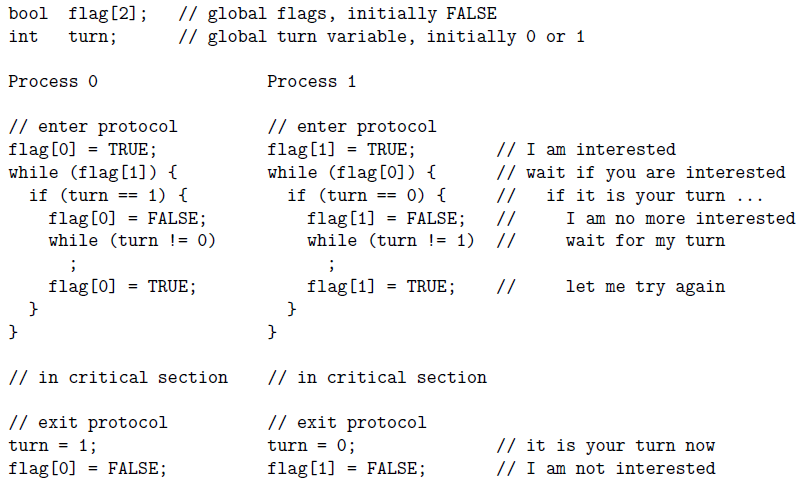
**CS 3331 Exam 1 Review**

1. **Explain interrupts and traps and provide a detailed account of the procedure that an operating system takes to handle an interrupt.**
2. **Explain what the CPU modes are and their uses. How does the CPU know what mode it is in?**
3. **What is a context? Provide a detailed description of all activities of a context switch.**
4. **Draw the state diagram of a process from its creation to termination, including all transitions. Make sure you elaborate on every state and every transition.**
5. **Enumerate the major differences between kernel-supported and user-level threads.**
6. **Define the meaning of a race condition. Answer the question first and use an execution sequence with a clear and convincing argument to illustrate your answer. You must explain step-by-step why your example causes a race condition.**
7. **Explain the progress and bounded waiting conditions and enumerate their differences.**
8. **What is an atomic instruction? What would happen if multiple CPUs/cores execute their atomic instructions?**
9. **Explain the one-to-one, many-to-one, and many-to-many thread models. Make sure you explain each model clearly.**
10. **What is thread cancellation? How many commonly used thread cancellation types are there? Name them and provide an explanation for each.**
11. **Explain the meaning of thread safe.**
12. **A good solution to the critical section problem must satisfy three conditions: mutual exclusion, progress, and bounded waiting. Explain the meaning of the progress condition. Does starvation violate the progress condition?**
13. **A computer system had two CPUs that share the same memory. All processes are stored in the shared memory but can be run on either CPU. To gain efficiency, the designers chose the following CPU scheduling policy: There is only one ready queue and is stored in the shared memory, each CPU has its own CPU scheduler, when a CPU is free, the schedule of that CPU picks up the first process in the ready queue to run on the same CPU. Do you think this policy works well? State your claim first and justify your claim step-by-step with execution sequences.**
14. **Why is handling threads cheaper than handling processes?**
15. **A good solution to the critical section problem must satisfy three conditions: mutual exclusion, progress, and bounded waiting. Both progress and bounded waiting involve some form of waiting. Explain and differentiate the waiting in progress and bounded waiting.**
16. **Define and compare the concepts of process, thread, and fiber. How are they scheduled?**
17. **What is dual-mode execution?**
18. **What is a privileged instruction and why are they needed?**
19. **What does it mean to be interrupt-driven?**
20. **What is a system call?**
21. **Why is an interval timer needed?**
22. **What is a process?**
23. **How is process space allocated?**
24. **What is a PCB? What information should be stored in a PCB?**
25. **Can a process assume the system scheduling process in order to run properly? Why?**
26. **What is a CPU scheduler?**
27. **What is a coroutine?**
28. **Jason and John are sharing an apartment, and they share the responsibility of buying milk. To avoid the possibility of both buying, they come up with the following “algorithm”: As soon as you arrive home, you leave a signed note on the fridge’s door. Only then you check, and if you find there is no milk and there is no note (other than yours), then you go and buy milk, put the milk in the fridge, and remove your note. Their pseudo-code looks like the following:**

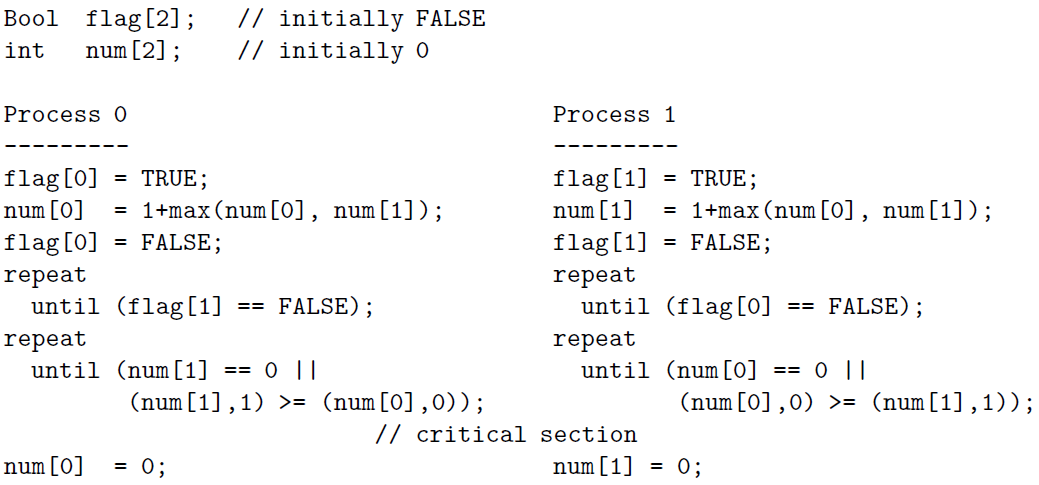
|  |  |
| --- | --- |
| **Jason** | **John** |
| **Jason leaves note**  **If (no note from John) then**  **If (no milk) then**  **Buy milk**  **End if**  **End if**  **Remove Jason’s note** | **John leaves note**  **If (not note from Jason) then**  **If (no milk) then**  **Buy milk**  **End if**  **End if**  **Remove John’s note** |

**Jason and John both suspect they may end up with no milk at all! Since they didn’t take concurrent, they aren’t certain how this can happen. Use a step-by-step execution of the above “algorithm” to show that Jason and John can end up with no milk at all. Note that Jason and John cannot see and talk to each other.**

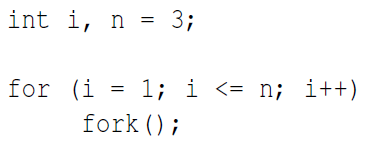
1. **Consider the following solution to the critical section problem for two processes. Show that this solution satisfies the mutual exclusion condition.**



1. **Design a C ­program segment so that the main method creates two child process using fork and each of these child processes creates two child processes, etc such that the parent-child relationship is a perfectly balanced binary tree of depth n with main() at the root. The depth n has already been stored as a valid positive integer. The main() prints its PID, and each child prints its PID and its parent’s PID.**
2. **Consider the following solution to the mutual exclusion problem for two processes. In the following holds if or . Prove rigorously that this solution satisfies the mutual exclusion condition.**



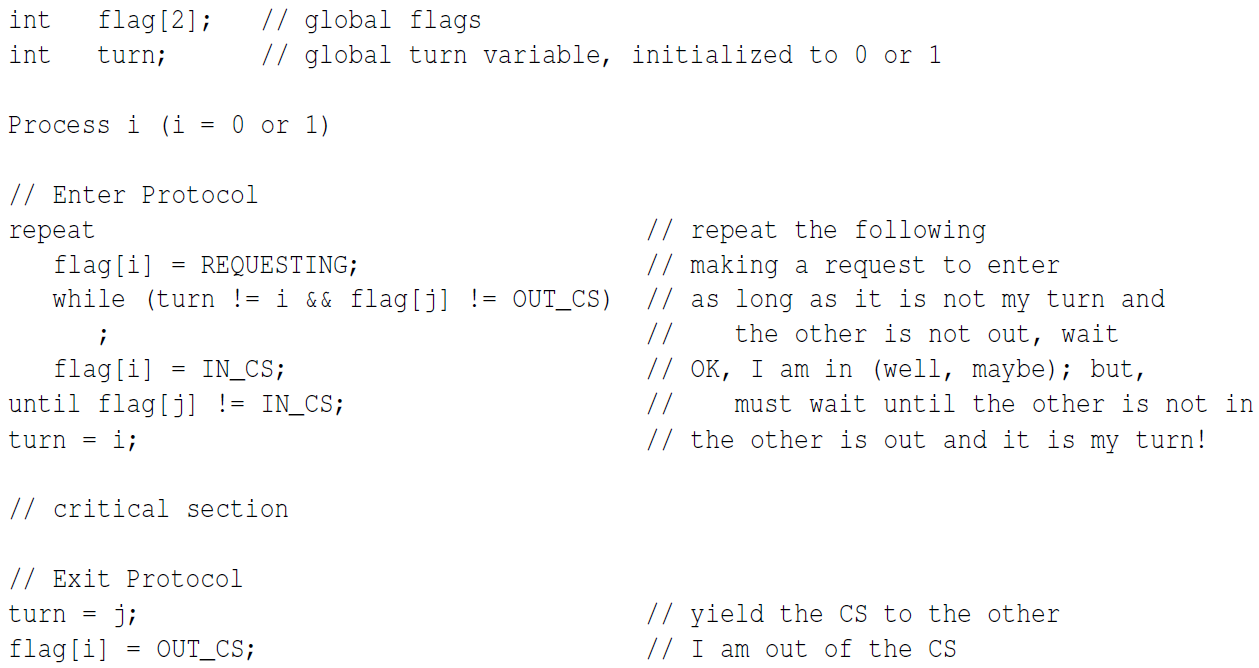
1. **Consider the following program segment. Suppose all fork calls are successful. Draw a diagram showing the parent-child relationship of all involved processes and provide an explanation of how this relationship is obtained. How many processes will be created if n is set to a positive integer k? Provide a justification for your answer.**



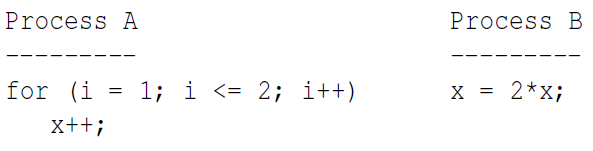
1. **Consider the following two processes to be run concurrently using shared memory for x. assume that x is initialized to 0 and x must be loaded into a register before further computations can take place. What are all possible values of x after both processes have terminated? Use a step-by-step execution sequence to show all possible results.**



1. **Consider the following solution to the mutual exclusion problem for two processes. A process can be making a request REQUESTING, executing in the critical section IN\_CS, or having nothing to do with the critical section OUT\_CS. This status information, represented by an int, is saved in flag[i] of process P1. Variable turn is initialized elsewhere to be 0 or 1. Note that flag[] and turn are global variables shared by P0 and P1. Prove rigorously that this solution satisfies the mutual exclusion condition.**



1. **Consider the following two processes to be run concurrently using shared memory for the int variable x. Assume x is initialized to 0 and x must be loaded into a register before further computations can take place. What are all possible values of x after both processes have terminated? Use clear step-by-step execution sequences to prove.**



1. **Consider the following solution to the mutual exclusion problem for two processes. The solution uses two global int variable x and y that are both initialized to 0. Prove that this solution satisfies the mutual exclusion condition.**

