CSCE 3613 Operating Systems Homework #2 ver. 4.8

Name: Morgan Maness

ID: 010813680

34 points

7 questions, some with multiple parts

# Instructions

* Type your work, print it to a \*single\* PDF, and upload it to Blackboard before the due date and time. It is strongly suggested to use the given document.
* Show all of your work. Without proper justification and details of steps, correct answers alone may not carry full credit.
* -2 points if you do not insert your name and ID at the top of the document.
* -5 points if it is not typed.
* -5 points if it is not a PDF file.
* -5 points if it is not a single PDF file. Submit one PDF file. Do not submit zip files containing one or more files.
* -5 points if you present the worked problems out of order. In other words, please present the problems in the order assigned, 1, 2, 3, …

1. (2 pts.) Describe an I/O-bound process.

**Answer:** An I/O bound process spends more time doing I/O than computations. It typically has many short CPU bursts.

2. (5 pts.) Consider the following code segment. Assume the parent processes will not exit until the children processes complete. How many times is “Hi” printed? Sketch a tree of how the processes are created.

**Answer:** The program will print Hi 5 times

int main()

{

pid\_t pid;

printf("Hi \n");

pid = fork();

pid = fork();

if (pid == 0){

printf("Hi \n");

wait(NULL);

}

else{

printf("Hi \n");

}

wait(NULL);

}

Child

Child

Child

Parent

Parent

Parent

Parent

3. (2 pts.) List the two models of interprocess communications.

**Answer:** Shared Memory and Message Passing

4. (5 pts.) Describe a thread pool.

**Answer:** A thread pool is when a number of threads are created ahead of time and then thread share assigned to tasks as they arrive. This has several advantages. First, the overhead of creating a thread is performed ahead of time which permits the task to be serviced quicker than if waiting and creating a thread after the task arrives. In addition, it permits the number of threads for an application to be bound by the size of the thread pool providing a way to limit the resources for an application.

5. Using Amdahl's Law, calculate the speedup gain of an application that has a 95% parallel component for the following:

5.1 (2 pts.) Eight processing cores. Show your work.

**Answer: =** 5.93

5.2 (2 pts) Sixteen processing cores. Show your work.

**Answer:** = 6.1

6. The following pair of processes, Process A and Process B, share a common variable X.

/\* Process A \*/

int Y;

Y = X\*4;

X = Y;

-----------------------

/\* Process B \*/

int Z;

Z = X+3;

X = Z;

X is set to 2 before either process begins execution. As usual, statements within a process are executed sequentially, but statements in Process A may execute in any order with respect to statements in Process B.

6.1 (8 pts.) How many different values of X are possible after both processes finish executing and what are the values?

**Answer:** x=11 if A finishes before B begins

x=8 if B finishes before A finishes

x=5 if A finishes after B has started but not finished

x=20 B begins before A

Four different values of x

7. Consider the following set of processes with the length of the CPU burst size given in milliseconds and the arrival time given in milliseconds. If two processes arrive at the same time and either can go according to the particular scheduling algorithm, the lower numbered process goes first. For example, if there is a tie between P3 and P4, P3 goes first because 3 is less than 4. Other ties are broken with FCFS. \*The priority column is only relevant to the non-preemptive priority scheduling algorithm. A lower priority value corresponds to a higher priority.

Process Burst Time (ms) Arrival Time (ms) Priority\*

P1 1 0 1

P2 5 0 2

P3 3 2 2

P4 1 2 3

P5 4 3 1

7.1 (8 pts.) Calculate the average waiting time for each of the scheduling algorithms in the table below and place the answers in the table below. (2 pts. for each entry)

|  |  |
| --- | --- |
| **Scheduling algorithm** | **Average waiting time** |
| FCFS |  |
| SJF |  |
| RR (quantum=2) |  |
| Non-preemptive priority\* |  |

The End.