**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.

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

The PCB should reside in kernel memory space, because it is a management structure used by the kernel to allocate and manage access to the physical hardware resources. If the user had access to this mechanism, then it would be a major security concern because they could modify the values, bypassing the kernel as the middle-man for hardware access.

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

The stack and heap should reside in user memory space, because these are the process’s workspaces – it probably needs somewhere to read from and write to if it is going to get any work done, and it needs access to this space if it is going to make use of it.

**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!**

With cloud facilities and the abstraction involved in large virtual systems these days, I’m not 100% sure about my answer here… but as far as I am aware distributed systems involve separate machines, with their own hardware, Operating systems, and memory states. Since threads share memory spaces with other threads in the same process, it would be a whole lot of overhead to maintain and synchronize memory states between how many different contributing systems – so threading is ruled out.

Each machine could have it’s own process running, communicating back and forth on their progress and assignments, however that is designed… but their memory couldn’t feasibly be shared like in a thread as far as I know – if it is used in a distributed system.

**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();**

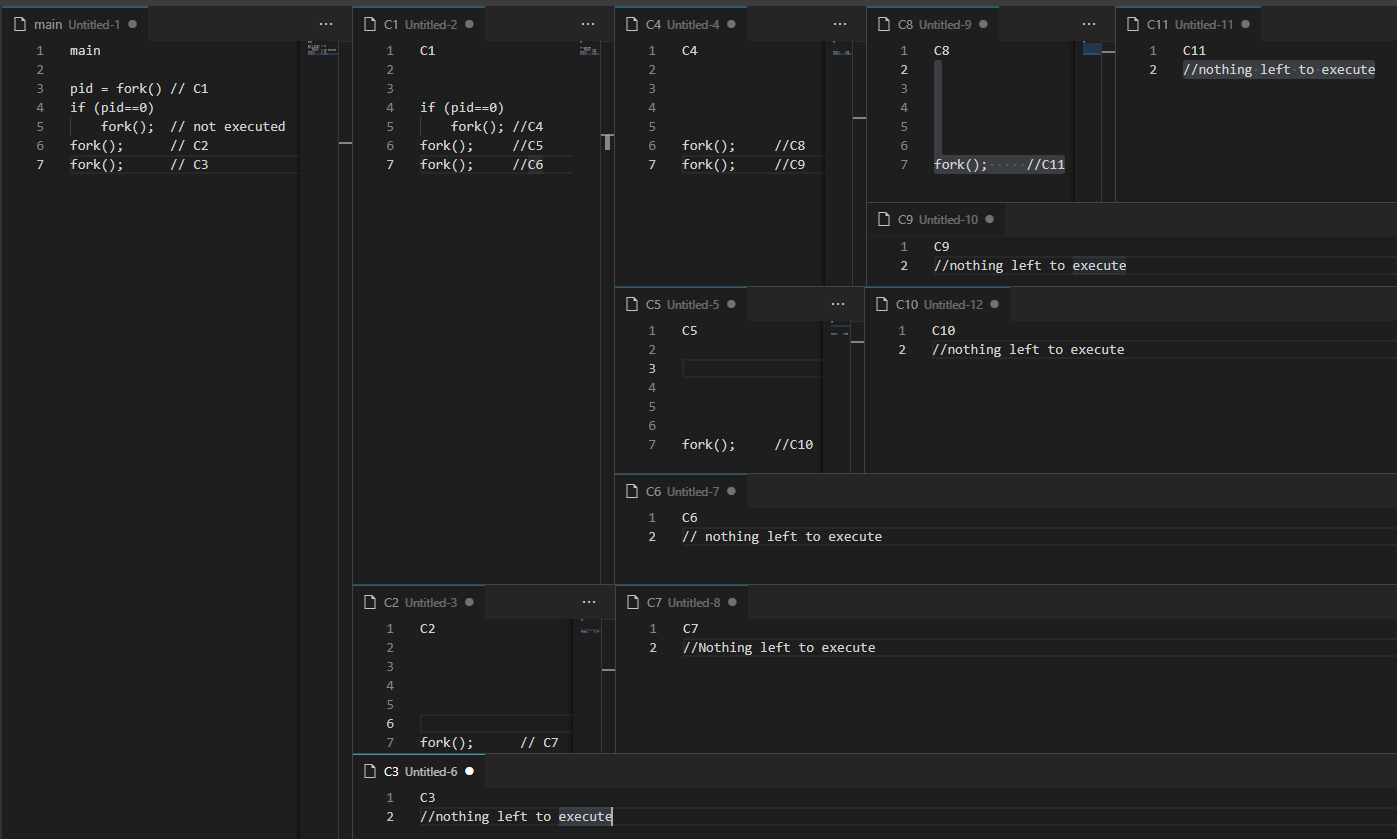
**if (pid == 0)**

**fork();**

**fork()**

**fork();**

11 processes created from this program. Screencap shows my visualization.



**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)

Data parallelism, because we only have a single task to be performed across a large data pool.

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

We would assign sections of the data list to each processor, to sum and report an average, and then calculate the average of the reported averages at the end. It could come up that the data set doesn’t breakup into even segments % our number of processors/threads, and you could assign a weighting scheme to the reporting scheme to account for and correct for that

But basically we just break up the dataset, and each thread walks along the dataset starting at a different index, reporting back on their segment.

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

**Question #5**

* 1. What is a Spin Lock?

A spin lock is a mechanism that keeps a process waiting actively on the processor instead of blocking, and getting swapped off the processor.

* 1. 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..

On a single core system spin locks can slow things down since nothing else is going on while it’s sitting there spinning, but on a multi-core system, it can be faster to sit and spin, waiting on a thread in another core to pop the lock than to deal with all of the overhead of context switching.

1. 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?
2. Hardware solutions are atomic – meaning they cannot be interrupted and interwoven, which is useful for resolving issues like ‘lost updates’ which become a concern when implementing locking mechanisms.
3. Hardware approaches are usually restricted for kernel use for two reasons, kind of. 1 is that the operating system providers have to work with the chip manufacturers to develop this functionality before they can provide an interface for it. and 2 is that it’s the kernel’s job to manage hardware, and provide users an interface to the systems and functionality provided by hardware. The kernel is the middle-man for hardware access, period.

**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.

Computers don’t know how to line up – interweaving can cause confusion on position assignments, and there needs to be logic to address when two threads try to take the same position in the queue

**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.?

Prevention involves engineering out one of the conditions for a deadlock. Each solution proposes an operational tradeoff which influences the structure of the system’s logic. Avoidance does not engineer out deadlocks. Avoidance requires the system to self-monitor, be aware of potential deadlock situations, and **avoid** those situations actively. Avoidance creates some overhead, but it doesn’t introduce programming challenges by disallowing certain behaviors altogether.

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

1. 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?

Removing the mutual exclusion condition would introduce the same ‘lost update’ risks involved with interweaving, especially if we have multiple files reading/writing to the file at the same time. If it was the case that the file was read-only, or the access requests to the file were read-only, then it might be okay to scrap mutual exclusion. If nothing is being modified then we don’t have any ‘lost updates’

**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 | **Y** |
| b: | 1 | T | F | **Y** |
| c: | 1 | F | T | **Y** |
| d: | 1 | F | F | **Y** |
| e: | 2 | T | T | **N** |
| f: | 2 | T | F | **Y** |
| g: | 2 | F | T | **N** |
| h: | 2 | F | F | **Y** |

**Getting a little confused on this one, but I’m answering on the assumption that we are identifying the possible arrangements of conditions that would let process 1 into the critical section.**

**And as I figure, 1 only gets in if:**

* **It is not 2’s turn (failing turn==j, breaking spin lock) a.k.a. turn=1**

**or**

* **Process 2’s flag is false (failing flag[j], breaking spin lock)**

**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.*