CSCE 3613 Operating Systems Homework #3, ver. 7.5

Process Synchronization and Deadlocks

Name: Morgan Maness

ID: 010813680

31 pts.

# **Instructions**

* Type your work, print it to a \*single\* PDF, and upload it to Blackboard before the due date and time. It is strongly suggested to use the given document.
* Show all of your work. Without proper justification and details of steps, correct answers alone may not carry full credit.
* -2 points if you do not insert your name and ID at the top of the document.
* -5 points if it is not typed.
* -5 points if it is not a PDF file.
* -5 points if it is not a single PDF file. Submit one PDF file. Do not submit zip files containing one or more files.
* -5 points if you present the worked problems out of order. In other words, please present the problems in the order assigned, 1, 2, 3, …

1. (3 pts.) What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. A process could wait by relinquishing the processor, and a block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

Busy Waiting: Process holding CPU waiting for condition

Avoiding Busy Waiting: overhead of putting process on queue cannot be avoided

2. (2 pts.) Discuss the tradeoff between fairness and throughput of operations in the First Readers-Writers problem.

Throughput in the readers-writers problem is increased by favoring multiple readers as opposed to allowing a single writer to exclusively access the share values.

No reader is kept waiting unless a writer already has permission to write. Writers can starve. Not fair to writers, readers have better throughput

3. (3 pts.) How does the signal() operation associated with monitors differ from the corresponding operation defined for semaphores?

The signal() operations associated with monitors is not persistent. If a signal is performed and if there are no waiting threads, then the signal is simply ignored, and the system does not remember the fact that the signal took place. If a subsequent wait operation is preformed, then the corresponding thread simply blocks. Whereas in semaphores, every signal result in a corresponding increment of the semaphore value even if there are no waiting threads. A future wait operation would immediately succeed because of the earlier increment.

Signal = +1 operation

Wait = -1 operation

If need to wake up process, signal will wake up. But if no one process is waiting in monitor, signal does nothing

4. (4 pts.) Describe the four necessary conditions for deadlock.

Mutual exclusion: at least one resource must be non-sharable

Hold and wait: When a process is holding a resource, and requests another resource

No preemption: A resource can be released only voluntarily by the process holding it

Circular wait: P1 needs something from P2, P2 needs something from P3, …, Pn needs something from P1

5. (3 pts.) Describe the difference between deadlock prevention and deadlock avoidance.

Deadlock prevention ensures that at least one of the necessary conditions to cause a deadlock will never occur while deadlock avoidance ensures that the system will never enter an unsafe state

Deadlock Prevention - Eliminate at least one necessary condition for deadlock

Deadlock Avoidance - Requires OS be given in advance additional information concerning resources a process request

6. Consider the resource-allocation graph of a system below with 3 processes and 3 resources each with a single instance.



6.a. (5 pts.) Is the state of the system safe? Explain.

Yes, because there are no loops so there is no possibility of deadlock.

No cycles, no circular wait = safe

6.b. (3 pts.) Starting with the original resource-allocation graph, assume R1 is assigned to P1. With this assignment, is the system safe or unsafe? Explain why.

This system is unsafe. It has a loop P1->R3->P3->R1->P1. P1 and P3 can put themselves in deadlock because one can be using the register the other needs, while waiting on the register the other is currently using.

6.c. (3 pts.) Starting with the original resource-allocation graph, assume R1 is assigned to P2. With this assignment, is the system safe or unsafe? Explain why.

Yes, the system is safe because there are no loops, so there is no possibility of deadlock.

7. The following system has 11 hard drives and each process declares its maximum needs. In addition, the current needs (hard drives) that each process is holding is listed.

|  |  |  |
| --- | --- | --- |
| Process | Maximum Needs | Current Needs |
| P0 | 4 | 1 |
| P1 | 11 | 4 |
| P2 | 7 | 1 |
| P3 | 3 | 2 |
| P4 | 6 | 2 |

7.a. (5 pts.) What is a safe sequence for this system? Prove it.

This is deadlock avoidance

Current Needs = 10 drives being used

Currently only 1 left, so find a process that only needs one more

P3 can request 1 more drive; 10+1=11 are used; P3 releases 3 drives

11-3 = 8 drives being used, 3 available

P0 can request 3 more drives; 8+3=11 are used; P0 releases 4 drives

11-4 = 7 drives being used, 4 available

So on until all processes complete

P3 -> P0 -> P4 -> P2 -> P1