CS344: OPERATING SYSTEMS LAB
Assignment 2b
Group 13

Adarsh Kumar, 190101002 **Upender Dahiya, 190101096**
Tanishq Katare, 190101093 **Keshav Chourasiya, 190101045**

Task 1

- Which process does the policy select for running?
  By default, xv6 uses Round Robin Scheduling Policy. There are states defined for every process, and all the process with 'RUNNABLE' states are in process table and gets a defined time quantum for the execution
- What happens when a process returns from I/O?
  After returning from I/O the process enters the RUNNABLE state and is added to the ready queue and waits for the CPU allocation to it.
- What happens when a new process is created?
  When a new process is created its state becomes 'RUNNABLE', then it's added to the ready queue, waiting for the CPU resources when CPU becomes idle it changes the process state changes to 'RUNNING' and it gets the resource it was waiting for.
- When/how often does the scheduling take place?
  Time quantum is low so the scheduling takes place many times and many context switching happens

Files changed for implementing different scheduling policies :

a) **proc.h :-** Added two features to *proc* structure, *priority* - that will denote the priority in case of SML and DML policies and *tick counter* - this counts the ticks used in preempting the process out of the running state.

b) **proc.c :-**
   i) Under *allocproc()* function when the process is in the process table and is in UNUSED state then we initialize the *ctime*, creation time as *'ticks'* and also in case of SML policy set the priority to value 2.
   ii) Under the wait2*()* function that was implemented in the last assignment the priority value of the ZOMBIE child process is set to 0 denoting that the process execution is completed.
   iii) Added a function called *findreadyproccess()* that finds the next RUNNABLE process which is used in policies SML and DML.
   iv) CPU calls the *scheduler()* to decide what process should be allotted to the CPU, this scheduler never stops and runs continuously. It is this function under which we will implement all the policies according to the given conditions.
   v) As said, the implemented *set_prio()* function system call that would change the priority queue of the caller process.
   vi) Added *increaseTicks()* function that would increase the '*tick counter'* value, this function is useful when the process is preemptive since we need to keep track of the clock ticks.
   vii) Added *decreaseInPriority()* that will decrease the priority of any process, but not below 1.

c) **param.h :-** Just set the QUANTA value to 5.

d) **trap.c :-** System calls, program faults, or hardware interrupts cause the CPU to go into "trap" state. This is implemented in this c file. On timer interrupt, a process "yields" CPU to scheduler – Ensures a process does not run for too long. So therefore when *tickcounter* reaches *QUANTA* then for DML policy priority is decreased and then *yield()* is called and only yield is called for any other policy.

e) **exec.c :-** It's given in the dynamic multilevel policy that we should call the exec system call to reset the process priority to 2 (default priority) so that is what is done in this file.

f) **defs.h :-** All the newly added functions are defined in this file.

Task 2

Files changed for adding the system call yield and set_prio:

a) **user.h**: It contains the system call definitions of xv6. Have to add function declaration here for the new system calls.
b) **usys.S**: It contains a list of system calls exported by the kernel. In this a global macro is defined called SYSCALL that inturn runs a small assembly code, which moves the mapped integer value to eax register and invokes an interrupt.
c) **syscall.h**: It contains a mapping from system call name to system call number.
d) **syscall.c**: It contains helper functions to connect the correct system call number to the actual system call implementations.
e) **sysproc.c**: It contains the actual implementations of process related system calls.

Task 3

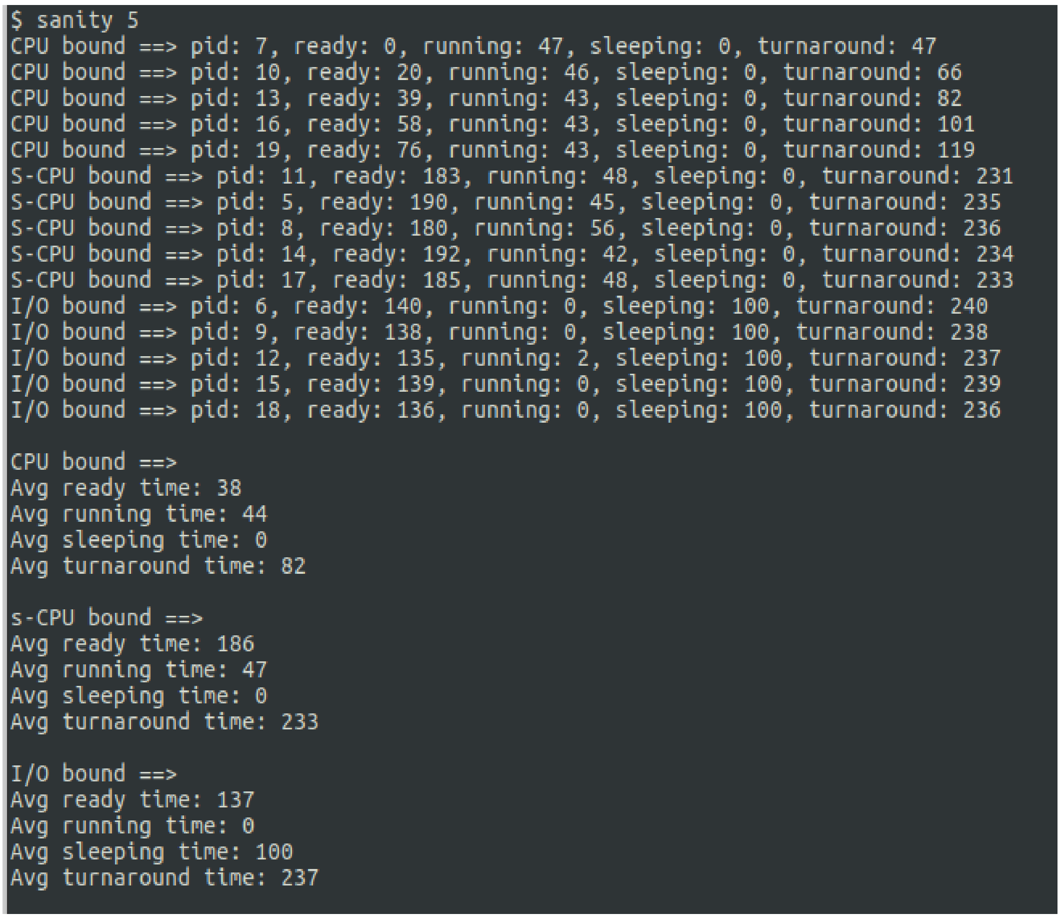
Files added to check running, ready and sleep times :

a) **sanity.c :-** As said added this c file that helps in getting the statistics of the given policies by creating 3*n child processes using fork(), here n is the argument provided to the sanity.c file
b) **SMLsanity.c :-** This file is for checking statistics of the priority policy SML

Things added in Makefile

1. Added the SCHEDFLAG ifdef for selecting the scheduling policy
2. Added the argument SCHEDFLAG for compiling under CFLAGS
3. Added _sanity and _SMLsanity under UPROGS to add new programs the system image
4. Since sanity.c and SMLsanity.c are external files hence have to add to EXTRA

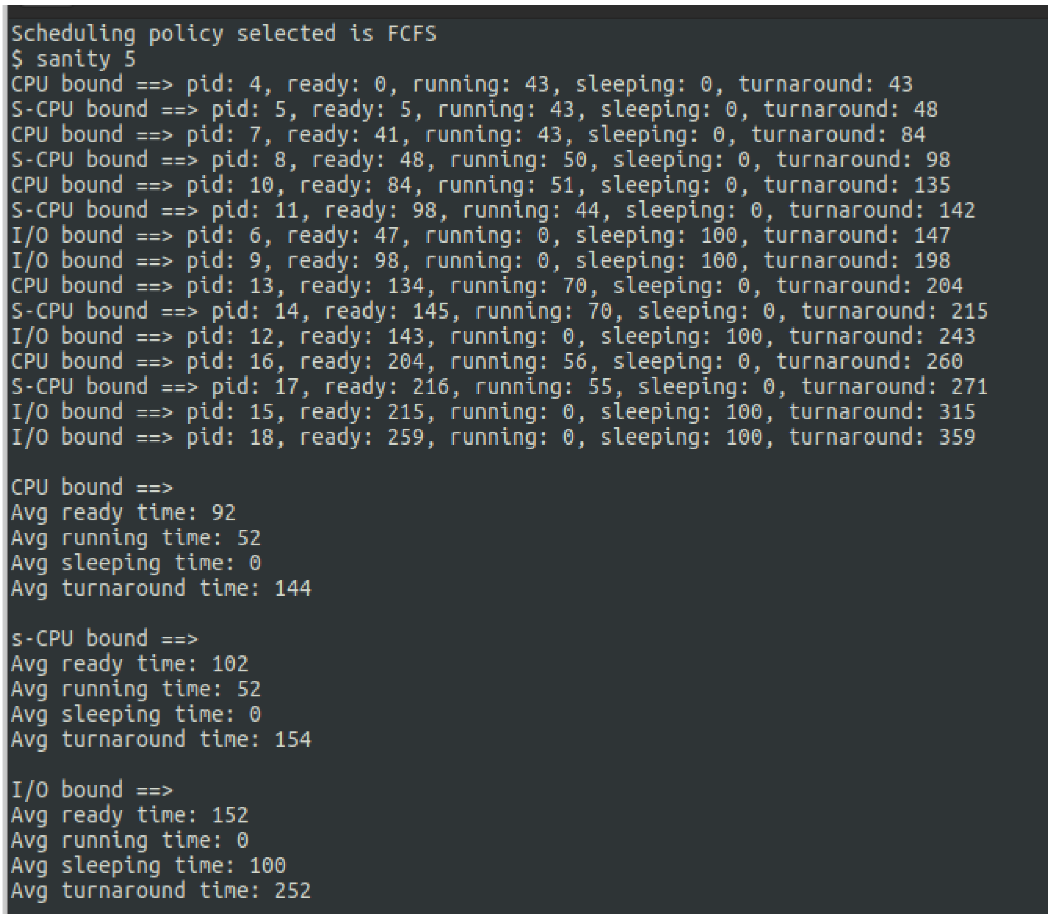
Outputs for all schedule policies:
Note xv6 has 2 CPUs so simultaneously 2 process can run in these individual CPUs
DEFAULT(ROUND ROBIN) :



```
$ sanity 5
CPU bound ==> pid: 7, ready: 0, running: 47, sleeping: 0, turnaround: 47
CPU bound ==> pid: 10, ready: 20, running: 46, sleeping: 0, turnaround: 66
CPU bound ==> pid: 13, ready: 39, running: 43, sleeping: 0, turnaround: 82
CPU bound ==> pid: 16, ready: 58, running: 43, sleeping: 0, turnaround: 101
CPU bound ==> pid: 19, ready: 76, running: 43, sleeping: 0, turnaround: 119
S-CPU bound ==> pid: 11, ready: 183, running: 48, sleeping: 0, turnaround: 231
S-CPU bound ==> pid: 5, ready: 190, running: 45, sleeping: 0, turnaround: 235
S-CPU bound ==> pid: 8, ready: 180, running: 56, sleeping: 0, turnaround: 236
S-CPU bound ==> pid: 14, ready: 192, running: 42, sleeping: 0, turnaround: 234
S-CPU bound ==> pid: 17, ready: 185, running: 48, sleeping: 0, turnaround: 233
I/O bound ==> pid: 6, ready: 140, running: 0, sleeping: 100, turnaround: 240
I/O bound ==> pid: 9, ready: 138, running: 0, sleeping: 100, turnaround: 238
I/O bound ==> pid: 12, ready: 135, running: 2, sleeping: 100, turnaround: 237
I/O bound ==> pid: 15, ready: 139, running: 0, sleeping: 100, turnaround: 239
I/O bound ==> pid: 18, ready: 136, running: 0, sleeping: 100, turnaround: 236

CPU bound ==>
Avg ready time: 38
Avg running time: 44
Avg sleeping time: 0
Avg turnaround time: 82

s-CPU bound ==>
Avg ready time: 186
Avg running time: 47
Avg sleeping time: 0
Avg turnaround time: 233

I/O bound ==>
Avg ready time: 137
Avg running time: 0
Avg sleeping time: 100
Avg turnaround time: 237
```

Process creation time can be seen the same as pid in the sense that lower the pid, the earlier the process came to the ready queue. Process with pid 5 first runs in CPU1, process with pid 6 and 9 wait as these are in sleep state waiting for I/O, process with pid 7 runs in CPU2 and process with pid 8 runs but pid 5 and 8 are S-CPU processes therefore eventually gets yielded as we called yield() system call, eventually both the CPUs would now be running with CPU bound processes with pid 7 and 10 after completion of these processes again I/O process waits and the S-CPU processes gets little amount of running time and then gets context switched out.
Since S-CPU processes are eventually yielded after a certain usage of CPU time and CPU bound processes runs hence avg turnaround time of CPU processes are a lot less that other two. Due to the waiting of S-CPU in the ready queue because of yield() they have large average turnaround time.

FCFS :

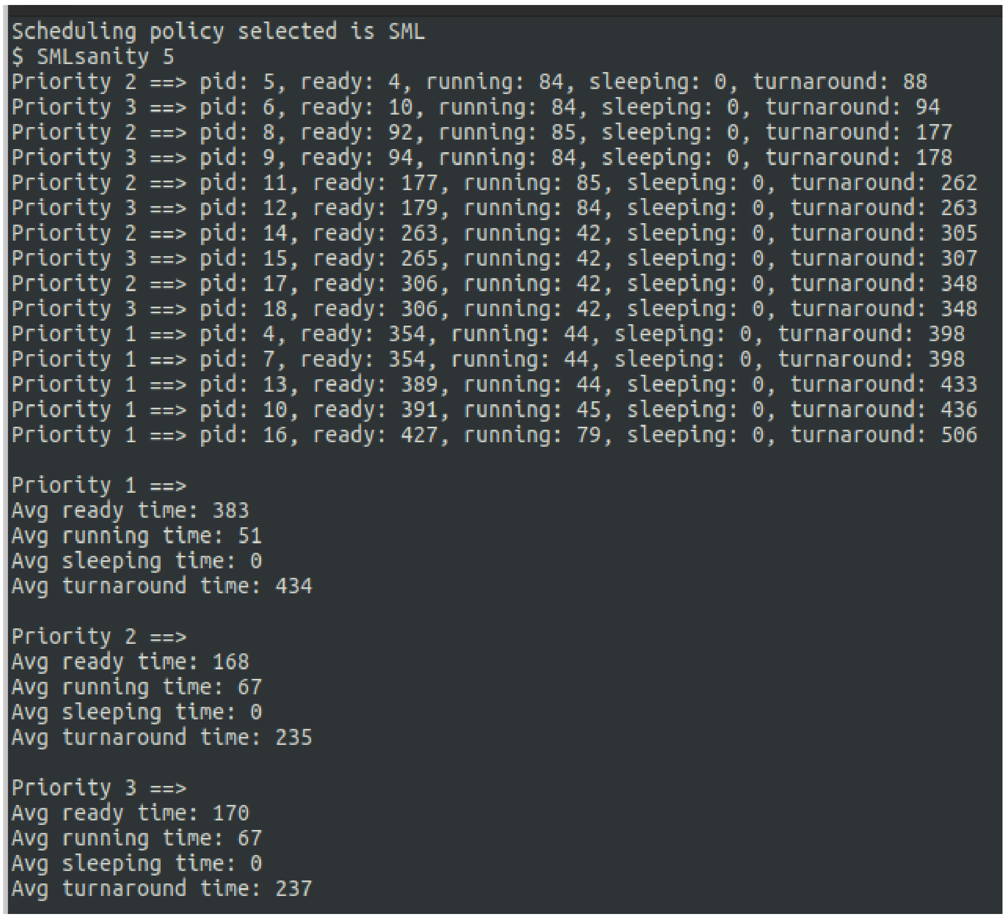


```
Scheduling policy selected is FCFS
$ sanity 5
CPU bound ==> pid: 4, ready: 0, running: 43, sleeping: 0, turnaround: 43
S-CPU bound ==> pid: 5, ready: 5, running: 43, sleeping: 0, turnaround: 48
CPU bound ==> pid: 7, ready: 41, running: 43, sleeping: 0, turnaround: 84
S-CPU bound ==> pid: 8, ready: 48, running: 50, sleeping: 0, turnaround: 98
CPU bound ==> pid: 10, ready: 84, running: 51, sleeping: 0, turnaround: 135
S-CPU bound ==> pid: 11, ready: 98, running: 44, sleeping: 0, turnaround: 142
I/O bound ==> pid: 6, ready: 47, running: 0, sleeping: 100, turnaround: 147
I/O bound ==> pid: 9, ready: 98, running: 0, sleeping: 100, turnaround: 198
CPU bound ==> pid: 13, ready: 134, running: 70, sleeping: 0, turnaround: 204
S-CPU bound ==> pid: 14, ready: 145, running: 70, sleeping: 0, turnaround: 215
I/O bound ==> pid: 12, ready: 143, running: 0, sleeping: 100, turnaround: 243
CPU bound ==> pid: 16, ready: 204, running: 56, sleeping: 0, turnaround: 260
S-CPU bound ==> pid: 17, ready: 216, running: 55, sleeping: 0, turnaround: 271
I/O bound ==> pid: 15, ready: 215, running: 0, sleeping: 100, turnaround: 315
I/O bound ==> pid: 18, ready: 259, running: 0, sleeping: 100, turnaround: 359

CPU bound ==>
Avg ready time: 92
Avg running time: 52
Avg sleeping time: 0
Avg turnaround time: 144

s-CPU bound ==>
Avg ready time: 102
Avg running time: 52
Avg sleeping time: 0
Avg turnaround time: 154

I/O bound ==>
Avg ready time: 152
Avg running time: 0
Avg sleeping time: 100
Avg turnaround time: 252
```

Processes with pid 4 and 5 are CPU and S-CPU processes respectively which are running on two CPUs and since FCFS is non-preemptive, these processes will get completed without being context switched. The process with pid 6 has to anyway wait as it is I/O but pid 7 and 8 also have to wait for idle CPU time which they will get after pid 4 and 5 complete.
So therefore the turnaround time of pid 4 and 5 is very similar to the ready time of pid 7 and 8. Since in FCFS, S-CPU processes are not yielded hence the avg turnaround time of CPU and S-CPU can be seen very nearly the same. Commenting on the I/O process due to high sleep time which we enforced to imitate I/O request waiting, these will have high ready time and also high turn around time.

SML :

1. **SMLsanity**

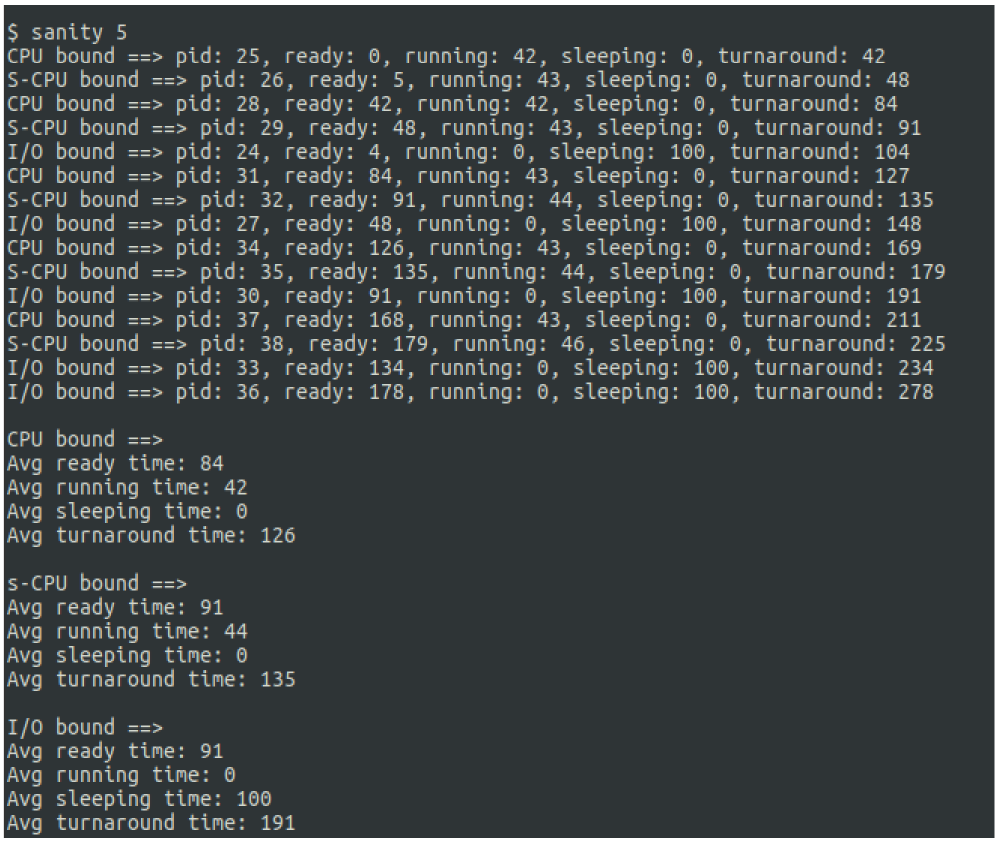


```
Scheduling policy selected is SML
$ SMLsanity 5
Priority 2 ==> pid: 5, ready: 4, running: 84, sleeping: 0, turnaround: 88
Priority 3 ==> pid: 6, ready: 10, running: 84, sleeping: 0, turnaround: 94
Priority 2 ==> pid: 8, ready: 92, running: 85, sleeping: 0, turnaround: 177
Priority 3 ==> pid: 9, ready: 94, running: 84, sleeping: 0, turnaround: 178
Priority 2 ==> pid: 11, ready: 177, running: 85, sleeping: 0, turnaround: 262
Priority 3 ==> pid: 12, ready: 179, running: 84, sleeping: 0, turnaround: 263
Priority 2 ==> pid: 14, ready: 263, running: 42, sleeping: 0, turnaround: 305
Priority 3 ==> pid: 15, ready: 265, running: 42, sleeping: 0, turnaround: 307
Priority 2 ==> pid: 17, ready: 306, running: 42, sleeping: 0, turnaround: 348
Priority 3 ==> pid: 18, ready: 306, running: 42, sleeping: 0, turnaround: 348
Priority 1 ==> pid: 4, ready: 354, running: 44, sleeping: 0, turnaround: 398
Priority 1 ==> pid: 7, ready: 354, running: 44, sleeping: 0, turnaround: 398
Priority 1 ==> pid: 13, ready: 389, running: 44, sleeping: 0, turnaround: 433
Priority 1 ==> pid: 10, ready: 391, running: 45, sleeping: 0, turnaround: 436
Priority 1 ==> pid: 16, ready: 427, running: 79, sleeping: 0, turnaround: 506

Priority 1 ==>
Avg ready time: 383
Avg running time: 51
Avg sleeping time: 0
Avg turnaround time: 434

Priority 2 ==>
Avg ready time: 168
Avg running time: 67
Avg sleeping time: 0
Avg turnaround time: 235

Priority 3 ==>
Avg ready time: 170
Avg running time: 67
Avg sleeping time: 0
Avg turnaround time: 237
```

Process with pid 4 has priority 1 (we have decided priority as (PID-4)%3, added -4 because pid values start with 4) and pid 2 has priority 2 these two will get CPU time but after pid 6 comes with priority 3 this will preempt out pid 4 and would take up the CPU, then pid 7 comes but have to wait because of low priority, then pid 8 also have to wait until the completion of pid 5 and 6.
Similarly here also the turnaround time of pid 5 and 6 is nearly same as ready time of pid 8 and 9.
Due to priority based preemption the process of priority 1 has a lot more average turnaround time as compared to priority 2 and 3, and they have nearly the same average turnaround time due to the fact xv6 has 2 CPU, so 2 concurrent processes.

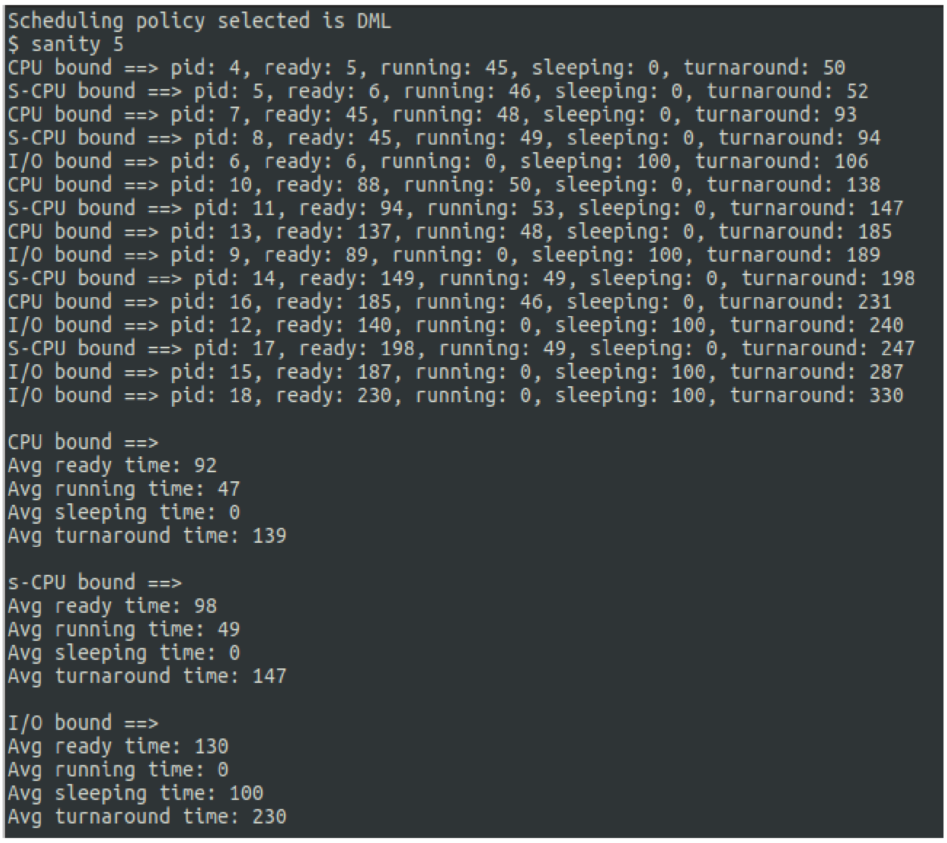
2. **sanity**

For the sanity test of SML, with a similar argument of preemption as said in above SMLsanity part, one can see nearly same average turnaround time of CPU and S-CPU bound processes, and the I/O bound processes as usual due to large sleeping time has very high average turnaround time.



```
$ sanity 5
CPU bound ==> pid: 25, ready: 0, running: 42, sleeping: 0, turnaround: 42
S-CPU bound ==> pid: 26, ready: 5, running: 43, sleeping: 0, turnaround: 48
CPU bound ==> pid: 28, ready: 42, running: 42, sleeping: 0, turnaround: 84
S-CPU bound ==> pid: 29, ready: 48, running: 43, sleeping: 0, turnaround: 91
I/O bound ==> pid: 24, ready: 4, running: 0, sleeping: 100, turnaround: 104
CPU bound ==> pid: 31, ready: 84, running: 43, sleeping: 0, turnaround: 127
S-CPU bound ==> pid: 32, ready: 91, running: 44, sleeping: 0, turnaround: 135
I/O bound ==> pid: 27, ready: 48, running: 0, sleeping: 100, turnaround: 148
CPU bound ==> pid: 34, ready: 126, running: 43, sleeping: 0, turnaround: 169
S-CPU bound ==> pid: 35, ready: 135, running: 44, sleeping: 0, turnaround: 179
I/O bound ==> pid: 30, ready: 91, running: 0, sleeping: 100, turnaround: 191
CPU bound ==> pid: 37, ready: 168, running: 43, sleeping: 0, turnaround: 211
S-CPU bound ==> pid: 38, ready: 179, running: 46, sleeping: 0, turnaround: 225
I/O bound ==> pid: 33, ready: 134, running: 0, sleeping: 100, turnaround: 234
I/O bound ==> pid: 36, ready: 178, running: 0, sleeping: 100, turnaround: 278

CPU bound ==>
Avg ready time: 84
Avg running time: 42
Avg sleeping time: 0
Avg turnaround time: 126

s-CPU bound ==>
Avg ready time: 91
Avg running time: 44
Avg sleeping time: 0
Avg turnaround time: 135

I/O bound ==>
Avg ready time: 91
Avg running time: 0
Avg sleeping time: 100
Avg turnaround time: 191
```

DML :



```
Scheduling policy selected is DML
$ sanity 5
CPU bound ==> pid: 4, ready: 5, running: 45, sleeping: 0, turnaround: 50
S-CPU bound ==> pid: 5, ready: 6, running: 46, sleeping: 0, turnaround: 52
CPU bound ==> pid: 7, ready: 45, running: 48, sleeping: 0, turnaround: 93
S-CPU bound ==> pid: 8, ready: 45, running: 49, sleeping: 0, turnaround: 94
I/O bound ==> pid: 6, ready: 6, running: 0, sleeping: 100, turnaround: 106
CPU bound ==> pid: 10, ready: 88, running: 50, sleeping: 0, turnaround: 138
S-CPU bound ==> pid: 11, ready: 94, running: 53, sleeping: 0, turnaround: 147
CPU bound ==> pid: 13, ready: 137, running: 48, sleeping: 0, turnaround: 185
I/O bound ==> pid: 9, ready: 89, running: 0, sleeping: 100, turnaround: 189
S-CPU bound ==> pid: 14, ready: 149, running: 49, sleeping: 0, turnaround: 198
CPU bound ==> pid: 16, ready: 185, running: 46, sleeping: 0, turnaround: 231
I/O bound ==> pid: 12, ready: 140, running: 0, sleeping: 100, turnaround: 240
S-CPU bound ==> pid: 17, ready: 198, running: 49, sleeping: 0, turnaround: 247
I/O bound ==> pid: 15, ready: 187, running: 0, sleeping: 100, turnaround: 287
I/O bound ==> pid: 18, ready: 230, running: 0, sleeping: 100, turnaround: 330

CPU bound ==>
Avg ready time: 92
Avg running time: 47
Avg sleeping time: 0
Avg turnaround time: 139

s-CPU bound ==>
Avg ready time: 98
Avg running time: 49
Avg sleeping time: 0
Avg turnaround time: 147

I/O bound ==>
Avg ready time: 130
Avg running time: 0
Avg sleeping time: 100
Avg turnaround time: 230
```

Process with pid 4 and 5 runs in CPU. Pid 6 has to wait as it is I/O, then pid 7 and 8 wait in ready queue for completion of pid 4 and 5. Also, here the pid 4 and 5 processes decrease their priority by 1 after every QUANTA time executed unlike the SML case, therefore giving almost equal opportunity to CPU and S-CPU processes which can be verified by seeing very close average turnaround time.
As usual, the I/O process has high ready time and also average turnaround time.