OS 2017 Spring Project 1 - Scheduling

資工二 B04902112 張凱捷
資工二 B04902034 賴達
資工二 B04902030 陳泓為
資工二 B04902060 周良冠

System Call

- **自訂一個system call，使一般程式可以將訊息寫入dmesg中**
  - ❑ **kernel/myservice.c**

```c
#include <linux/linkage.h>
#include <linux/kernel.h>
asmlinkage int sys_myservice(char* arg1)
{
    printk(arg1);
    return 0;
}
```

  - ❑ **arch/x86/entry/syscalls/syscall_64.tbl**

```
341 332 common  myservice       sys_myservice
```

Process

- start time : scheduler 在 fork 小孩之前使用 clock_gettime 獲得系統時間
- end time : 小孩死掉之前 clock_gettime
- dmesg : 死掉之前將字串扔到 syscall 332
  - ❑ 有嘗試做過在程式 do_exit() 的時候，或取精確的程式開始時間和結束時間，但是這樣變成所有的程式都會印出死亡訊息，所以做後決定不採用此作法。

Scheduler

❑ **FIFO - first in first out**

- **Design**
  - ❑ Sort the process by ready time and if ready time is equal ,sort by input sequence.
  - ❑ The schdedular select the process by the sorted process queue.
  - ❑ The schedular runs on CPU 0, and processes run on CPU 1.
  - ❑ As soon as process is ready, the schedular forks the process and “sched_setchedular(SCHED_FIFO)” to set its priority to do FIFO scheduling.
- **Result**

FIFO_1.txt

input:	**output:**
5 P1 0 500 P2 0 500 P3 0 500 P4 0 500 P5 0 500	P1 4208 P2 4209 P3 4210 P4 4211 P5 4212

dmesg:

```
[ 1046.082058] [project1] 4208 1493120137.280483793 1493120138.175177443
[ 1046.976132] [project1] 4209 1493120137.280565708 1493120139.069314529
[ 1047.869402] [project1] 4210 1493120137.280635167 1493120139.962650856
[ 1048.762844] [project1] 4211 1493120137.280698614 1493120140.856155222
[ 1049.656041] [project1] 4212 1493120137.280756321 1493120141.749416251
```

FIFO_2.txt

input:	output:
4 P1 0 80000 P2 100 5000 P3 200 1000 P4 300 1000	P1 4230 P2 4231 P3 4232 P4 4233

dmesg:

```
[ 1234.419151] [project1] 4230 1493120181.985825508 1493120326.525695871
[ 1243.471363] [project1] 4231 1493120182.168936461 1493120335.578549589
[ 1245.268727] [project1] 4232 1493120182.351661537 1493120337.376042614
[ 1247.064885] [project1] 4233 1493120182.534364255 1493120339.172328981
```

FIFO_3.txt

input:	output:
7 P1 0 8000 P2 200 5000 P3 300 3000 P4 400 1000 P5 500 1000 P6 500 1000 P7 600 4000	P1 4396 P2 4397 P3 4398 P4 4399 P5 4400 P6 4401 P7 4402

dmesg:

```
[ 1341.256864] [project1] 4396 1493120419.046098515 1493120433.371020266
[ 1350.223755] [project1] 4397 1493120419.411456768 1493120442.338550207
[ 1355.599644] [project1] 4398 1493120419.594125878 1493120447.714824732
[ 1357.390982] [project1] 4399 1493120419.776911455 1493120449.506288622
[ 1359.191755] [project1] 4400 1493120419.960643082 1493120451.307190560
[ 1360.997688] [project1] 4401 1493120419.960740917 1493120453.113245364
[ 1368.256765] [project1] 4402 1493120420.143561721 1493120460.372847118
```

❑ SJF - shortest job first

- **Design**
  - ❑ Sort the process by ready time and if ready time is equal ,sort by input sequence.
  - ❑ The schedular runs on CPU 0, and processes run on CPU 1.
  - ❑ First,simulate the processes execute sequence,and give each process priority by sequence.
  - ❑ Second,the schedular forks processes by ready time and set priority by first_step's given priority.
- **Result**

SJF_1.txt

input:	output:
4 P1 0 7000 P2 0 2000 P3 100 1000 P4 200 4000	P2 5297 P3 5298 P4 5299 P1 5308

dmesg:

```
[ 3364.903878] [project1] 5642 1493122453.587460853 1493122457.171329722
[ 3366.689843] [project1] 5643 1493122453.770450121 1493122458.957418317
[ 3373.852294] [project1] 5644 1493122453.953428118 1493122466.120381225
[ 3386.359090] [project1] 5641 1493122453.587372563 1493122478.628068258
```

SJF_2.txt

input:	output:
5 P1 200 7000 P2 100 100 P3 200 4000	P2 5702 P4 5703 P1 5704 P3 5705

P4 100 4000 P5 200 200	P5 5706

dmesg:

```
[ 3533.909783] [project1] 5702 1493122626.006386298 1493122626.189278969
[ 3534.268274] [project1] 5706 1493122626.189644662 1493122626.547796077
[ 3541.416015] [project1] 5703 1493122626.006488646 1493122633.696043026
[ 3548.581522] [project1] 5705 1493122626.189591426 1493122640.862062255
[ 3561.124059] [project1] 5704 1493122626.189518626 1493122653.405491246
```

SJF_3.txt

input:	**output:**
8 P1 100 3000 P2 100 5000 P3 100 7000 P4 200 10 P5 200 10 P6 300 4000 P7 400 4000 P8 500 9000	P1 5929 P2 5930 P3 5931 P4 5932 P5 5933 P6 5934 P7 5935 P8 5936

dmesg:

```
[ 3991.782429] [project1] 5929 1493123078.722095800 1493123084.094551682
[ 3991.800858] [project1] 5932 1493123078.905069958 1493123084.112983940
[ 3991.819116] [project1] 5933 1493123078.905161364 1493123084.131246365
[ 3999.013822] [project1] 5934 1493123079.088127395 1493123091.326458533
[ 4006.183190] [project1] 5935 1493123079.271064984 1493123098.496343303
[ 4015.170962] [project1] 5930 1493123078.722198075 1493123107.484754289
[ 4027.679677] [project1] 5931 1493123078.722244810 1493123119.994360852
[ 4043.729012] [project1] 5936 1493123079.453975570 1493123136.044836234
```

❑ PSJF - preemptive shortest job first

- **Design**
  - ❑ We sort the job by start time and execute time to make operation easier.
  - ❑ We have a loop to record the time unit it has run.
  - ❑ Whenever run time match the start time of a process, we run the process with the smallest execute time that is waiting to be executed.
  - ❑ Besides ,we give the process with the second lowest execute time the second high priority .This operation is aim to prevent the blank between a process's death and the next process running. Since without the second high priority, once the process with highest priority , the default CPU scheduler will find run on other process, which we are not pleasant to see, so we set the second high priority to make sure it run on the correct process.
  - ❑ Then once the scheduler detect the highest priority process's death, the scheduler will set the priority of jobs to the most suitable schedule.
- **Result**

PSJF_1.txt

input:	**output:**
4 P1 0 10000 P2 1000 7000 P3 2000 5000 P4 3000 3000	P1 6310 P2 6311 P3 6313 P4 6314

dmesg:

```
[ 4758.388986] [project1] 6314 1493123845.266204795 1493123850.755747556
[ 4765.689115] [project1] 6313 1493123843.404839492 1493123858.056393062
[ 4776.626293] [project1] 6311 1493123841.518198750 1493123868.994352510
[ 4793.052338] [project1] 6310 1493123839.660631537 1493123885.421568953
```

PSJF_2.txt

input:	**output:**
5 P1 0 3000 P2 1000 1000 P3 2000 4000 P4 5000 2000 P5 7000 1000	P1 6385 P2 6386 P3 6387 P4 6389 P5 6390

dmesg:

```
[ 4970.775066] [project1] 6386 1493124061.325554088 1493124063.156959359
[ 4974.435802] [project1] 6385 1493124059.493387511 1493124066.817954006
[ 4980.010523] [project1] 6389 1493124068.710891598 1493124072.393075459
[ 4981.858881] [project1] 6390 1493124072.415747694 1493124074.241566748
[ 4987.274413] [project1] 6387 1493124063.160983221 1493124079.657481688
```

PSJF_3.txt

input:	**output:**
4 P1 0 2000 P2 500 500 P3 1000 500 P4 1500 500	P1 6552 P2 6553 P3 6554 P4 6555

dmesg:

```
[ 5382.044619] [project1] 6553 1493124473.541587369 1493124474.455826309
[ 5382.960856] [project1] 6554 1493124474.458647224 1493124475.372126277
[ 5383.877388] [project1] 6555 1493124475.374142122 1493124476.288726399
[ 5386.610804] [project1] 6552 1493124472.623293751 1493124479.022331743
```

❑ RR - round robin

- **Design**
  - ❑ We sort the job by it start time to let it be easier to operate.
  - ❑ We have a loop to record how long the process has run.
  - ❑ If the job's start time match the process running time, put it into the job queue.
  - ❑ In the job queue, we set the head of queue to max priority , and the tail of queue to min priority.
  - ❑ If the head of job queue has run for 500 time unit or reach its execute time, pop it ,and decide to put it back to queue or not base on its execute time.

- **Result**

RR_1.txt

input:	**output:**
5 P1 0 500 P2 0 500 P3 0 500 P4 0 500 P5 0 500	P1 6614 P2 6615 P3 6616 P4 6617 P5 6618

dmesg:

```
[ 5617.965910] [project1] 6614 1493124709.471151954 1493124710.393930586
[ 5618.880018] [project1] 6615 1493124709.471218396 1493124711.308100657
[ 5619.795829] [project1] 6616 1493124709.471264275 1493124712.223978284
[ 5620.710678] [project1] 6617 1493124709.471304520 1493124713.138892702
[ 5621.626715] [project1] 6618 1493124709.471350635 1493124714.054992520
```

RR_2.txt

input:	**output:**
2 P1 600 4000 P2 800 5000	P1 6683 P2 6684

dmesg:

```
[ 5947.385420] [project1] 6683 1493125026.124127310 1493125039.836914271
[ 5950.136923] [project1] 6684 1493125026.488918909 1493125042.588614554
```

RR_3.txt

input:	**output:**
6 P1 1200 5000 P2 2400 4000 P3 3600 3000 P4 4800 7000 P5 5200 6000 P6 5800 5000	P1 6750 P2 6757 P3 6764 P4 6765 P5 6766 P6 6767

dmesg:

```
[ 6074.417689] [project1] 6764 1493125139.993998961 1493125166.878224480
[ 6076.335538] [project1] 6750 1493125135.590679154 1493125168.796221419
[ 6078.254170] [project1] 6757 1493125137.790899069 1493125170.714987363
[ 6093.461086] [project1] 6767 1493125144.036211957 1493125185.922991630
[ 6097.119802] [project1] 6766 1493125142.933663351 1493125189.581969091
[ 6098.877634] [project1] 6765 1493125142.199898059 1493125191.339925845
```

Comparison

FIFO: Our results is quite similar with the theoretical results. Fifo is most implemented by system call itself, and that shows SCHED_FIFO scheduler is well-crafted.

RR: The results fit well with the theoretical one. We only use a simple queue to implement it and the only thing has to decide is the process should push back to the queue again so the overhead is small.

SJF: The results fit theoretical one as well. Except for assign the priority before forking process, we also manage the scheduler and processes run on different CPUs.

PSJF: In order to fit the theoratical results, we set two priority. One is for the running process and the other is to prevent the blank exists. It turns out that our results fit well again.

Contribution

- kernel / process / main / analyze & test - B04902112 張凱捷
- FIFO / SJF - B04902034 賴達
- PFJS / RR - B04902030 陳泓為
- Report - B04902060 周良冠