Spring 2019 COMP 3511 Homework Assignment

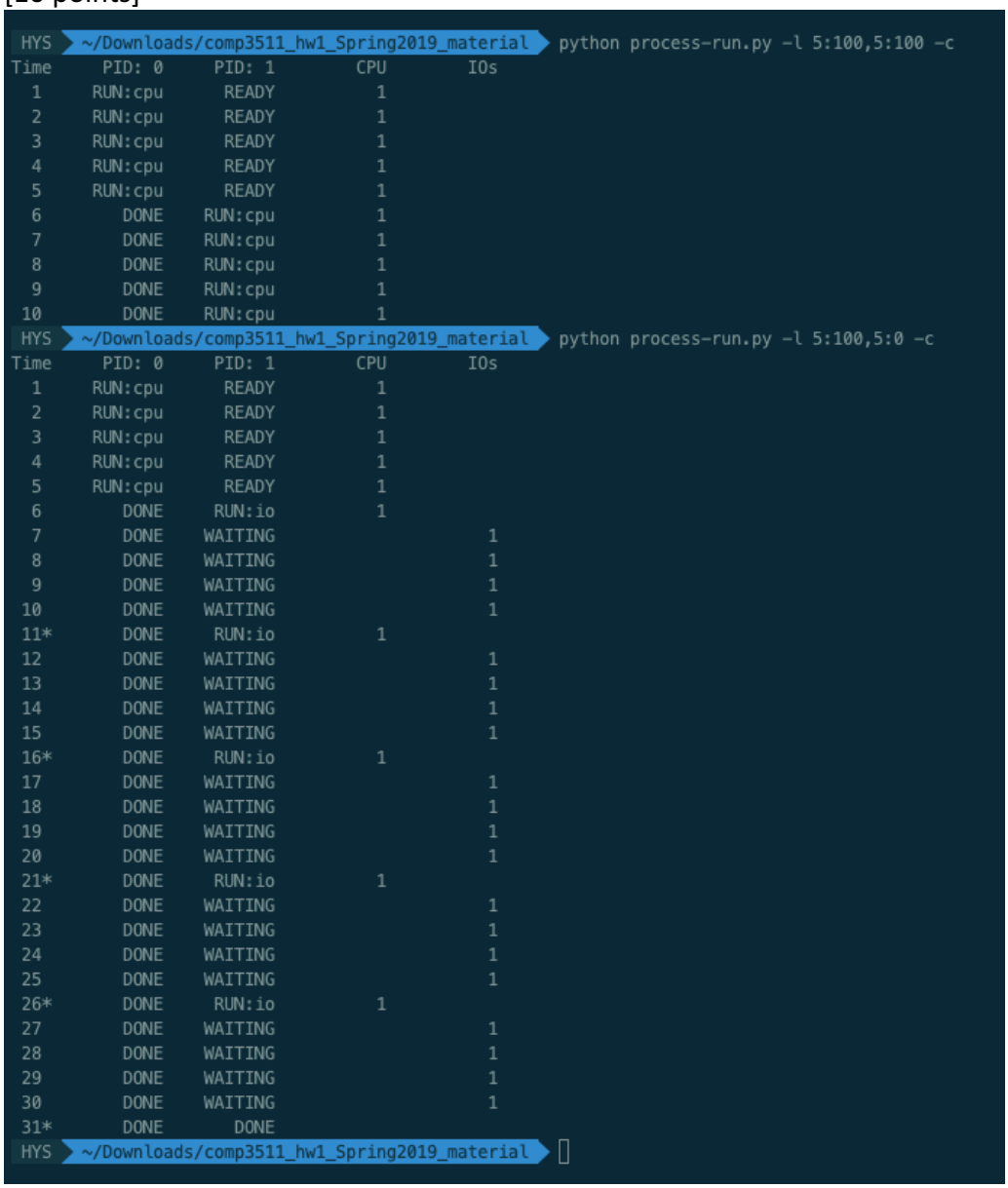
Handout Date: Feb. 16, 2019 Due Date: Mar. 4, 2019

Name: Hsieh, Yuan-Shen ID: 20306931 E-Mail: yhsiehaa@connect.ust.hk

1. [20 points] Multiple choices.

1)	A	6)	B
2)	A	7)	A
3)	A	8)	A
4)	A	9)	B
5)	A	10)	C

2. [10 points]



```
HYS ~/Downloads/comp3511_hw1_Spring2019_material python 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
HYS ~/Downloads/comp3511_hw1_Spring2019_material python process-run.py -l 5:100,5:0 -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     RUN:cpu      READY          1
  6        DONE     RUN:io          1
  7        DONE    WAITING                    1
  8        DONE    WAITING                    1
  9        DONE    WAITING                    1
 10        DONE    WAITING                    1
 11*       DONE     RUN:io          1
 12        DONE    WAITING                    1
 13        DONE    WAITING                    1
 14        DONE    WAITING                    1
 15        DONE    WAITING                    1
 16*       DONE     RUN:io          1
 17        DONE    WAITING                    1
 18        DONE    WAITING                    1
 19        DONE    WAITING                    1
 20        DONE    WAITING                    1
 21*       DONE     RUN:io          1
 22        DONE    WAITING                    1
 23        DONE    WAITING                    1
 24        DONE    WAITING                    1
 25        DONE    WAITING                    1
 26*       DONE     RUN:io          1
 27        DONE    WAITING                    1
 28        DONE    WAITING                    1
 29        DONE    WAITING                    1
 30        DONE    WAITING                    1
 31*       DONE       DONE
HYS ~/Downloads/comp3511_hw1_Spring2019_material
```

The different between the first and the second command is that the second process of the second command is issuing IO events instead of using the CPU as the first one does. According to the process state slide we discuss in class, when there isn't any

interruption or waiting event happening during the running state, the process will run continuously until terminating. Once an IO or event occurs in the middle of the running state, it will turn into the waiting state until completing the event. Thus, in the screenshot attached above, different from the five $RUN: CPU$, the five instructions of the second process of the $python\ process - run.py - l\ 5{:}\ 100{,}5{:}\ 0$ command is $RUN: IO \rightarrow WAITING \rightarrow WAITING \rightarrow WAITING \rightarrow WAITING$.

3. [10 points]

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main(){
      pid_t pid = fork();

      if (pid == 0){
            printf("hello\n");
      }

      else {
            sleep(20);
            printf("goodbye\n");
      }
}
```

4. [12 points] On fork()
   1) 6.
      The first $fork()$ creates a child and a parent process. The first $fork()$ in the second line also creates a child process for the origin child and parent. However, the second $fork()$ of the second command only create child for the original child and parent since the newly create child will move on to the other line of code. Thus, totally six "forked" is printed.

   2) 3.
      First variable c belongs to the original parent. First child, which also have a copy of variable c, is created in the first line of code. During the else block, which states the original parent process, created a new child as well.

The original parent	Variable c = 20
The first child	Variable c = 10
The second child	Variable c = 15

5. [48 points] Please answer the following questions in a few sentences.

1) The main reason for separating kernel mode and user mode in operating system is to protect the computer from being damaged by poor codes. Since the kernel mode have the highest privilege, directly using the kernel mode might cause damage that is hard to recover from.
   The way to change from user model to kernel mode is by system call APIs.
2) Temporal locality means to keep the data that is frequently accessed in a short period of time in a high speed storage in order to save time from calling it again and again, while spatial locality means to locate the data that are always called together in the nearby memory location in order to retrieve them without jumping around.

$$\begin{aligned} &Average\ access\ time \\ &\quad = (hit\ rate \times cache\ access\ time) \\ &\quad + ((1\ -\ hit\ rate) \times memory\ access\ time) \end{aligned}$$

   Base on the formula written above, for temporal locality, it lowers the $cache\ access\ time$ to save time, and for spatial locality, it rises the $hit\ rate$ to prevent missing penalty.
3) The Apple Mac OS X is a hybrid OS listed as below:

Aqua UI
Cocoa programming environment
Kernel environment (Mach microkernel, BSD Unix, I/O kit, and kernel extensions)

4) Linker turn the object module into executable module, and loader load the executable module to the main memory.
5)

fork()	CreateProcess()
No parameters are needed	No fewer than 10 parameters
Child process inheriting the address space of its parent	Requires loading a specified program into the address space of the child process during process creation

6) Sockets. The 4-tuples are source IP address source port number, destination IP address, and destination port number.
7) Orphan process is the process that still running while its parent process has already terminated. In Unix, the orphan process will be re-parented to the special init system process.
8)

Fork()	Create a new child process from the original parent process
Exec()	Replace the process's memory space with a new program
Wait()	Call in parent process to wait until the child process is terminated
Exit()	Terminate the current process