Spring 2019 COMP 3511 Homework Assignment

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

Name: HE Jianle ID: 20536134 E-Mail: jhebg@ust.hk

Please read the following instructions carefully before answering the questions:

- You should finish the homework assignment **individually**.
- There are total **5** questions.
- When you write your answers, please try to be precise and concise.
- Fill in your name, student ID, email at the top of this page.
- **<u>Homework Submission</u>:** the homework is submitted to **assignment #1** on **CASS**

1. [20 points] Multiple choices.

1) The main function of the command interpreter is
A) to get and execute the next user-specified command
B) to provide the interface between the API and application program
C) to handle the files in operating system
D) none of the mentioned

2) Embedded systems typically run on a ____ operating system.
A) real-time
B) Windows XP
C) network
D) clustered

3) Which of the following should NOT be part of a microkernel?
A) File system service
B) Inter-process communication
C) CPU scheduling
D) Address space management

4) DMA is used for:
A) High speed devices (disks and communications network)
B) Low speed devices
C) Utilizing CPU cycles
D) All of the mentioned

5) An interrupt vector
A) is an address that is indexed to an interrupt handler
B) is a unique device number that is indexed by an address
C) is a unique identity given to an interrupt
D) none of the mentioned

6) Two important design issues for the cache memory are ____.
A) speed and volatility
B) size and replacement policy
C) power consumption and reusability
D) size and access privileges

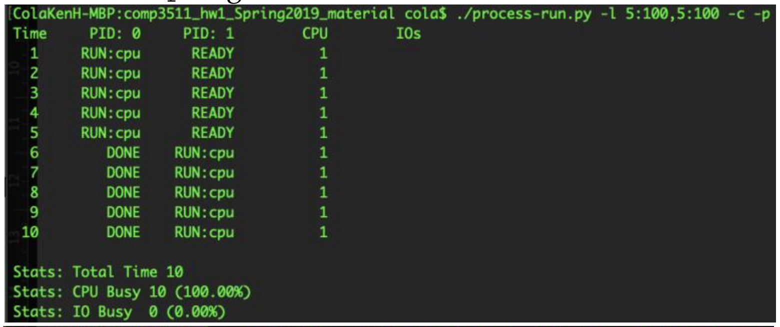
7) What is a long-term scheduler supposed to do?
A) It selects which process has to be brought into the ready queue
B) It selects which process has to be executed next and allocates CPU
C) It selects which process to remove from memory by swapping
D) None of the mentioned

8) The child process can :
A) be a duplicate of the parent process
B) never be a duplicate of the parent process
C) cannot have another program loaded into it
D) never have another program loaded into it

9) Which of the following statements is NOT true about pipes?
A) Name pipes do not require parent-child relationships.
B) An ordinary pipe can be accessed from outside the process that created the pipe.
C) Name pipes allow multiple processes to use it for communications and multiple processes can write to it.
D) Ordinary pipes allow two processes to communicate in a standard producer and consumer fashion.

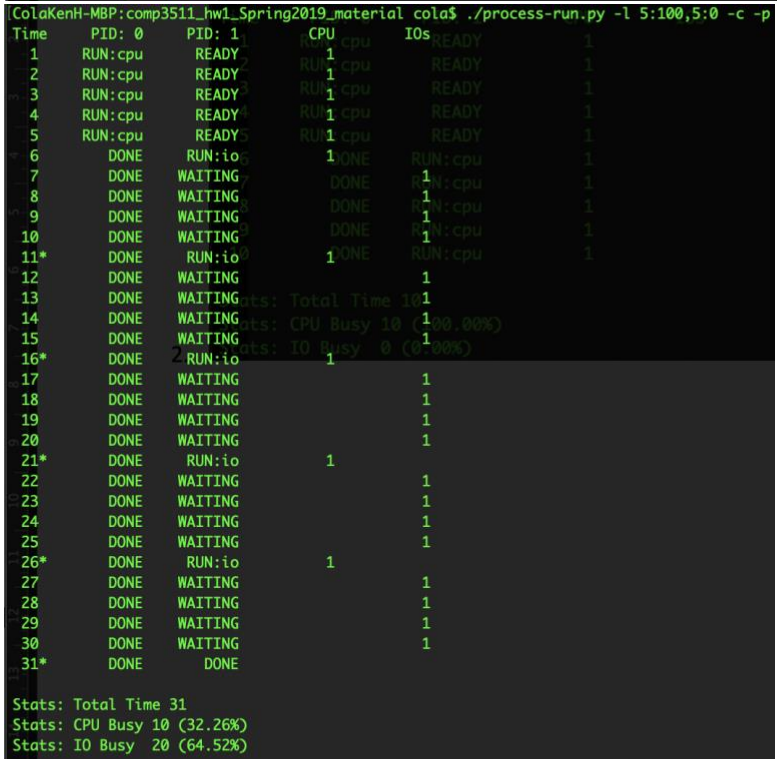
10) In indirect communication between processes P and Q :
A) there is another process R to handle and pass on messages between P and Q
B) there is another machine between the two processes to help communication
C) there is a mailbox to help communication between P and Q
D) none of the mentioned

2. [10 points] The program, `process-run.py`, allows you to see how process states change as programs run and either use the CPU (e.g., perform an add instruction) or do I/O (e.g., send a request to a disk and wait for it to complete). See the README for details. Run `python process-run.py -l 5:100,5:100` and `python process-run.py -l 5:100,5:0`. What should the CPU utilizations be (e.g., the percent of time the CPU is in use)? What are the differences between the two runs? Please justify your answer. Attach screen captures of the two runs. Hint: Use the `-c` and `-p` flags.



```
[ColaKenH-MBP:comp3511_hw1_Spring2019_material cola$ ./process-run.py -l 5:100,5:100 -c -p
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

Stats: Total Time 10
Stats: CPU Busy 10 (100.00%)
Stats: IO Busy  0 (0.00%)
```



```
[ColaKenH-MBP:comp3511_hw1_Spring2019_material cola$ ./process-run.py -l 5:100,5:0 -c -p
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

Stats: Total Time 31
Stats: CPU Busy 10 (32.26%)
Stats: IO Busy  20 (64.52%)
```

The utilizations of CPU are 100% and 32.26% respectively.

In the first run `python process-run.py -l 5:100,5:100,` no IOs are issued. The processes have been executed one by one with fully utilized the CPU.

In the second run `python process-run.py -l 5:100,5:0`, the second process is five IO instructions with 5 IO length. So that after finish running the first process, the CPU is

in idle state when the process is waiting for IO instructions.
As a result, the CPU usage in second run is much lower than the first run.

3. [10 points] Write a short program using `fork()`. The child process should print "hello"; the parent process should print "goodbye". You should try to ensure that the child process always prints first; can you do this without calling `wait()` in the parent? Attach your code. Hint: Do anything to delay the parent, e.g., looping, `sleep()`, or relinquish CPU temporarily

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

int main()
{
    pid_t pid;

    pid = fork();
    if (pid == -1)
        return -1;
    else if (pid)
    {
        sleep(1);
        printf("Parent: goodbye\n");
    }
    else
    {
        printf("Child: hello\n");
    }

    return 0;
}
```

4. [12 points] On `fork()`

1) Consider the following code segments, how many "`forked\n`" will be printed? Please elaborate. (6 points)

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main() {

   fork();
   fork() && fork();

   printf("forked\n");
   return 0;
}
```

6.

Let $P_{x,y}$ and $C_{x,y}$ be the parent and child process, while x,y representing the y-th process in x-th level.

At the beginning, there is

[$P_{0,1}$]

After running first fork(), there are

[$P_{1,1}$, $C_{1,1}$]

After running second fork(), there are

[$P_{2,1}$, $C_{2,1}$], [$P_{2,2}$, $C_{2,2}$]

After running third fork(), there are

{ [($P_{3,1}$, $C_{3,1}$), $c_{2,1}$] , [($p_{3,2}$, $C_{3,2}$) $C_{2,2}$] }

2) Consider the following code segments, how many different copies of the variable c are there? What are their values, respectively? (6 points)

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main() {
  int child = fork();
  int c = 5;
  if(child == 0)
  {
      c += 5;
  }
  else
  {
      child = fork();
      c += 10;
      if(child)
            c += 5;
  }
  printf("%d",c);
  return 0;
}
```

There are 3 copies of c.
Parent: 20
Child_1: 10
Child_2: 15

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

1) What are the main reasons for separating kernel mode and user mode in operating system? Can you name the ways that the CPU mode changes from user model to kernel mode? (6 points)

   Protect the OS itself and other system components, prevent user misusing the resources.

2) What do we mean by the temporal locality and spatial locality in a caching system? How do they affect the average access time (Hint: the average access time =hit rate x cache access time + (1 – hit rate) x memory access time) (6 points)

   Temporal locality: keep recently accessed data items closer to processor
   Spatial locality: move contiguous blocks to upper level
   They can increase the hit rate to reduce the average access time.

3) Commercial operating system usually adopts a hybrid approach in the design. Can you illustrate such an approach using Apple Mac OS X? (6 points)

   Hybrid combines multiple approaches to address performance, security, usability needs. Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment.

4) What does linker do? What does loader do? (6 points)

   Linker combine multiple relocatable object files and bring libraries into single executable binary file.
   Loader bring the binary executable from secondary storage into memory.

5) Please compare and contrast `fork()` in Unix and `CreateProcess()` in Microsoft Window operating system (6 points)

   There are mainly two difference:
   1. `fork()` passes no parameters and `createProcess()` expects no fewer than 10 parameters
   2. `fork()` create child with inheriting the address space while `createProcess()` create child with loading a specified program into address space.

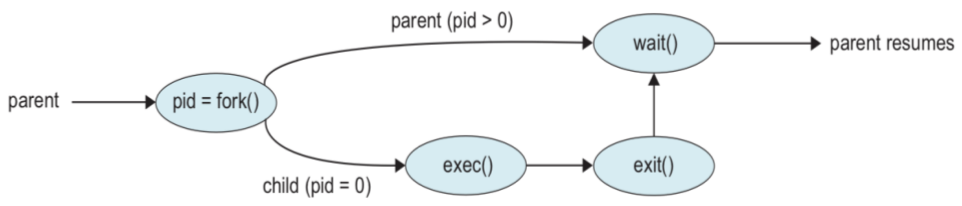
6) What are the distinctive features in a client-server communication? What are the four pieces of information (4-tuples) to determine a socket communication? (6 points)

   Client-Server Systems can be used over any kind of network, no matter local to a machine or over the Internet. The four pieces of information is client IP address, client port number, server IP address and server port number.

7) What do we refer as an orphan process? How does Unix operating system handle orphan processes? (6 points)

The processes which their parent process have been terminated. The OS will assign a new parent to them.

8) Please briefly explain how system calls are used in the following diagram including `fork()`, `exec()`, `wait()` and `exit()`. (6 points)



When `fork()`, a child process has been created. The process id will be returned to its parent and 0 will be returned to itself.
The parent process will then `wait()` for the `exit()` signal from its child.
The child process will `exec()` a program and emit a `exit()` signal when finished.
After the parent process received the signal from child, it will resume its own instructions.