1. Using Semaphores In class we discussed a solution to the bounded-buffer problem using three semaphores (mutex, emptyBuffers, and fullBuffers):

|  |  |
| --- | --- |
| Producer () {  emptyBuffers. wait ();  mutex. wait ();  put 1 coke in machine;  mutex. post ();  fullBuffers. post ();  } | Consumer() {  fullBuffers. wait ();  mutex. wait ();  take 1 coke from machine;  mutex. post ();  emptyBuffers. post ();  } |

Given each of the following variations, say whether it is correct or incorrect. If you say correct, explain any of the advantages and disadvantages of the new code. If you say incorrect, explain what could go wrong (i.e., trace through an example where it does not behave properly).

* 1. Variation 1

|  |  |
| --- | --- |
| Producer () {  mutex. wait ();  emptyBuffers.wait();  put 1 coke in machine;  fullBuffers.post();  mutex. post ();   } | Consumer() {  mutex. wait ();  fullBuffers. wait ();  take 1 coke from machine;  mutex. post ();  emptyBuffers. post ();  } |

This code is incorrect. It can lead to deadlock. Consider the case where the coke machine is initially full. Suppose a Producer comes and grabs the mutex and then waits for emptyBuffers. A consumer then hangs on mutex and no one will ever consume a coke to empty a buffer. An analogous example is the case where the coke machine is initially empty and a Consumer grabs the mutex and waits for fullBuffers.

* 1. Variation 2

|  |  |
| --- | --- |
| Producer () {  emptyBuffers.wait();  mutex. wait ();  put 1 coke in machine;  fullBuffers.post();  mutex. post ();   } | Consumer() {  fullBuffers. wait ();  mutex. wait ();  take 1 coke from machine;  emptyBuffers. post ();  mutex. post ();  } |

This code is correct. As mentioned in lecture, this code allows more concurrency which is the advantage to coding in this manner. Since the mutex immediately surrounds both the Producer and Consumer actions of putting a coke and taking a code from the machine, if we release the mutex quickly, we achieve better concurrency.

**Introduction quiz**

1. On a pre-EFI PC architecture, the BIOS is loaded from the Master Boot Record (MBR), which then loads the Volume Boot Record (VBR)   
         True        False
2. A process executes faster when running in kernel mode than when running in user mode.   
         True        False
3. A system call always results in a context switch.   
         True        False
4. In an Intel PC architecture, the Master Boot Record (MBR):   
   (a) Loads the operating system.   
   (b) Loads the system BIOS.   
   (c) Loads the Volume Boot Record (VBR).   
   (d) Allows the user to choose which operating system to load.
5. When does preemption take place?   
   (a) When a quantum expires.   
   (b) When a process issues an I/O request.   
   (c) When a process exits.   
   (d) All of the above.
6. In contrast to the BIOS, EFI (Extensible Firmware Interface):   
   (a) Is stored in system firmware instead of on the disk.   
   (b) Loads a boot loader from the master boot record (MBR).   
   (c) Bypasses the Master Boot Record (MBR) and load a boot loader from the volume boot record (VBR) directly.   
   (d) Can load a boot loader from a file system on the disk.
7. Which of these statements is true?   
   (a) A mode switch precedes a context switch.   
   (b) A context switch precedes a mode switch.   
   (c) A context switch can occur without a mode switch.   
   (d) A mode switch is just a different name for a context switch.
8. A context switch takes place at every system call.   
         True        False

**Process quiz**

1. The POSIX execve system call creates a new process.   
         True        False
2. When a process is first launched, the operating system does not know the size of this segment:   
   (a) text   
   (b) data   
   (c) bss   
   (d) heap
3. Which process state transition is not valid?   
   a) Ready → Running   
   (b) Running → Ready   
   (c) Running → Blocked   
   (d) Blocked → Running
4. Which process state transition is valid only on a preemptive multitasking system?   
   (a) Ready → Running   
   (b) Running → Ready   
   (c) Running → Blocked   
   (d) Blocked → Running
5. The wait system call systems puts a process to sleep until:   
   (a) A semaphore wakes it up.   
   (b) The specified elapsed time expires.   
   (c) A child process terminates.   
   (d) The process is preempted by another process.

**Thread quiz**

1. Switching among threads in the same process is more efficient than switching among processes.   
         True        False
2. A Thread Control Block (PCB) stores:   
   (a) User (owner) ID   
   (b) Memory map   
   (c) The machine state (registers, program counter)   
   (d) Open file descriptors
3. What information is stored in a thread control block (TCB)?   
   (a) List of open files.   
   (b) Stack pointer.   
   (c) Memory map.   
   (d) Thread owner ID.
4. Switching between user level threads of the same process is often more efficient than switching between kernel threads because:   
   (a) User level threads require tracking less state.   
   (b) User level threads share the same memory address space.   
   (c) Context Mode switching is not necessary.   
   (d) Execution stays within the same process with user level threads.

**IPC quiz**

1. Mailboxes are a form of messaging using indirect addressing.   
         True        False
2. Rendezvous messaging between processes on the same system reduces the amount of message copying needed.   
         True        False
3. Rendezvous is a form of messaging that uses indirect addressing.   
         True        False

**Synchronization quiz**

1. Using mutual exclusion ensures that a system avoids deadlock.   
         True        False
2. The value of a semaphore can never be negative.   
         True        False
3. Hardware support for mutual exclusion, such as test-and-set locks has the advantage of avoiding the need for spin locks.   
         True        False
4. We need to sum up a huge array of numbers and will use multiple threads to do so. A worker thread is started with a thread ID as a parameter, where threads are numbered 0, 1, ... (numvalues-1). Each of Nworker threads sums up every Nth element of the array in parallel into a local variable, sum. At the end, they each add their answer to a running total called total. A separate thread, main, dispatches the workers and computes and prints the final result.

/\* global values \*/

int total, values[], numvalues, N;

sem s=

event e=

main() {

pthread t[N];

int i;

for (i=0; i < N; ++i) {

pthread\_create(&t[i], NULL, worker, i);

}

Cout << “total” << total << endl;

}

worker(int i) {

int j, sum=0;

for (j=i; j < numvalues; j+=N) {

sum += values[j];

}

total += sum;

}

Using one semaphore, *s*, and one event variable *e*, and without waiting on (joining) threads, define the initial values of *s* and *e* and the placement of *up(s)*/*down(s)* and *advance(e)*/*await(e, value)* operations to ensure that all the summations takes place without race conditions and only when the data is ready.

1. A race condition is:   
   (a) When one process is trying to beat another to execute a region of code.   
   (b) When a process cannot make progress because another one is blocking it.   
   (c) When the outcome of processes is dependent on the exact order of execution among them.   
   (d) A form of locking where processes coordinate for exclusive access to a critical section.
2. Given that we can create user-level code to control access to critical sections (e.g., Peterson’s algorithm), why is it important for an operating system to provide synchronization facilities such as semaphores in the kernel?
3. Two threads are considered to be asynchronous when:   
   (a) They have no reliance on one another.   
   (b) The outcome of a thread is dependent on the specific sequence of execution of both threads.   
   (c) Only one thread is allowed to access a shared resource at a time.   
   (d) The threads require occasional synchronization.
4. A thread that is blocked on a semaphore is awakened when another thread:   
   (a) Tries to decrement a semaphore’s value below 0.   
   (b) Tries to increment the semaphore.   
   (c) Causes the semaphore’s value to reach a specific number.   
   (d) Tries to block on the same semaphore.
5. Condition variables support these operations:   
   (a) Wait / notify   
   (b) Read-and-increment / wait-for-value   
   (c) Increment / decrement-and-wait   
   (d) Set-value / wait-for-value

**Deadlock quiz**

**Scheduling quiz**

1. A process scheduler is responsible for moving processes between these states:   
   (a) Ready and Blocked   
   (b) Running and Blocked   
   (c) Ready and Running   
   (d) Ready, Running, and Blocked
2. Every process gets the same share of the CPU with a:   
   (a) Round-robin scheduler.   
   (b) Shortest remaining time first scheduler.   
   (c) Priority scheduler.   
   (d) Multilevel feedback queues.
3. A weighted exponential average is useful in process scheduling for:   
   (a) Determining the length of a quantum for the thread.   
   (b) Allowing a priority scheduler to assign a priority to the process.   
   (c) Estimating the length of the next CPU burst for the thread.   
   (d) Ordering processes in the ready queue for a round robin scheduler.
4. Process aging:   
   (a) Increases the priority of a process if it sits in the ready state for a long time.   
   (b) Decrease the priority of a process each time the process gets to run.   
   (c) Increases the priority of a process as it gets older.   
   (d) Decreases the priority of a process as it gets older
5. Which scheduler does not risk starvation of processes?   
   (a) Round-robin scheduler.   
   (b) Shortest remaining time first scheduler.   
   (c) Priority scheduler.   
   (d) Multilevel feedback queues.
6. Processes A and B are ready to run at the same time. A needs 600 msec of CPU time. B needs 800 msec of CPU time. Both need to finish within one second. Assume that an Earliest Deadline First scheduler is used. When do A and B terminate?   
   (a) A at 600 msec, B at 1400 msec.   
   (b) A at 1400 msec, B at 800 msec.   
   (c) A and B at 1400 msec.   
   (d) A and B at 1000 msec (1 sec).

Both processes have the same deadline. The scheduler may: (a) run A first, then B (b) run B first, then A (c) alternate among A and B

1. In contrast to a cooperative scheduler, a preemptive scheduler supports the following state transition:   
   (a) Ready → running   
   (b) Running → ready   
   (c) Ready → blocked   
   (d) Blocked → running
2. Starvation is the case when a thread:   
   (a) Loops continuously until it runs out of memory.   
   (b) Is never scheduled to run.   
   (c) Can never acquire a lock on a critical section.   
   (d) Cannot create a child process or thread.
3. A quantum is:   
   (a) The absolute minimum time that a process can run.   
   (b) The maximum time that a process can run before being preempted.   
   (c) The amount of time that a process runs before it blocks on I/O.   
   (d) The fraction of a time slice during which the process is running.
4. Process aging is:   
   (a) Computing the next CPU burst time via a weighted exponential average of previous bursts.   
   (b) The measurement of elapsed CPU time during a process’ execution.   
   (c) Boosting a process’ priority temporarily to get it scheduled to run.   
   (d) Giving a process a longer quantum as it gets older.
5. Differing from a soft deadline, a hard deadline:   
   (a) Is one where it is difficult to predict when the thread will exit.   
   (b) Applies to periodic (nonterminating) rather than terminating processes.   
   (c) Is one where there is no value to the computation if the deadline is missed.   
   (d) Is one where it is difficult to predict when the CPU burst period will end.
6. Which scheduler gives each process an equal share of the CPU?   
   (a) Round robin.   
   (b) Shortest remaining time first.   
   (c) Priority.   
   (d) Multilevel feedback queues.
7. A multilevel feedback queue scheduler generally assigns a long quantum to:   
   (a) High priority processes.   
   (b) Low priority processes.   
   (c) New processes.   
   (d) Old processes.
8. Which scheduler relies on predicting the next CPU burst based on an average of previous bursts?   
   (a) First-come, first served.   
   (b) Round robin   
   (c) Shortest remaining time first.   
   (d) Multilevel feedback queues.
9. Multilevel queues allow multiple processes to share the same priority level.   
         True        False