CS 354 Midterm review Fall 2019

Consider a set of Xinu processes. Each of the questions below starts with this set of processes

|  |  |  |
| --- | --- | --- |
| **PID** | **Priority** | **State** |
| 1 | 20 | Current |
| 2 | 20 | Ready |
| 3 | 25 | Waiting on semaphore 10 |
| 4 | 30 | Waiting on semaphore 11 |

1. Given the current processes above and process quantum of 100 ms, how much run time would each process get in the CPU in the first 1 s of execution?

1: 500ms, 2: 500ms, 3: 0ms, 4: 0ms

1. Suppose the same instance as above, except after 400 ms, the current process calls signal(11)?

1: 200ms, 2: 200ms, 3: 0ms, 4: 600ms

1. Suppose this set of Xinu processes is running a safety-critical operation at an industrial plant and process 2 is responsible for raising an alarm if certain safety violations occur. What dangers might occur if process 1 calls signal(10)?

Process 2 could starve, meaning warnings would not appear to an operator.

1. Suppose process 3 is responsible for running code to call signal(11) and process 4 is responsible for calling signal(10). What is this phenomenon called and how can it be fixed?

Deadlock. Cognizant programming and ensuring that all mutexes are locked and freed in the same order.

1. Why does Xinu disable interrupts during system calls? What might happen if it didn’t?

This is to prevent interrupts from interrupting interrupts. This could damage the system state, since a lot of system calls handle non-thread-safe data structures or I/O devices.

1. What is a buffer pool? Why are they used?

A buffer pool is a collection of pre-allocated memory buffers primarily used by I/O devices. It’s also faster than making repeated calls to getmem().

Consider a set of Xinu processes. Each of the questions below starts with this set of processes.

|  |  |  |
| --- | --- | --- |
| **PID** | **Priority** | **State** |
| 1 | 20 | Current |
| 2 | 20 | Ready |
| 3 | 10 | Ready |

1. What would the process table look like if process 1 made a call to create() with priority 5? What would the ready list look like?

|  |  |  |
| --- | --- | --- |
| **PID** | **Priority** | **State** |
| 1 | 20 | Current |
| 2 | 20 | Ready |
| 3 | 10 | Ready |
| 4 | 5 | Suspended |

Current: 1. Ready list (starting with front): [2, 3]

1. What would the process table look like if PIDs 1, 2 and 3 all made calls to wait()? Which process would run?

|  |  |  |
| --- | --- | --- |
| **PID** | **Priority** | **State** |
| 1 | 20 | Waiting for semaphore |
| 2 | 20 | Waiting for semaphore |
| 3 | 10 | Waiting for semaphore |

The null process would run.

1. What is the difference between Xinu’s resched() call and the ctxsw.S call?

Resched() handles placement of processes in the free list and scheduling and makes a call to ctxsw(). Ctxsw() handles the overhead of actually swapping one process out for another and managing stacks.

Consider the following free list for a 32-bit system:

|  |  |
| --- | --- |
| **Address** | **Size** |
| 4050 | 300 |
| 4300 | 1020 |
| 6000 | 250 |

1. A number of things are incorrect with the above free list. List them.

First address isn’t aligned to a multiple of 8. Xinu’s roundmb macro rounds up to the nearest multiple of 8 bytes. First free block overlaps second free block. Third size needs to be rounded up to 256.

1. What is the difference between a call to Xinu’s getmem() and getstk()?

Getmem() finds the first free memory block in the list whereas getstk() finds the last.

1. In Xinu (and most OSes, for that matter), stack space is meant to only be used by the process which created it. In Xinu, could a potentially malicious process “peek into” the stack space of another process? What about modifying it?

Yes to both. Xinu provides no isolation or protection between processes, allowing them to modify memory that doesn’t belong to them.

|  |  |
| --- | --- |
| **Address** | **Size** |
| 2048 | 256 |
| 2504 | 1024 |

1. Using the same free list as above, what would the free list look like if we called freemem(3528, 253)?

|  |  |
| --- | --- |
| **Address** | **Size** |
| 2048 | 256 |
| 2504 | 1280 |

1. Let’s assume our free list looks like the above. What does the free list look like after a call to freemem(2304, 196)?

|  |  |
| --- | --- |
| **Address** | **Size** |
| 2048 | 1732 |