CS3161 Operating System Principles
Fall 2007
Homework 2
Due Monday Nov 5th, 2007 4:30PM on class with hard copy

1. Servers can be designed to limit the number of open connections. For example, a server may wish to have only *N* socket connections at any point in time. As soon as *N* connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

Answer: A semaphore is initialized to the number of allowable open socket connections. When a connection is accepted, the acquire() method is called, when a connection is released, the release() method is called. If the system reaches the number of allowable socket connections, subsequent calls to acquire() will block until an existing connection is terminated and the release method is invoked.

2. 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.

Answer: *Busy waiting* means that a process is waiting for a condition to be satisfied in a tight loop without relinquish the processor. Alternatively, a process could wait by relinquishing the processor, and 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.

3. Consider the traffic deadlock depicted in Figure 7.1.
a. Show that the four necessary conditions for deadlock indeed hold in this example.
b. State a simple rule for avoiding deadlocks in this system.

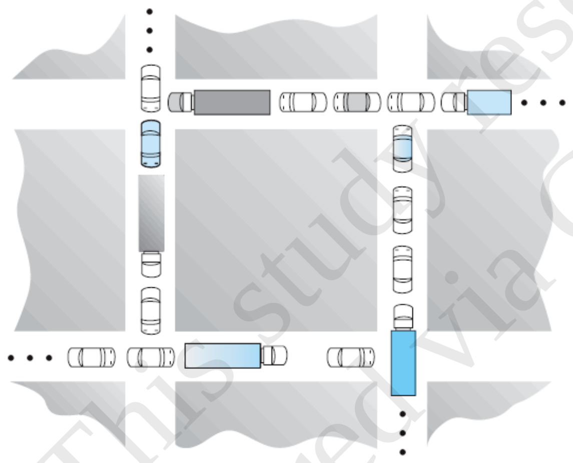

Figure 7.1 Traffic deadlock for Exercise 7.1.

Answer:
a. The four necessary conditions for a deadlock are (1) mutual exclusion;
(2) hold-and-wait; (3) no preemption; and (4) circular wait.
The mutual exclusion condition holds as only one car can occupy a space in the roadway. Hold-and-wait occurs where a car holds onto their place in the roadway while they wait to advance in the roadway. A car cannot be removed (i.e. preempted) from its position in the roadway. Lastly, there is indeed a circular wait as each car is waiting for a subsequent car to advance. The circular wait condition is also easily observed from the graphic.

b. A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear they will not be able to immediately clear the intersection.

4. Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	ABCD	*ABCD*	*ABCD*
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

Answer the following questions using the banker's algorithm:
a. What is the content of the matrix *Need*?
b. Is the system in a safe state?
c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately?

Answer:
a. What is the content of the matrix *Need*? The values of *Need* for processes P_0 through P_4 respectively are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).
b. Is the system in a safe state? Yes. With *Available* being equal to (1, 5, 2, 0), either process P_0 or P_3 could run. Once process P_3 runs, it releases its resources which allow all other existing processes to run.
c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately? Yes it can. This results in the value of *Available* being (1, 1, 0, 0).One ordering of processes that can finish is P_0, P_2, P_3, P_1, and P_4.

5. A single-lane bridge connects the two Vermont villages of North Tunbridge and South tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a northbound and a southbound farmer get on the bridge at the same time (Vermont farmers are stubborn and are unable
to back up.) Using semaphores, design an algorithm that prevents deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

Answer:
```
semaphore ok to cross = 1;
void enter bridge() {ok to cross.wait();
}
void exit bridge() {ok to cross.signal();
}
```

6. Consider a paging system with the page table stored in memory.
a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?
b. If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)

Answer:
a. 400 nanoseconds; 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.
b. Effective access time = 0.75 × (200 nanoseconds) + 0.25 × (400 nanoseconds) = 250 nanoseconds.

7. (a) Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.
(b) Why are segmentation and paging sometimes combined into one scheme?

(a) Answer: Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

(b) Answer: Segmentation and paging are often combined in order to improve upon each other. Segmented paging is helpful when the page table becomes very large. A large contiguous section of the page table that is unused can be collapsed into a single segment table entry with a page table address of zero. Paged segmentation handles the case of having very long segments that require a lot of time for allocation. By paging the segments, we reduce wasted memory due to external fragmentation as well as simplify the allocation.