**S****p****ring 2019 COMP 3511 Homework Assignment Solution #1**

**Handout Date: Feb. 19, 2019 Due Date: Mar. 4, 2019**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_E-Mail: \_\_\_\_\_\_\_\_\_\_\_\_\_

**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.
* Please fill in your answers in the space provided, or you can type your answers in the Microsoft Word file.
* **Homework Submission:** the homework is submitted to **assignment #1** on **CASS**.

1. [20 points] Multiple choices.
2. 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

**Answer:** A

1. Embedded systems typically run on a \_\_\_\_ operating system.

A) real-time

B) Windows XP

C) network

D) clustered

**Answer**: A

1. 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

**Answer**: A

1. 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

**Answer:** A

1. 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

**Answer:** A

1. 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

**Answer:** B

1. 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

**Answer:** A

1. 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

**Answer:** A

1. 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.

**Answer**: B

1. 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

**Answer:** C

1. [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.

**Answer**:

|  |
| --- |
| $ ./process-run.py -l 5:100,5:100 -c -p && ./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: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%)  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%) |

CPU Utilization:

First run has a CPU utilization of 100% while the second run has a CPU utilization of 32.26%.

Differences:

First run only has CPU operations for the two processes, while the second run has one process with only CPU operations and another process with only I/O operations.

Second run takes significantly longer than first run.

CPU utilization of the second run is significantly lower than the first run.

First run, the processes are CPU bound, while in second run, one process is CPU bound while the other is I/O bound.

1. [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.

**Answer**:

|  |
| --- |
| for (i = 0; i < 0xEFFFFFF; ++i); // Parent, before printf  // OR  sleep(1); // Parent, before printf  // OR  pipe(fd); // Before fork  read(fd[0], &i, 1); // Parent, before printf  write(fd[1], “a”, 1); // Child, after printf  // OR any other possible ways that is consistent |

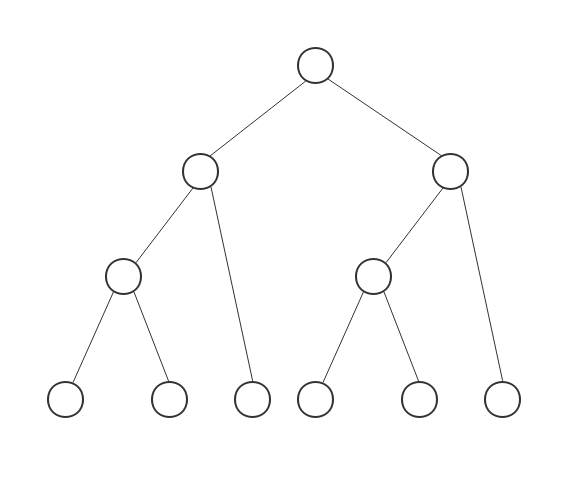
1. [12 points] On fork()
2. Consider the following code segments, how many “forked\n” will be printed? Please elaborate. (6 points)

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <unistd.h>  int main() {    fork();  fork() && fork();    printf("forked\n");  return 0;  } |

**Answer**: The result is 6

key points:

1. in the line of fork() && fork(); when the first fork() returns zero, the second fork() will not be executed.
2. refer to the figure below:



1. 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;  } |

**Answer**: The piece of code shown creates two processes. Therefore, we have a total of three processes, the parent, the first and second child. Each of these has its own private copy of the variable c. For the parent, the variable c be 20 before the end of the program. For the first child (the one created in the first program statement), the variable c will contain the value 10 before the end of the program. For the second child (the one created in the else clause), the variable c will contain the value 15 before the end of the program.

1. [48 points] Please answer the following questions in a few sentences.
2. 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)

**Answer**: The distinction between kernel mode and user mode provides a basic form of protection in that certain (privileged) instructions could be executed only when the CPU is in kernel mode. Similarly, hardware devices could be accessed only when the program is executing in kernel mode. This enforces protection of critical resources. Methods include system calls, interrupt, and trap.

1. 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)

**Answer**: temporal locality refers to that the memory location recently accessed will likely be accessed again in the near future (time), and spatial locality implies that the surrounding memory locations of the recent access will likely be visited (space). These two factors contribute to the hit rate.

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

**Answer**: Apple Mac OS X is considered a hybrid operating system; it is layered, the top is GUI (Aqua), the next layer is application environment and service including Cocoa programming environment. The bottom layer is the kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and other dynamically loadable modules. (2 points each)

1. What does linker do? What does loader do? (6 points)

**Answer**: Linker combines multiple object files into single binary executable file, while loader loads (brings) the executive file, possibly along with dynamically linked libraries or DLLs into the main memory. (3 points each)

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

**Answer**: Both are used to create a new process (2 points). Process creation system call CreateProcess() in Microsoft Window system is quite different from fork(), in which it requires loading a specific program into the address space of the child process at the time of process creation, while fork() duplicated the parent process address space (2 points). CreateProcess() expects no fewer than 10 parameters and fork() has no parameter (2 points).

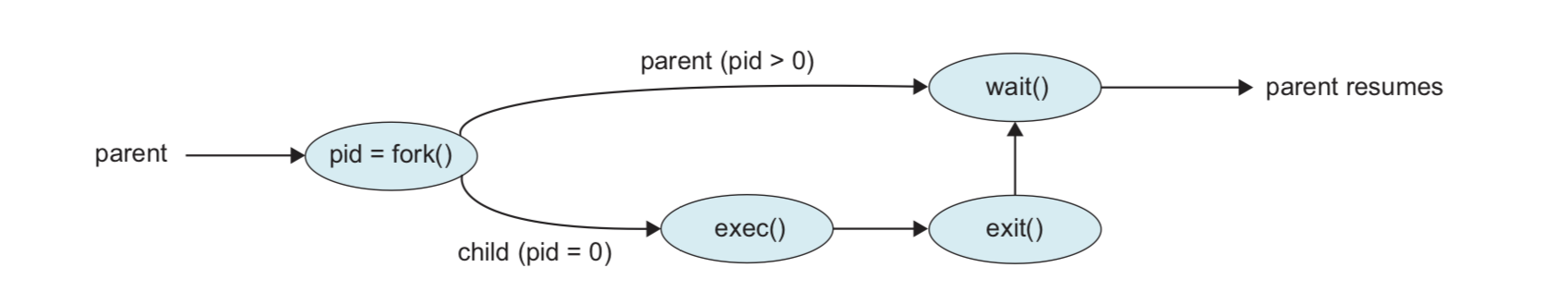
1. What are the distinctive features in a client-server communication? What are the four pieces of information (4-tuples) to determine a socket communication? (6 points)

**Answer**: In a client-server system, only clients can initiate the communications to servers. A server has a well-known address and is always running and waiting for clients to contact it for services. A socket communication is determined by source and destination IP addresses, source and destination port numbers.

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

**Answer**: If a parent did not invoke wait() and instead terminated, thereby leaving its child processes as orphans. Linux and UNIX address this scenario by assigning the init process as the new parent to orphan processes. (The init process is the root of the process hierarchy in UNIX and Linux systems.) The init process periodically invokes wait(), thereby allowing the exit status of any orphaned process to be collected and releasing the orphan’s process identifier and process-table entry.

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



**Answer**: A process uses fork() to create a child process. The parent (the original process) uses wait() to wait for child process completion of its execution. The newly created child process uses exec()to load a new program to execute, afterwards it uses exit() to inform the parent it has completed the execution, so the OS can do the clean up.