Test 2 Study Guide
Operating Systems 20032 Section 1

1. General Information

I plan on having the test be a mix of short answer, essay and problems, ala the first test. The chapters to be covered are chapters 7 and 8, with emphasis on things we covered in class. You will again have the opportunity to bring cheat sheet, on paper I provide, with notes on it to the test.

2. Terminology & Basic Concepts You Should Be Able to Describe

- deadlock
- mutual exclusion
- critical section
- semaphore
- monitor
- difference between gladiator and mediator monitor types
- similarities and difference between P and wait, and V and signal
- 4 conditions necessary for deadlock
- 3 techniques for deadlock handling: prevention, avoidance, and detection/recovery

3. Problems

3.1. Semaphores

Examples of each of these types of problems in given in a later section.

(1) You should be able to show me how to use one to protect a critical section.
(2) You should be able to show me how to use them to synchronize processes such as a producer/consumer, or reader/writer, etc.
(3) You should be able to tell me what value to initialize a semaphore to in a particular situation.
(4) If I give you a "solution" to a simple synchronization problem, you should be able to tell me if it works or not - ala our dining philosophers solutions we looked at in class. Does it provide appropriate mutual exclusion? Does it lead to a possibility of deadlock? Does it provide the necessary synchronization?

3.2. Monitors

(1) You should be able to write a function for a monitor, given the other functions for it (ie. extend something we’ve done in class). There are problems like this in the problem set.
(2) If I give you some monitor code, you should be able to identify a problem in it.
(3) If I ask you to explain some aspect of the monitor examples handed out in class, you should be able to do it (e.g. why is the local variable needed in retrieve_date in the Producer/Consumer code).
3.3. Deadlock
Examples of each of these types of problems is in the problem set.

(1) You should be able to apply the Banker’s Algorithm to single resource problems and multiple resource problems.

(2) You should be able to do resource graph reductions for serially reusable resources and tell if the system is in a deadlock or not.

4. Discuss Issues

(1) You should be able to compare some methods for deadlock prevention, or for deadlock avoidance, or deadlock detection and recovery. Which ones can raise problems with starvation? What other problems might the technique introduce (e.g. lost work, etc)? What information do you need to have to be able to implement such a technique (e.g. many require you know in advance the maximum number of each source type that a process needs).

(2) Sort of an oddball one, but something to think about. Solaris has semaphores which act pretty much like what we’ve seen, and they also have mutexes, which have an "owner". With a mutex, the process or threads that does the P is the only one that can do the corresponding V, whereas with semaphores any process can do the V regardless of who did the V. What is an advantage of a mutex over a semaphore? What is a disadvantage? Would you have wanted to use a mutex in the first project?

(3) Another more interesting one. First some background. There is the notion of "contention scope", which has to do with how threads are scheduled. In simplistic terms, "process scope" means that all the threads within a single process are bound together as one unit as far as the OS scheduler is concerned, and within the process the thread manager figures out which thread will run. "System scope" means that all the threads in a single process are scheduled as independent entities by the OS scheduler. We’ve seen a negative impact of process scope in some examples in class; in specific, if one thread is in an infinite loop, none of the other threads get a chance to run (because within the process the scheduling is non-preemptive). Process scope is the default.

Given that there are potential negative impacts with process scope, why would the designers choose process scope to be the default instead of system scope?

5. Problem Sets

5.1. Semaphores

(1) Here’s a routine used in a multi-threaded process:

```c
(void *)some_function( void *arg ) {
    if (arg ) {
        shared_data1 += 1;
        shared_data2 *= shared_data1;
    }
    else {
        cout << "error" << endl;
    }
} // some_function
```

The variables `shared_data1` and `shared_data2` are shared by all the threads in the process. Assume a variable of class Semaphore has been declared, named `lock`, and is accessible to all the threads.
a) Add operations on \texttt{lock} to ensure mutual exclusion in the critical section of the above function. It is important that the thread be blocked for the minimum time absolutely necessary to ensure the mutual exclusion.
b) What value should \texttt{lock} be initialized to?

(2) Three threads need to synchronize their actions so that the following holds:
\texttt{P1} -> \texttt{data1} -> \texttt{P2} -> \texttt{data2} -> \texttt{P3}
meaning that \texttt{P1} produces \texttt{data1}, and only when \texttt{data1} is ready can \texttt{P2} proceed, etc. You are guaranteed that each thread will be executed only once, and there are no cycles (ie. what you see above is the whole story).

Here’s the skeleton of each thread (note: where it says "do some work" or "do more stuff", it means do work independent of the shared data):

\begin{verbatim}
thread1() { thread2() { thread3() {
    // do some work    // do some work    // do some work
    ....
    ....
    // produce data1  // do stuff with data1  // do stuff with data2
    data1 := 1;
    // and produce data2  cout << data2 << endl;
    data2 := data2 * data1;
    // do more stuff    // do more stuff    // do more stuff
    ....
    ....
    ....
} }
}
\end{verbatim}

Add Semaphores and operations on them as needed, assuming that all threads can access them, to ensure the proper order of execution. Indicate the initial value for each Semaphore.

(3) A proposed solution to the Dining Philosophers deadlock problem is as follows:

\begin{verbatim}
Philosopher( int i ) {
    while(1) {
        think();

        // grab forks if we can
        lock.P();
        fork[i].P();
        fork[(i+1)%5].P();
        lock.V();

        eat();

        // put down forks
        lock.P();
        fork[i].V();
        fork[(i+1)%5].V();
        lock.V();
    }
}
\end{verbatim}

There are five philosophers and five forks. All the lock and fork semaphores are initialized to 1.

a) Is the second lock.P/lock.V pair necessary? Why or why not?
b) If the second lock.P/lock.V pair isn’t necessary, are there any negative consequences to having it there?
5.2. Monitors

(1) In the Dining Philosophers monitor handed out in class, would it be ok in the `pickup_forks` function to move the line

```c
    int other (( who + 1 ) % 5);
```

before the line

```c
    enter();
```

Why or why not? The code is available in `~mmr/pub/OS/DiningPhilosophers/monitor`.

(2) For the Producer/Consumer monitor handed out in class, write a function named `peek`. It takes no arguments. It returns an integer that contains the contents of the next full buffer, but does not change the state of the indices into the buffer (i.e., it reads but doesn’t “consume” the information). If there is no full buffer, it should wait until some data is available. The code is available in `~mmr/pub/OS/ProducerConsumer/monitor`.

(3) For the Producer/Consumer monitor handed out in class, modify the `retrieve_data` function so that it doesn’t block if there are no full buffers.

5.3. Banker’s Algorithm

(1) Given 5 total units of the resource, tell whether the following system is in a safe or unsafe state. Show your work.

<table>
<thead>
<tr>
<th>Process</th>
<th>Used</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

(2) Given a total of 5 units of resource type 1 and 4 units of resource type 2, tell whether the following system is in a safe or unsafe state. Show your work.

<table>
<thead>
<tr>
<th>Process</th>
<th>Type 1 Used</th>
<th>Type 1 Max</th>
<th>Type 2 Used</th>
<th>Type 2 Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

(3) Given a total of 10 units of a resource type, and given the safe state shown below, should process 2 be granted a request of 2 additional resources? Show your work.

<table>
<thead>
<tr>
<th>Process</th>
<th>Used</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P5</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
5.4. Resource Graphs

(1) A system has four processes P1 through P4 and two resource types R1 and R2. It has 2 units of R1 and 3 units of R2. Given that:

P1 requests 2 units of R2 and 1 unit of R1
P2 holds 2 units of R1 and 1 unit of R2
P3 holds 1 unit of R2
P4 requests 1 unit of R1

Show the resource graph for this state of the system. Is the system in deadlock, and if so, which processes are involved? Show your work.

(2) A system has five processes P1 through P5 and four resource types R1 through R4. There are 2 units of each resource type. Given that:

P1 holds 1 unit of R1 and requests 1 unit of R4
P2 holds 1 unit of R3 and requests 1 unit of R2
P3 holds one unit of R2 and requests 1 unit of R3
P4 requests 1 unit of R4
P5 holds one unit of R3 and 1 unit of R2, and requests 1 unit of R3

Show the resource graph for this state of the system. Is the system in deadlock, and if so, which processes are involved? Show your work.