HW2

姓名: 陈锐林, 学号:21307130148

2023 年 9 月 18 日

Chapter4

Question1:

执行代码”python3 process-run.py -l 5:100,5:100” 后，CPU 利用率应该是 100%，因为在该程序中的”<x:y>”，x 是指令数，y 是使用到 CPU 的占比。而这里两个进程占比都是 100%。使用-c 查看也如此：

```
chenrl959@SK-20210701MSSI:~/NEWHW/ostep-homework/cpu-intro$ python3 process-run.py -l 5:100,5:100 -c
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
```

Question2:

首先在该实验中假定 I/O 处理需要 5 时间，执行代码”python3 process-run.py -l 4:100,1:0” 后，前 4 条指令用时 4，后一条从 io 开始、处理、结束，共需时间 7。所以总时间为 11。-c 查看后符合:

```
chenrl959@SK-20210701MSSI:~/NEWHW/ostep-homework/cpu-intro$ python3 process-run.py -l 4:100,1:0 -c
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       BLOCKED                           1
  7          DONE       BLOCKED                           1
  8          DONE       BLOCKED                           1
  9          DONE       BLOCKED                           1
 10          DONE       BLOCKED                           1
 11*         DONE   RUN:io_done             1
```

Question3

交换了执行顺序之后，应该出现 CPU 在两个进程中切换，因为 IO 被堵塞期间是可以拿来处理第二个进程的指令的。这时候所用时间为 1+5(后一进程执行 4)+1=7。-c 查看后符合：

```
chenrl959@SK-20210701MSSI:~/NEWHW/ostep-homework/cpu-intro$ python3 process-run.py -l 1:0,4:100 -c
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED       RUN:cpu             1             1
  3        BLOCKED       RUN:cpu             1             1
  4        BLOCKED       RUN:cpu             1             1
  5        BLOCKED       RUN:cpu             1             1
  6        BLOCKED          DONE                           1
  7*   RUN:io_done          DONE             1
```

Question4

如果更改标签"SWITCH_ON_END"，即 CPU 不会在一个进程完成前切换，所以仍然执行上面的指令，时间还会是 Question2 中的 11。-c 查看后符合：

```
chenrl959@SK-20210701MSSI:~/NEWHW/ostep-homework/cpu-intro$ python3 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        BLOCKED         READY                           1
  3        BLOCKED         READY                           1
  4        BLOCKED         READY                           1
  5        BLOCKED         READY                           1
  6        BLOCKED         READY                           1
  7*   RUN:io_done         READY             1
  8          DONE       RUN:cpu             1
  9          DONE       RUN:cpu             1
 10          DONE       RUN:cpu             1
 11          DONE       RUN:cpu             1
```

Question5

如果更改标签"SWITCH_ON_IO"，即 CPU 会进行切换，所以会回到 Question3 中的时间 7。-c 查看后符合：

```
chenrl959@SK-20210701MSSI:~/NEWHW/ostep-homework/cpu-intro$ python3 process-run.py -l 1:0,4:100 -c -S  SWITCH_ON_IO
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED       RUN:cpu             1             1
  3        BLOCKED       RUN:cpu             1             1
  4        BLOCKED       RUN:cpu             1             1
  5        BLOCKED       RUN:cpu             1             1
  6        BLOCKED          DONE                           1
  7*   RUN:io_done          DONE             1
```

Question6

这个问题中有 4 个进程，通过改标签我们让 I/O 完成时发出的进程不会马上执行，这会导致：I/O 准备好了，但是 CPU 仍然在执行其他进程，最后无法很好地利用 I/O 堵塞的时间，导致额外的时间开销。-c 查看后符合：首个进程的 I/O 堵塞多在后面，最后 CPU 和 IO 利用率惨淡。

```
chenrl959@SK-20210701MSSI:~/NEWHW/ostep-homework/cpu-intro$ python3 process-run.py -l 3:0,5:100,5:1
00,5:100 -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        BLOCKED       RUN:cpu         READY         READY            1            1
  3        BLOCKED       RUN:cpu         READY         READY            1            1
  4        BLOCKED       RUN:cpu         READY         READY            1            1
  5        BLOCKED       RUN:cpu         READY         READY            1            1
  6        BLOCKED       RUN:cpu         READY         READY            1            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          DONE            1
 18         RUN:io          DONE          DONE          DONE            1
 19        BLOCKED          DONE          DONE          DONE                         1
 20        BLOCKED          DONE          DONE          DONE                         1
 21        BLOCKED          DONE          DONE          DONE                         1
 22        BLOCKED          DONE          DONE          DONE                         1
 23        BLOCKED          DONE          DONE          DONE                         1
 24*   RUN:io_done          DONE          DONE          DONE            1
 25         RUN:io          DONE          DONE          DONE            1
 26        BLOCKED          DONE          DONE          DONE                         1
 27        BLOCKED          DONE          DONE          DONE                         1
 28        BLOCKED          DONE          DONE          DONE                         1
 29        BLOCKED          DONE          DONE          DONE                         1
 30        BLOCKED          DONE          DONE          DONE                         1
 31*   RUN:io_done          DONE          DONE          DONE            1

Stats: Total Time 31
Stats: CPU Busy 21 (67.74%)
Stats: IO Busy  15 (48.39%)
```

Question7

和 Question6 相比，这里很明显会使得总耗时最短。因为这样我们能保证利用到 I/O 的首个进程更快地开始下一条指令，保证先进入 I/O 的堵塞再切换进程，达到更短的时间。-c 后符合：此时 CPU 利用率最高，CPU 在不同的进程间游龙，自由切换。

```
chenrl959@SK-20210701MSSI:~/NEWHW/ostep-homework/cpu-intro$ python3 process-run.py -l 3:0,5:100,5:1
00,5:100 -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        BLOCKED       RUN:cpu         READY         READY            1            1
  3        BLOCKED       RUN:cpu         READY         READY            1            1
  4        BLOCKED       RUN:cpu         READY         READY            1            1
  5        BLOCKED       RUN:cpu         READY         READY            1            1
  6        BLOCKED       RUN:cpu         READY         READY            1            1
  7*   RUN:io_done          DONE         READY         READY            1
  8         RUN:io          DONE         READY         READY            1
  9        BLOCKED          DONE       RUN:cpu         READY            1            1
 10        BLOCKED          DONE       RUN:cpu         READY            1            1
 11        BLOCKED          DONE       RUN:cpu         READY            1            1
 12        BLOCKED          DONE       RUN:cpu         READY            1            1
 13        BLOCKED          DONE       RUN:cpu         READY            1            1
 14*   RUN:io_done          DONE          DONE         READY            1
 15         RUN:io          DONE          DONE         READY            1
 16        BLOCKED          DONE          DONE       RUN:cpu            1            1
 17        BLOCKED          DONE          DONE       RUN:cpu            1            1
 18        BLOCKED          DONE          DONE       RUN:cpu            1            1
 19        BLOCKED          DONE          DONE       RUN:cpu            1            1
 20        BLOCKED          DONE          DONE       RUN:cpu            1            1
 21*   RUN:io_done          DONE          DONE          DONE            1

Stats: Total Time 21
Stats: CPU Busy 21 (100.00%)
Stats: IO Busy  15 (71.43%)
```

Question8

对于题目给出的这几组随机生成进程，我都进行了尝试，最后结果如下，该结果表明：使用 IO_RUN_IMMEDIATE 往往会表现得比 IO_RUN_LATER 更佳，使用 SWITCH_ON_IO 会比 SWITCH_ON_END 更佳。效果更佳即有更短的执行时间和更高的 CPU 利用率。

seed	IO_RUN_()	SWITCH_ON_()	Total Time	CPU busy
1	LATER	IO	15	53.3%
1	LATER	END	18	44.4%
1	IMMEDIATE	END	18	44.4%
1	IMMEDIATE	IO	15	53.3%
2	LATER	IO	16	62.5%
2	LATER	END	30	33.3%
2	IMMEDIATE	END	30	33.3%
2	IMMEDIATE	IO	16	62.5%
3	LATER	IO	18	50.0%
3	LATER	END	24	37.5%
3	IMMEDIATE	END	24	37.5%
3	IMMEDIATE	IO	17	52.9%

Chapter5
说明

这章的作业都要求 fork 后在父进程/子进程中执行操作，为了避免冗余，不贴完整代码，只会在每题里说明父进程/子进程分别做了什么操作。

Question1

这题是最基本的，我设一变量 t，初始化为 100，在父进程中 $t = t/2$，在子进程中 $t = t * 2$，然后输出结果。我们会发现两个进程对 t 的改变是互不干扰的，在两个进程刚开始时，t 都等于初始值 100。运行结果如下：

```
Before change,in parent: t = 100
In parent,t=50
Before change,in child: t = 100
In child,t=200
```

Question2

这题先打开一个文件描述符，"fd = open("./q2.txt",O_CREAT|O_RDWR|O_TRUNC,S_IRWXU)"。在子进程和父进程中，都调用 write，写入不同的信息。我们会发现两个进程都是可以调用 fd 的，并发写入时，输入都会被保留。运行结果如下：

```
parent write return: 28
child write return: 16
```

code > q2.txt

```
1 It's child yep.
2 It's parent oh my goodness.
3
```

q2.txt 中

Question3

题目要求不使用 wait 确保子进程先输出，最简单的操作就是在父进程输出前调用系统函数 sleep(1)，1s 足够子进程先输出。输出效果即子进程很快打印信息，而父进程等待 1s 后才打印。

Question4

经过试验，子进程中能尝试题目中 exec() 的 6 个变体，调用形式如下。其中，针对不同函数要准备不同的参数，以 e 结尾意味着可以改变运行环境，此处不改变就用 main 后的 argv 代替；其他的参数类似，args 表示”ls”,”-a”。最后的输出结果类似，如下：

```
else if(pid == 0){
    //execl("/bin/ls","ls",NULL);
    //execle("/bin/ls","ls",NULL,argv);
    //execlp("/bin/ls","ls",NULL);
    //execv("/bin/ls",args,NULL);
    execve("/bin/ls",args,NULL);
    //execvp("/bin/ls",args,NULL);
    //execvP("/bin/ls",args,NULL,argv);
    //printf("error\n");
}
```

```
Hello World
.  q2.txt    question1.c  question2.c  question3.c  question4.c  question5.c  question6.c  question7.c  question8.c
.. question1  question2    question3    question4    question5    question6    question7    question8
parent is done
```

Question5

根据课上学到的知识，wait 会返回等待的 pid。这里运行两次，第一次是父进程 wait()，检查返回值发现和 fork 的 pid 一样；第二次子进程 wait()，会发现返回-1。结果如下：

```
It's child
In parent,pid:12220 - w_return: 12220
```

(a) 父 wait

```
It's parent
In child, pid = 12416,w_return = -1
```

(b) 子 wait

Question6

换成 waitpid 后，waitpid 参数更多，能执行更多操作，如非阻塞的 wait，还可以根据 pid 等待指定进程。waitpid 调用如下：”w = waitpid(pid,NULL,0)”。执行结果和 Question5 区别不大，子进程 waitpid 返回-1，父进程返回子进程 pid。

Question7

关闭标准输出后，子进程将不能打印信息，而父进程不受影响。关闭标准输出：”close(STDOUT_FILENO)”，之后终端上只会打印”It’s parent”。

```
else if(pid == 0){
    close(STDOUT_FILENO);
    printf("It's child\n");
    exit(0);
}
else{
    int w = waitpid(pid,NULL,0);
    printf("It's parent\n");
}
```

Question8

这题类似之前的 Lab1，但较为简单，最主要的工作在子进程中再一次 fork，然后进行 write 和 read 操作。只需要注意要关闭不用的管道 (write 时的 fd[0]，read 时的 fd[1])。

```
}else if(ppidd == 0){
    close(fd[0]);
    char *buf = "out something\n";
    write(fd[1],buf,strlen(buf) * sizeof(char));
    printf("pid: %d,is sending message!\n",getpid());
}else{
    int w = wait(NULL);
    char buf[20];
    close(fd[1]);
    int ret = read(fd[0],buf,sizeof(char)*20);
    printf("pid: %d,is receiving message.return: %d,string: %s\n",getpid(),ret,buf);
}
```