CS 326-02 Operating Systems Name________________________
Fall 2014 Thursday, 23-Oct-2014

Midterm Exam

Closed book/Closed notes
1 - 8.5 X 11 sheet of notes allowed
Show all work

(20 points) 1) a) Suppose a thread is running in a critical section of code, meaning that it has acquired all the locks through proper arbitration. Can it get context switched? Why or why not?
Yes, a process holding a lock can get context switched. Locks (especially user-level locks) are independent of the scheduler. (Note that threads running in the kernel with interrupts disabled would not get context-switched).

b) Name the four conditions required for deadlock and give a brief (one sentence) description of each.
Mutual exclusion: a resource can be possessed by only one thread.
Hold and wait: A thread can hold a resource such as a lock while waiting for another.
No preemption: The resource cannot be taken away from the thread.
Circular wait: Two or more threads form a circular chain where each thread waits for a resource that the next thread in the chain holds

c) Does a cyclic dependency always lead to deadlock? Why or why not?
No. If multiple equivalent resources exist, then a cycle could exist that is not a deadlock. The reason is that some thread that is not part of the cycle could release a resource needed by a thread in the cycle, thereby breaking the cycle.

d) What is the difference between deadlock prevention and deadlock avoidance? What category does Bankers algorithm falls in and why?
Deadlock prevention prevents deadlock by preventing one of the four conditions required for deadlock to occur.
Deadlock avoidance ensures the system is always in a safe state by not granting requests that may move the system to an unsafe state. A is consisered safe if it is possible for all processes to finish executing (i.e. a sequence exists such that each process can be given all its required resources, run to completion, and return resources allocated, thus allowing another process to do the same, etc until all process complete). Deadlock avoidance requires the system to keep track of the resources such that it knows the allocated, available, and remaining resource needs.
The Bankers algorithm is a deadlock avoidance scheme since it defines an algorithm and structure to ensure the system remains in a safe state before granting any new resource requests.

e) Why would two processes want to use shared memory for communication instead of using message passing?
Performance. Communicating via shared memory is often faster and more efficient since there is no kernel intervention and no copying.

f) What is internal fragmentation? External fragmentation? Give a brief example of each.
Internal fragmentation: allocated space that is unused. For example, a process may have allocated a 1KB page, but uses only 10 bytes (internal to a page)
External fragmentation: unallocated "free" space that lies between allocated space such that contiguous regions of free space is insufficient to satisfy subsequent space allocation requests even though sufficient free space exists in aggregate (external to a page).

(20 points) 2) CPU Scheduling.
Here is a table of processes and their associated arrival and running times.

ProcessID	Arrival Time	Expected CPU Running Time
1	0	5
2	1	5
3	5	3
4	6	2

a) Show the scheduling order for these processes under First-In-First-Out (FIFO), Shortest-Job First (SJF), and Round-Robin (RR) with a quantum = 1 time unit.

Time	**FIFO**	**SJF**	**RR**
0	**1**	**1**	**1**
1	**1**	**1**	**2**
2	**1**	**1**	**1**
3	**1**	**1**	**2**
4	**1**	**1**	**1**
5	**2**	**3**	**3**
6	**2**	**3**	**4**
7	**2**	**3**	**2**
8	**2**	**4**	**1**
9	**2**	**4**	**3**
10	**3**	**2**	**4**
11	**3**	**2**	**2**
12	**3**	**2**	**1**
13	**4**	**2**	**3**
14	**4**	**2**	**2**
15			

b) For each process in each schedule above, indicate the queue wait time and turnaround time (TRT).

Scheduler	Process 1	Process 2	Process 3	Process 4
FIFO queue wait	0	4	5	7
FIFO TRT	5	9	8	9
SJF queue wait	0	9	0	2
SJF TRT	5	14	3	4
RR queue wait	8	9	6	3
RR TRT	13	14	9	5

The queue wait time is the total time a process spends in the wait queue.
The turnaround time is defined as the time a process takes to complete after it first arrives.

(20 points) 3) Virtual Memory Page Replacement
Given the following stream of page references (page A, page B, Page C, etc.) by an application, calculate the number of page faults the application would incur with the following page replacement algorithms. Assume that all pages are initially free.

Reference Stream: A B C D A B E A B C D E B A B
a) FIFO (First In/First Out) page replacement with 3 physical pages available.

Reference Stream	A	B	C	D	A	B	E	A	B	C	D	E	B	A	B
Mem Page 1	A	A	A	D	D	D	E	E	E	E	E	E	B	B	B
Mem Page 2		B	B	B	A	A	A	A	A	C	C	C	C	A	A
Mem Page 3			C	C	C	B	B	B	B	B	D	D	D	D	D
Page Fault	√	√	√	√	√	√	√			√	√		√	√	

(11 Page Faults)

b. LRU (Least Recently Used) page replacement with 3 physical pages available.

Reference Stream	A	B	C	D	A	B	E	A	B	C	D	E	B	A	B
Mem Page 1	A	A	A	D	D	D	E	E	E	C	C	C	B	B	B
Mem Page 2		B	B	B	A	A	A	A	A	A	D	D	D	A	A
Mem Page 3			C	C	C	B	B	B	B	B	B	E	E	E	E
Page Fault	√	√	√	√	√	√	√			√	√	√	√	√	

(12 Page Faults)

c. Optimal page replacement with 3 physical pages available.

Reference Stream	A	B	C	D	A	B	E	A	B	C	D	E	B	A	B
Mem Page 1	A	A	A	A	A	A	A	A	A	C	D	D	D	A	A
Mem Page 2		B	B	B	B	B	B	B	B	B	B	B	B	B	B
Mem Page 3			C	D	D	D	E	E	E	E	E	E	E	E	E
Page Fault	√	√	√	√			√			√	√			√	

(8 Page Fault)

d. True or False. If we increase the number of physical pages from 3 to 4, the number of page faults always decreases using FIFO page replacement. Briefly explain.

False due to Belady's anomaly. For instance, for this problem if we increase the number of physical pages available from 3 to 4, then the number of page faults increases from 11 to 12 using a FIFO page replacement algorithm.

(20 points) 4) Memory Management

a) Consider a memory system with a cache access time of 10ns and a memory access time of 200ns. If the effective access time is 10% greater than the cache access time, what is the hit ratio p (probability of page being in memory)? (Fractional answers are okay).

Effect Access Time: $T_e = H \times T_c + (1 - H)(T_m + T_c)$,

where T_c = 10ns, $T_e = 1/1 \times T_c$, and $T_m + T_c$ = 200ns.

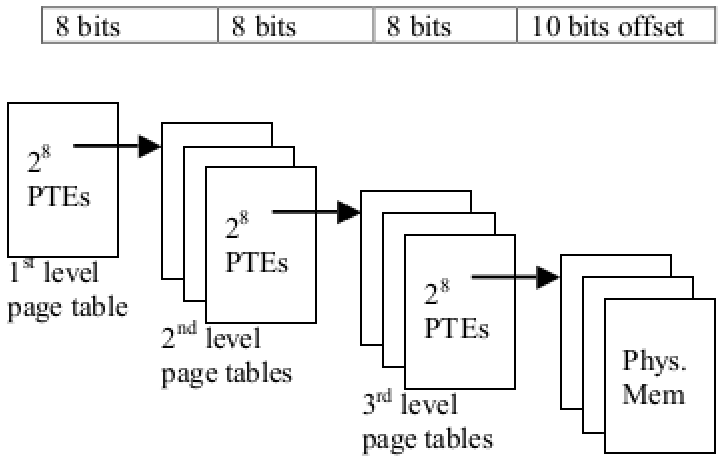
Thus, (1.1) x (10) = H x 10 + (1 – H) (200)

11 = 10H + 200 – 200H

H = 189/190

b) Assuming a page size of 1 KB and that each page table entry (PTE) takes 4 bytes, how many levels of page tables would be required to map a 34-bit address if every page table fits into a single page. Be explicit in your explanation.

Since a page is 2^{10} bytes (1 KB) and each PTE is 2^2 bytes (4 bytes), a 1-page page table contains 256 or 2^8 PTEs ($2^{10}/2^2 = 2^8$). Each entry points to a page that is 2^{10} bytes (1 KB). With one level of page tables we can address a total of $2^8 \times 2^{10} = 2^{18}$ bytes). Adding another level yields another 2^8 pages of page tables, addressing a total of $2^8 \times 2^8 \times 2^{10}$ = 2 26 bytes. Finally, adding a third level 2^8 pages of page tables, addressing a total of $2^8 \times 2^8 \times 2^8 \times 2^{10} = 2^{34}$ bytes. So, we need 3 levels.



(20 points) 5) Concurrency: the “H2O” problem
You have just been hired by Mother Nature to help her out with the chemical reaction to form water, which she does not seem to be able to get right due to synchronization problems. The trick is to get two H atoms and one O atom all together at the same time. The atoms are threads. Each H atom invokes a procedure hReady when it is ready to react, and each O atom invokes a procedure oReady when it is ready. For this problem, you are to write the code for hReady and oReady. The procedures must delay until there are at least two H atoms and one O atom present, and then one of the threads must call the procedure makeWater (which just prints out a debug message that water was made). After the makeWater call, two instances of hReady and one instance of oReady should return. Your solution should avoid starvation and busy-waiting. You may assume that the semaphore implementation enforces FIFO order for wakeups—the thread waiting longest in P() is always the next thread woken up by a call to V().

a) Specify the correctness constraints. Be succinct and explicit.

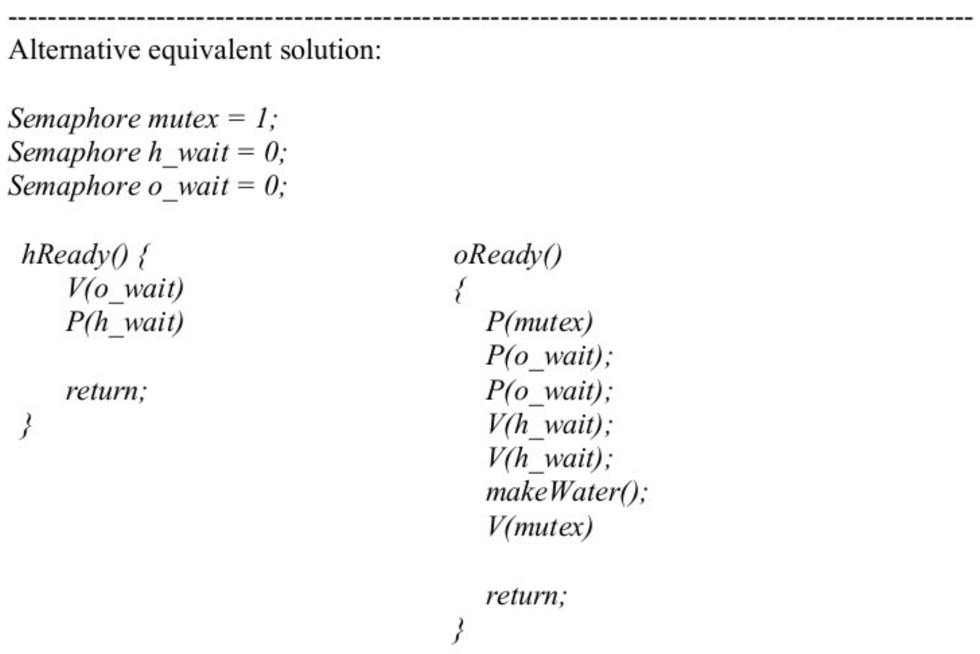
- No call to makeWater should be made when fewer than 2 H atoms have called hReady and no O atom has called oReady.
- Exactly two H atoms return and one O atom returns for every call to makeWater.

b) Provide the pseudo implementation of hReady and oReady using semaphores.

```
Semaphore mutex = 1;
Semaphore h_wait = 0;
Semaphore o_wait = 0;
int count = 0;

 hReady() {                          oReady()
     P(mutex);                       {
     count++;                           P(o_wait);
     if(count %2 == 1) {                V(h_wait);
          V(mutex);                     V(h_wait);
          P(h_wait);                    makeWater();
     } else {
          V(o_wait);                    return;
          P(h_wait);                 }
          V(mutex);
     }

     return;
 }
```



Alternative equivalent solution:

```
Semaphore mutex = 1;
Semaphore h_wait = 0;
Semaphore o_wait = 0;

 hReady() {                       oReady()
     V(o_wait)                    {
     P(h_wait)                        P(mutex)
                                      P(o_wait);
     return;                          P(o_wait);
 }                                    V(h_wait);
                                      V(h_wait);
                                      makeWater();
                                      V(mutex)

                                      return;
                                  }
```