Operating System - EXAM I (Spring 2021)

100 Pts
Due: : Sunday March 7th 11:59 PM (dropbox)
{or email directly to me if Dropbox not working}

Your Name :

Question #1

The Process/Task Control Block (PCB) represents a process in the MIX. It contains many pieces of information associated with a specific process. (single threaded or multi-threaded). Generally, the PCB of a process will contain things like the Process state, Program counter value, CPU register values ... Along with the PCB a process also must be assigned memory for the Code, stack, heap (maybe several of some if multi-threaded). A process will be made up of both a PCB and this other memory.

How and where the PCB and other memory is implemented varies on different software and hardware.

One consideration is what address space these different things will reside in. It could be in the Kernel's memory space or the User's memory space.

a) State which memory (Kernel or User) should the PCB reside in? Then explain **WHY**.

b) State which memory (Kernel or User) should the Stack and Heap reside in? Then explain **WHY**.

Question #2

Assume we are developing a program solution where we plan to also implement it on a distributed system in the future. At one point it becomes clear that it would be useful to break the process up into 2 separate pieces. They could be run in parallel but even running concurrently would be beneficial. However, once we break it up into different parts, they will have to communicate with each other and share some data. We have a choice. a) fork a separate process. b) create a second thread in one process.

Given all the information above. Which choice should be made to make the implementation of the sharing of data easier to implement and maintain over the life of the program as it is used in our company? **Explain!**

Question #3

Consider the following code segment. How many unique **new** (do not count the starting process) processes are created?

(*you may want to supply some reasoning/diagram to allow for partial credit if applicable*)

```
pid = fork();

if (pid == 0)
   fork();

fork()

fork();
```

Question #4

Assume you have a system with many processors. You are asked to design a solution to the following problem.

*Given a "VERY" large Data set of N numbers (N will vary each time). Create a solution that will find the **averag**e of all the numbers in the set. (you can assume no overflow number issues).*

1st: Choose the best parallel approach (Data or Task)

2nd. Describe, at a high level, how it would work.

It should make the best use parallelism to get the solution.

Question #5

1)
   a. What is a Spin Lock?

   b. Why might we choose to use a form of Mutex Lock that suffers from Spin Lock even though other locking choices are available that do not have spinlock..

2) Explain why Synchronization hardware is useful for an Operating System. A) What does it provide easily that becomes complicated for a software approach? B) why are these hardware approaches usually restricted for kernel use only?

Question #6

The **Bakery** algorithm is easy to explain using an example of a real Bakery. However, trying to implement is to help with a solution to the *critical section* problem runs into one particular problem not encountered in a real Bakery.

What is it that becomes hard to replicate from the real bakery? Explain.

Question #7

To handle **Deadlocks** we can use protocols to **prevent** or **avoid** deadlocks so that no deadlocks will occur.

1. What is the difference between a deadlock-prevention and deadlock-avoidance approach.?

   You should think about what makes them different not just on the formal definitions in the text.

2. To implement Deadlock-Prevention we could remove Mutual Exclusion from the system. A) Why might that not be the best of the 4 necessary conditions to remove? B) What might be a situation where removing Mutual Exclusion would be ok?

Question #8

Peterson's solution presented in the book solves the critical section problem for 2 processes with the use of both a **Boolean Flag[2]** and an **int turn** data structure. The turn saying whose turn it is to get in the critical section and Flag telling if a process wants into the critical section. **ASSUME** process **1** is in the critical section, what are the possible valid values of both turn and flag for both processes 1 and 2?

Fill in the last column of the table for a: thru h: {**Y** if it is a valid state and **N** if it is not a valid state.}

	Turn	**Flag[1]**	**Flag[2]**	**Valid State(*Y/N*)**
a:	1	T	T	
b:	1	T	F	
c:	1	F	T	
d:	1	F	F	
e:	2	T	T	
f:	2	T	F	
g:	2	F	T	
h:	2	F	F	

Person's Solution:

```
int turn;
boolean flag[2];

do
{
   flag[i] =true;
   turn = j;
   while (flag[j] && turn == j);

      << CRITICAL SECTION>>   Process 1 is here !!!!!!!

   flag[i] = false;

      << REMANDER SECTION >>

} while (true);
```

Help: things to think about: what must be or must not be true for P1 to be in the CS. Some might be complicated, but some might be very simple.