Study Set for Lecture 07: Process Synchronization

Due Oct 12 at 11:59pm **Points** 10 **Questions** 13
Available Oct 6 at 12pm - Dec 8 at 8pm 2 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect07_Process_Synchronization.pdf**. The archive of the accompanying code is in **lect07code.zip**.

Then answer the questions from this study set and submit them to gain access to the further part of the course.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 2	25 minutes	0 out of 10
LATEST	Attempt 2	25 minutes	0 out of 10
	Attempt 1	2,867 minutes	0 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **0** out of 10
Submitted Oct 13 at 9:41am
This attempt took 25 minutes.

Question 1 — 0 / 0 pts

In the lecture, we said that the implementation of semaphores constitutes a critical section problem itself? Explain why.

Why using spinlocks is not a big problem in this implementation?

Your Answer:

The **implementation of semaphores is a critical section** problem because it must guarantee that no two processes can execute wait() and signal() on the same semaphore at the same time.

A semaphore is an integer variable with two indivisible (atomic) standard operations to modify it. Yes it still suffers from the busy-waiting problem. Yes it is possible to implement semaphores without busy-waiting. Instead, it just implements block and wake up. Block places the process invoking the operation on the appropriate waiting queue, and then block. Wake up removes one of the processes in the waiting queue and place it in the ready queue.

Using **spinlocks** is not a problem because the implementation of wait() and signal() is short so there is little busy-waiting if the critical section is rarely occupied

Question 2

0 / 0 pts

In the lecture notes, we analyze step-by-step two possible scenarios for scheduling processes that use Peterson's solution to the synchronization problem. Using similar analysis, analyze two potential scenarios for executing at least two processes that utilize an atomic operation of testing and setting variables for entering their critical sections. Show all the steps in your analysis of the scenarios.

Your Answer:

Scenario 1

flag[1] = TRUE;

turn = 2;

while (flag[2] && turn == 2) ;

// critical section

flag[1] = FALSE;

flag[1] = TRUE;

turn = 2;

while (flag[2] && turn == 2) ;

flag[2] = TRUE;

turn = 1;

while (flag[1] && turn == 1) ;

// critical section

flag[2] = FALSE;

flag[2] = TRUE;

turn = 1;

while (flag[1] && turn == 1) ;

// critical section

Scenario 2

flag[1] = TRUE;

turn = 2;

while (flag[2] && turn == 2) ;

// critical section

flag[1] = FALSE;

flag[2] = TRUE;

turn = 1;

while (flag[1] && turn == 1) ;

// critical section

flag[2] = FALSE;

Question 3

0 / 0 pts

In the lecture, we said that the solution of the Dining Philosophers problem with semaphores suffers from a potential starvation problem (a deadlock); explain why. Provide a detailed step-by-step example that leads to a deadlock.

Your Answer:

Deadlock is two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes. If all of the philosophers get hungry at the same time, they will all reach for the chopstick on the left and because there are no chopsticks left on the table, they all can't eat because they all need another chopstick.

A deadlock example could involve 4 cars crossing an intersection, and if there is no proper signaling and they all go at the same time, none of them will be able to move any further

Question 4

0 / 0 pts

Define what a monitor is.

Explain what problems related to semaphores led to the introduction of monitors?

Why is using monitors safer than using semaphores?

Your Answer:

A monitor is a high-level abstraction that provides a convenient and effective high-level language mechanism for process synchronization, and only one process may be active in the monitor at a time. If a semaphore is used incorrectly, then it can result in timing errors that are hard to detect, as they only occur in certain sequences and don't always occur. The monitor type also declares the variables whose values define the state of an instance of that type, along with the bodies of functions that operate on

those variables. Thus, a function defined within a monitor can access only those variables declared locally within the monitor and its formal parameters.

Question 5

0 / 0 pts

Define what a spinlock is.

Under what conditions using spinlocks might be acceptable.

Your Answer:

A spinlock is when a process is in it's critical section and another process tries to enter theirs, the process loops continuously in the entry code, it's called that because the process is "spinning" while waiting for the lock to become available.

On single processor systems, the thread holding a lock cannot be running while another thread is testing the lock, because only one thread/process can be running at a time. Therefore the thread will continue to spin and waste CPU cycles until its time slice end

They are often used in multiprocessor systems because some processors can spin-lock while others continue to execute. It can spin while waiting for the lock

Question 6

0 / 0 pts

What is a semaphore? Does it suffer from the busy-waiting problem? Is it possible to implement it so that it does not?

Explain how a semaphore can be used to solve the critical section problem. Please note that that requires a proof that using a semaphore indeed is the basis for a solution to the critical section problem.

Your Answer:

A semaphore is an integer variable with two indivisible (atomic) standard operations to modify it. It does suffer from the busy-waiting problem. It can be implemented so it doesn't by using a waiting queue with each semaphore so it now has a value (of type integer; like before) and a pointer to the waiting queue, and with 2 operations, block (place in waiting queue) and wakeup (move to ready queue).

A solution to the critical section problem using semaphores is using a mutex to enforce mutual exclusion so that only one process is in it's mutual exclusion at a time, this is accomplished by using wait() before a critical section and then signal() once the critical section ends

Question 7

0 / 0 pts

Explain what a race condition is.

Provide two examples with a detailed step-by-step justification for including them.

Your Answer:
A race condition is when there is concurrent access to shared data and the final outcome depends upon the order of execution that is controlled by the OS and may vary over time

Whoever gets up first gets to use the bathroom first, they both want to shower but the order of execution says that whoever wakes up first gets to use it first

Driving to a stopsign, and the one who gets there first drives through it first. Whoever arrives first can use it thanks to order of execution

Question 8

0 / 0 pts

Explain with details what a critical section is.

Provide two real-life examples of problems that are similar to the critical section problem and explain why do you think they fall to the same category.

Your Answer:

A critical section is a section of code prone to race condition (where shared data is accessed), no other critical section is allowed to run at the same time as another one. (CO-OPERATION)

If you and a roommate are both out of milk and you both go to get some without consulting each other, you then have too much milk. It's similar because if you cooperated with your roommate, only one of you would get the milk and you wouldn't have too much.

If you need to leave a car at a mechanic and both you and your spouse both decide to take the car that needs repairing to the mechanic, but since you both went in the same car, there's no car to take you two home. If you two cooperated, then you would know to take another car so you can go back home

Question 9

0 / 0 pts

Provide a formal proof that blocking interrupts can be used as a solution for the critical section problem on a single-processor computer.

Would it work on a multiprocessor machine? Justify your answer.

Your Answer:

When a process is ready to enter into its critical section it sets a flag to disable interrupts. This ensures that no other process can be scheduled

and ensures mutual exclusion re-enabling the interrupts after the critical section can ensure progress and bounded waiting. On multiprocessor machines, the flag for disabling interrupts would have to be relayed to all processors which is costly

Question 10

0 / 0 pts

Provide a proof that the two-process Peterson's solution satisfies all requirements for a correct solution to the critical section problem.

HINT: Analyze all possible scheduling cases for two processes.

Your Answer:

Requirements for a solution:
Mutual Exclusion: if process Pi is executing in its critical section, then no other processes can be executing in their critical sections
Progress: if no process is executing in its critical section and there exist some processes that which to enter their critical section, then the section of the processes that will enter the critical section next cannot be postponed indefinitely
Bounded Waiting: a bound just exit the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Solutions:
Mutual Solution: for a two process system, P0 and P1, they can never be in their critical section at the same time since no state can satisfy both turn = 0 and turn = 1
Progress: The peterson algorithm satisfies this condition since the OS selects P0 first, P0 will set the flag[0] condition to false after it exits its critical section therefore allowing P1 to break its busy wait every time and will access its critical section
Bounded waiting: Petersons algorithm satisfies this condition since the wait for one process to access its critical section will never be longer than one turn. For example, once P0 will give priority to P1 once it exits its critical section by setting flag[0] to false and then break P1's loop.

Question 11

0 / 0 pts

What requirements does a solution to the critical section problem need to satisfy?
Provide a commentary for each.

Your Answer:

Mutual exclusion, if a process is executing in it's critical section, no other processes can be in it's critical section at the same time.

Progress, if no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

Bounded Waiting, a bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Question 12

0 / 0 pts

In the lecture notes, we analyze step-by-step two possible cases for scheduling processes that use Peterson's solution to the synchronization problem. Using similar analysis, analyze two potential scenarios for executing at least two processes that utilize an atomic operation of comparing and swapping two variables for entering their critical sections. Show all the steps in your analysis of the scenarios.

Your Answer:

Scenario 1

```
flag[1] = TRUE;
turn = 2;
while (flag[2] && turn == 2) ;
// critical section
flag[1] = FALSE;
flag[1] = TRUE;
turn = 2;
while (flag[2] && turn == 2) ;
```

```
flag[2] = TRUE;
turn = 1;
while (flag[1] && turn == 1) ;
// critical section
flag[2] = FALSE;
flag[2] = TRUE;
turn = 1;
while (flag[1] && turn == 1) ;
// critical section
```

Scenario 2

```
flag[1] = TRUE;
turn = 2;
while (flag[2] && turn == 2) ;
// critical section
flag[1] = FALSE;
```

```
flag[2] = TRUE;
```

```
turn = 1;
while (flag[1] && turn == 1) ;
// critical section
flag[2] = FALSE;
```

Question 13

10 / 10 pts

I have submitted answers to all questions in this study set.

- (●) True
- ( ) False

Quiz Score: **0** out of 10