Spring 2019 COMP 3511 Homework Assignment

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

Name: _Huang Daoji_ ID: _20623420_E-Mail: _dhuangal@connect.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.
- **Homework Submission:** the homework is submitted to **assignment #1** on **CASS**

1. [20 points] Multiple choices.

   1) The main function of the command interpreter is A
      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
      A) real-time
      B) Windows XP
      C) network
      D) clustered

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

   4) DMA is used for: A
      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
      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 ____. B
      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
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
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? B
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 : C
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.

For the first command, the CPU utilization is 100%, and the second, the CPU utilization is 33.3%(1 /3 * 100%, notice that if the final "done" is also considered, this would be 10 / 31 * 100%).

This process simulates the CPU schedulling with default config as running proceses one by one as the order they are input and when one process issued an io, switch to next availble process.

So in the first command's case, two processes contains 10 instructions which are all non-io instructions, so the cpu utilization should be 100%.

While in the second run, the second process contains 5 io instructions, which will cost 1 cycle to issue, and 4 cycle to wait. Notice that when the second process is running, no process is remained, so the cpu has to idle when waiting for the io. So these two processes totally makes the cpu work for 5 + 5 cycle while waiting for 4 * 5 cycles. The utilization should be 10 / 30 = 33.3%

```
yihan@ubuntu:~/Desktop/comp3511_hw1_Spring2019_material$ python process-run.py -
l 5:100,5:100
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu
  cpu

Process 1
  cpu
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

```
yihan@ubuntu:~/Desktop/comp3511_hw1_Spring2019_material$ python process-run.py -
l 5:100,5:0
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu
  cpu

Process 1
  io
  io
  io
  io
  io

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

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.

Below are two solutions to this problem. The first one is based on the hint given, while the second one can actually guarantee the child process always prints first.

```
#include <stdlib.h>
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
#include <fcntl.h>

int main(){
  if(fork()){ // parent
    sleep(10);
    printf("goodbye\n");
  }else{ // child
    printf("hello\n");
    exit(0);
  }
  return 0;
}
```

and below is the second version.

```c
#include <stdlib.h>
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
#include <fcntl.h> // O_CREAT, O_EXCL

int main(){
    // not sem_init(), which will not create a semaphore
shared by different processes
    sem_t* mutex = sem_open("mutex", O_CREAT | O_EXCL, 0644,
0);

   if(fork()){
        // if sem_wait() is also considered a wait() in this
   question, using a varible in the shared memory created by
   mmap() function instead.
        sem_wait(mutex);
        printf("goodbye\n");
    }else{
        // sleep(10); // test only
        printf("hello\n");
        sem_post(mutex);
        exit(0);
    }

    // parent free the semaphore
    sem_unlink("mutex");
    sem_close(mutex);

    return 0;
}
```

(Here, I used semaphore, which is not covered in Chap2&3. That is for when I worked on this problem, the pipe method had not been mentioned).

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 of them.

Notice that in C, e.g. a boolean expression foo && bar, the bar expression will not be executed when the foo expression(calculated first) turns out to be false. So in code "fork() && fork()", the child process of the first fork() will not execute the second fork().So in the above code, the execution timeline is something like:

--------fork--------fork---------fork-------printf

| | |--------printf

| |----------printf

|----------fork---------fork-------printf

| |---------printf

|----------printf

where the parent process of each fork remains in the same line while the child process is in the lower line.

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;
}
```

Using the same representation with 4(1), the execution timeline with the value of c is like:

---------fork--5---------fork--5----15----------print

| |-----5----15-----20-------print

|----5----10---------------------------------print

So there are 3 copies of c in the three processes, with its value being 15, 20, 10 respectively

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 mode to kernel mode? (6 points)

- **To protect the system from malicious use from users, the dual mode serves as a rudimentary form of protection. Also to protect one program form another. So that in kernel mode, certain privileged instrucions can be executed. The kernel mode also handles interrupts and interaction with the hardware. So the resources are protected and hide the complexity form user program.**
- **System calls, trap, interrupt.**

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: when an item(memory address) is accessed, it is highly possible that this specific item will be accessed/reused in the near future.**
- **Spatial locality: when an item X is accessed, it is highly possible that the items near X will be accessed/used in the near future.**
- **They decreases the average access time. For in the caching system, the item accessed now with its neighbor will be cached in a more fast device(e.g. CPU register, on-chip cache), so that when these locality works, CPU will access items right in the cache, thus increase the cache hit rate. For the cache access time is much smaller than memory access time, increase in hit rate will decrease 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)

- **Apple Mac OS X has several layers, from the UI layer, app. framework layer, core framework to kernel environment.**
- **The kernel layer consists of a Mach micro kernel, and BSD Unix parts, and other dynamically loadable modules.**
- **So in Mac OS X, the layered design, microkernel design and other approaches mix together.**

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

- **Linker combines several relocatable object modules into a single binary executable file**
- **Loader loads a executable fine into memory, which becomes eligible to run on CPU**

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

- **Fork(): takes no parameters, the child being a copy of the parent process**
- **CreateProcess(): has no less than 10 parameters, the child process should be specified, thus have a different address space from the parent process.**

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)

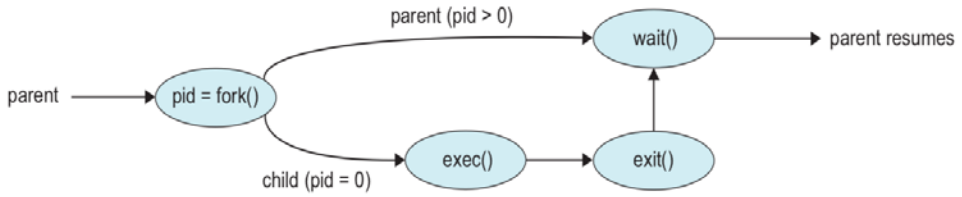
- **In a client-server communicaiton, the server may reply to multiple clients, while one client only communicates to one server.**
- **The client and server usually are on different physical machines, but can also be on the same machine.**

- **The server waiting for the client, while the client connect to the server first.**
- **4-tuples: (client ip, client port, server ip, server port)**

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

- **A running child process whose parent has terminated / finished without invoking wait()**
- **Assign the init process (or its equivelent) to be its parent, and then have it handled by the init process**

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



- **The parent process calls fork() to create a process same as itself, including global data, code, etc. And the fork() system call returns twice, with its return value being the child pid in the parent process, 0 in the child process**
- **The child process then calls exec(), which loads a specified program into memory, and execute it, the original child process is now replaced.**
- **The exit() system call terminate the loaded process, kernel will send a signal(e.g. SIGCHLD) to the parent and return the status data to the parent, deallocate all the resources of this process.**
- **The parent process will block(if it procedes faster than the child) at the wait() call until the signal received. Then the parent can get the child's return value and continue working.**