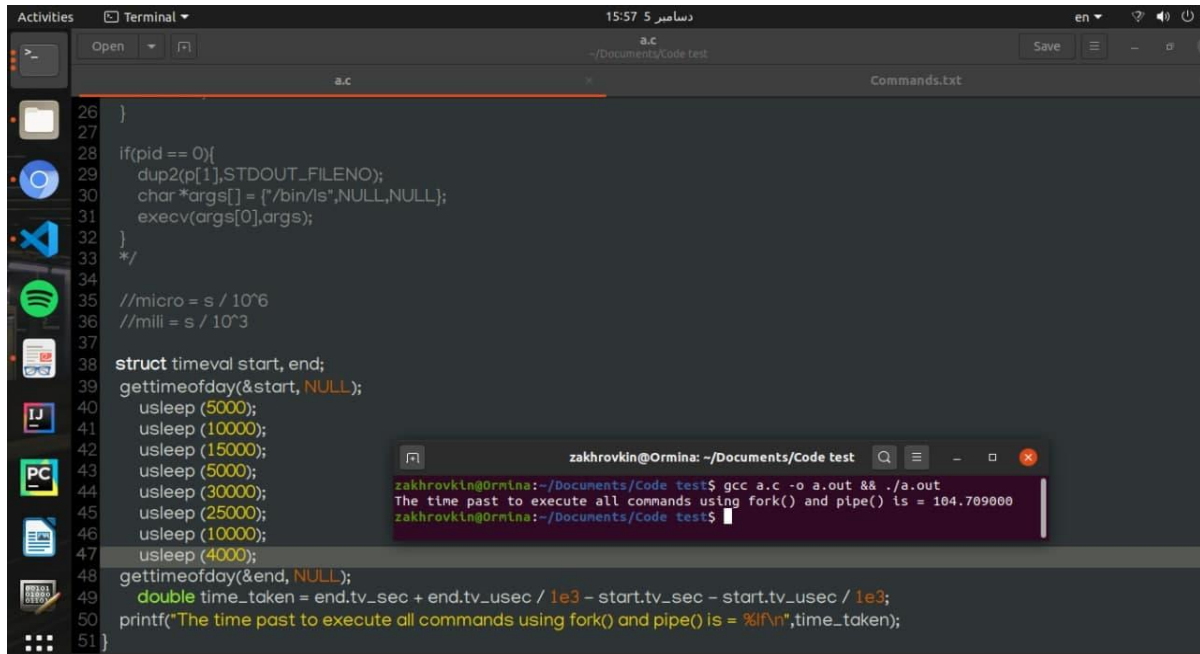
Fork & pipe:



```
zakhrovkin@Ormlna: ~/Documents/University/Operating systems/Documents/Homeworks/Project 1/HW1_Seyyed Adel Mirsharji_9730653
zakhrovkin@Ormlna:~/Documents/University/Operating systems/Documents/Homeworks/Project 1/HW1_Seyyed Adel Mirsharji_9730653$ gcc EXEC.c -o EXEC.out
zakhrovkin@Ormlna:~/Documents/University/Operating systems/Documents/Homeworks/Project 1/HW1_Seyyed Adel Mirsharji_9730653$ gcc Os_Project.c -o run.out && ./run.out
number of first forks = 4
number of second forks = 2
The commands are:
sleep 5
sleep 10
sleep 15
sleep 5
sleep 30
sleep 25
sleep 10
sleep 4
*****
Second stage child successfully executed command!
Second stage child successfully executed command!
Second stage child successfully executed command!
Second stage child successfully executed command!
Second stage child successfully executed command!
Second stage child successfully executed command!
Second stage child successfully executed command!
*****
Time taken to to execute 8 commands using 4 first stage 2 second stage fork is = 31.509000 milliseconds
zakhrovkin@Ormlna:~/Documents/University/Operating systems/Documents/Homeworks/Project 1/HW1_Seyyed Adel Mirsharji_9730653$
```

Time taken by executing whit fork and pipe = 31.50900 milliseconds

Sequential:



```
26 }
27
28 if(pid == 0){
29     dup2(p[1],STDOUT_FILENO);
30     char *args[] = {"/bin/ls",NULL,NULL};
31     execv(args[0],args);
32 }
33 */
34
35 //micro = s / 10^6
36 //milli = s / 10^3
37
38 struct timeval start, end;
39 gettimeofday(&start, NULL);
40 usleep (5000);
41 usleep (10000);
42 usleep (15000);
43 usleep (5000);
44 usleep (30000);
45 usleep (25000);
46 usleep (10000);
47 usleep (4000);
48 gettimeofday(&end, NULL);
49 double time_taken = end.tv_sec + end.tv_usec / 1e3 - start.tv_sec - start.tv_usec / 1e3;
50 printf("The time past to execute all commands using fork() and pipe() is = %f\n",time_taken);
51 }
```

zakhrovkin@Ormina: ~/Documents/Code test

```
zakhrovkin@Ormina:~/Documents/Code test$ gcc a.c -o a.out && ./a.out
The time past to execute all commands using fork() and pipe() is = 104.709000
zakhrovkin@Ormina:~/Documents/Code test$
```

Time taken by executing sequential = 104.70900 milliseconds

اجرا با fork = 31.5 میلی ثانیه اجرا به صورت خطی = 104.7 میلی ثانیه

پس اجرای دستورات با **fork** به علت اجرای دستورات بصورت موازی زمان کمتری نسبت به حالت خطی و پشت سرهم دارد. در مثال فوق نیز واضح است که زمان اجرا با **fork** تقریباً 2/7 زمان اجرای خطی است.

(C) اجرا با پیکربندیهای مختلف:

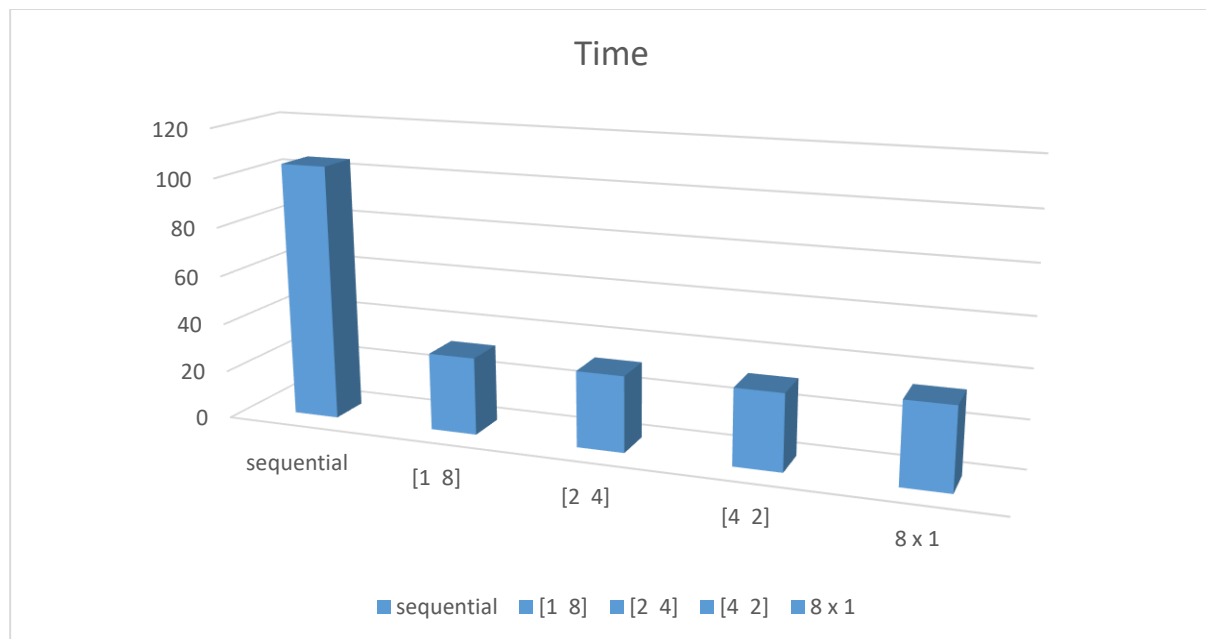
دستورات:

```

2 sleep 5
3 sleep 10
4 sleep 15
5 sleep 5
6 sleep 30
7 sleep 25
8 sleep 10
9 sleep 4

```

Number of forks of 1 st and 2 nd level	Time taken to execute above commands
Sequential	104.7(ms)
1 8	31.9(ms)
2 4	31.31(ms)
4 2	31.5(ms)
8 1	33.92(ms)



طبق مقایسه بالا بهترین ترکیب بر اساس کمترین زمان (2 4) است دو فرزند سطح اول و چهار فرزند سطح دوم.