Name: ____________________________

Operating Systems
V22.0202 Spring 2011

Midterm Exam

ANSWERS

1. **True/False**. Circle the appropriate choice (there are no trick questions).

   (a) **T** The CPU's kernel mode provides operations that are not available in user mode.

   (b) **F** The disk driver is the microprocessor within the hard drive that controls the movement of the disk arm.

   (c) **F** A trap is an interrupt caused by an external event such as a mouse click.

   (d) **T** A context switch is initiated by an interrupt, such as clock interrupt or a trap.

   (e) **T** In a batch system, every process runs to completion before the next process runs.

   (f) **F** All the processes on a modern computer share a single virtual address space.

   (g) **T** In lottery process scheduling, the total starvation of lower-priority processes is avoided as long as they are each given at least 1 "lottery ticket".

   (h) **T** In round robin process scheduling, the longer the quantum the more efficient the use of the CPU (i.e. fewer wasted cycles).

   (i) **T** On a multiprogramming system supporting swapping (but not virtual memory), base and limit registers can be used to point to the beginning and end of the memory space occupied by the current running process.

   (j) **F** There is no need to have virtual memory on a computer whose physical memory is bigger than the size of the address space.

2. **Short Answers: Answer this question on this sheet**

   (a) In a batch system using shortest-job-first scheduling, what is the average turnaround time for a set of four jobs whose running times are 13, 3, 10 and 15 minutes? Show your work.

   The first process (i.e. the one that takes 3 minutes) would finish after 3 minutes. The second process (which takes 10 minutes) would finish after a total of 13 minutes, the third process would finish at 26 minutes, and the fourth process would finish after 41 minutes. Therefore, the average turnaround time is $(3 + 13 + 26 + 41)/4 = 83/4 = 20.75$**.**

   (b) In a multiprogramming system, how would you implement priority scheduling without having the scheduler search through all the processes in the ready queue looking for the highest priority process to run next?

   Have a different ready queue for each priority level. The scheduler would choose the next process from the head of the highest priority non-empty queue.

3. (a) Describe briefly what happens on a CPU when an interrupt occurs. That is, what actions does the hardware perform (not the OS)?

   The hardware will 1) save some registers, such as the IP, PSW (aka Flags) register, and SP, 2) switch to kernel mode and 3) jump to the interrupt handler whose address is found at the place in the interrupt table corresponding to the interrupt number.

   (b) In your programming assignment, it was assumed that when a keyboard interrupt occurred there was a proces that was blocked, waiting for input from a keyboard. In the real world, of course, a user can type on the keyboard even if no process is waiting for the input. Describe how you think an operating system actually handles keyboard interrupts in the real world. Be specific on the actions that an OS would take when a keyboard interrupt occurs.

   When the keyboard interrupt occurs, if no process is blocked waiting for input from the keyboard, then the keyboard input is placed in a buffer and the current process can continue executing. The next time a process requests input from the keyboard, the OS will check to see if there is input already in the buffer. If so, the request can be satisfied immediately and the requesting process can continue running. Otherwise, the process will have to block.

4. Suppose you were programming on a system that didn't support semaphores but did support a test-and-set instruction. Suppose further that there was a C library procedure

```
test_and_set(int *val, int *lock);
```

that atomically writes the contents of the location that `lock` points to into the location that `val` points to and writes a 1 into the location that `lock` points to.

Write some (short!) C code for two processes that have a shared data structure that should only be accessed by one process at a time (i.e. requires mutual exclusion) and that use the `test_and_set()` procedure to ensure mutual exclusion. You can use any other shared variable that you want - assume that any global variable used in both processes is shared. Be sure to write as correct C code as possible. It doesn't have to perform a meaningful computation, it just has to show that you know how to write code that requires mutual exclusion and how to implement mutual exclusion using `test_and_set()`.

Here is a simple example. The shared data is:

```
//Assume the following variables are shared
int A[100];
int index = 0;
int lock = 0;
```

To keep it simple, both processes call the same procedure:

```
//Here is a procedure that both processes execute
void run()
{ int val;  //local, not shared
  int x = 0; //also not shared
  while (x < 50) {
    do {
      test_and_set(&val, &lock);  //busy wait until val = 0
    } while (val != 0);
    //entering critical section
    A[index] = x;
    index++;
    //leaving critical section
    lock = 0;
    x++;
  } //end while
}
```

The code for process 1 would be

```
//Process 1
void Process1()
{
   run();
}
```

and the code for process 2 would be

```
//Process 2
void Process2()
{
   run();
}
```

5. Assume you have a 16-bit computer (i.e. addresses are 16 bits) supporting virtual memory where the page size is 4KB.

   (a) How many elements in a page table would there have to be?
   Since the size of the address space is 2^{16} bytes and the size of the page is 2^{12} bytes, the number of pages in the address space – which is the number of elements needed in the page table – would be $2^{16}/2^{12} = 2^4 = 16$.

   (b) How would a virtual address be partitioned into page number and offset (i.e. how many bits in each part)?
   The upper 4 bits would be used as the page number and the lower 12 bits would be used as the offset into the page

   (c) Suppose the MMU translated the virtual addresses issued by a program into physical addresses as follows (all address shown in hexidecimal):

   3E1F → 7E1F
   5454 → 9454
   2A8D → 6A8D
   E127 → 2127
   C8BA → 08BA
   9762 → D762
   72CF → B2CF

Draw the entire page table, leaving blank those elements that cannot be determined from the information given. Ignore the present/absent bit, dirty bit, etc.

Since the upper four bits corresponds to a single hex digit, there's no need in this case to even think about the binary representation of the address. The first hex digit in the virtual address is the page number and gets replaced by the hex digit representing the page frame in the physical address. The page table is simply an array of 16 elements containing the page frame number along with the various bits (present/absent, referenced, dirty, etc. that are ignored in this question). The virtual page number does not appear in the page table – the page number is used as the index into the page table.

misc. bits	**page frame**
...	
...	
...	**6**
...	**7**
...	
...	**9**
...	
...	**B**
...	
...	**D**
...	
...	
...	**0**
...	
...	**2**
...	

(d) How would the code that your wrote for the first programming assignment be different if one of the traps you had to handle was a page fault? Describe what your trap handler would have done in that case.

The trap handler (typically called `handle_trap()` in the assignment) would have had to handle the page fault trap. It would have to issue a disk read request to fetch the disk block containing the desired page, change the status of the current process to BLOCKED, and choose a process on the ready queue to run.