1. **True/False.** Circle the appropriate choice on this sheet. There are no trick questions.

   (a) T F A disk interrupt is generated when the disk controller has satisfied a disk read request.
   
   Answer: T

   (b) T F The TLB is a cache used to store the most recent data retrieved from RAM.
   
   Answer: F

   (c) T F A blocked process may be unblocked and put onto the ready queue when its quantum expires.
   
   Answer: F

   (d) T F A thread implemented in user space does not rely on being scheduled by the OS.
   
   Answer: T

   (e) T F One goal of process scheduling, for efficiency, is to allow two processes to be in a critical section for the same shared structure at the same time.
   
   Answer: F

   (f) T F The goal of batch scheduling (such as shortest job first) is to reduce the response time for interactive processes, such as word processing programs.
   
   Answer: F

   (g) T F An application program is a program that performs a function for a user, as opposed to providing services required by the computer.
   
   Answer: T

   (h) T F A daemon is a process that runs in the background to provide a service required by the computer.
   
   Answer: T

   (i) T F A separate page table is required for each running thread.
   
   Answer: F

   (j) T F Assuming a fixed address space size, the page table size in a virtual memory system grows inversely with the page size.
   
   Answer: T

2. Fill in the answers on this sheet

   (a) \( \frac{3}{2^{20}} + \frac{1}{2^{17}} = \) ________ (to add fractions, the denominators have to be the same).
   
   Answer: \( \frac{3}{2^{20}} + \frac{1}{2^{17}} = \frac{3}{2^{20}} + (\frac{2^{3}}{2^{20}}) = \frac{3}{2^{20}} + \frac{8}{2^{20}} = \frac{35}{2^{20}} \)

   (b) A _____-bit address is required to reference all the bytes in a 16GB memory.
   
   Answer: 34

   (c) \( 2^{20} = \) ________ (you can leave the answer in terms of K, M, etc.)
   
   Answer: 512M
3. Answer these questions in the blue book

(a) Describe how the TSL (test-and-set-lock) instruction works and give an example of its use.

The TSL instruction reads the value of a flag (e.g. bit, register, or variable) and sets the value of the flag to 1, in a single atomic (indivisible) operation. An example of the use of TSL is (in generic assembly code):

```
enter_critical_section:
    TSL R, flag //write the value of flag into R and set flag = 1
    cmp R,0 //compare R (the old value of flag) to 0
    jne enter_critical_section //if it wasn't zero, then try again
```

(b) Suppose several processes are sharing a stack data structure that support push() and pop() operations (for integers). The data structure is implemented as a fixed-size integer array, A, and an index, top, into the array that defines the top of the stack (and whose value is initially 0). Write the code for push() and pop() that (1) prevents race condition bugs, (2) causes a process that tries to perform a pop on an empty stack to block, and (3) prevents stack overflow by causing a process that tries to perform a push on a full stack to block. You can use any IPC mechanism for this that we have studied in this class, be sure to state any assumptions and any initialization that is needed.

This is just the Producer/Consumer problem discussed in class. To prevent race condition bugs, we will use a binary semaphore, m, to enforce mutual exclusion on the stack structure. To block a process that attempts to perform a pop on an empty stack, we will use a counting semaphore, s_pop, whose value is initially zero. To block a process that attempts to perform a push on a full stack, we will use a counting semaphore, s_push, whose value is initially the size of the array. The code for push() and pop() would be as follows.

```
int pop()
{
    int x;
    down(s_pop);    // block a push
    down(m);
    A[top] = y;
    top--;          // update top
    x = A[top];     // return the popped value
    up(m);          // unblock a push
    up(s_pop);
    return x;
}

void push(int y)
{
    down(s_push);  // block a pop
    down(m);
    top++;         // update top
    A[top] = y;
    up(m);         // unblock a pop
    up(s_push);    // unblock a push
}
```

(c) When implementing a round robin scheduler, what are the advantages and disadvantages of choosing a long quantum time over a short one? Give a brief answer.

The advantage of longer quantum is higher CPU efficiency, because the fraction of time spent performing context switches between processes is lower. The disadvantage of a longer quantum time is that an interactive process may have to wait longer for the CPU (since every process gets a longer turn), potentially making the interactive program appear sluggish to the user.
4. Answer these questions in the blue book

(a) In a swapping memory manager, where each process in memory must be in one contiguous block, briefly explain why the Best-Fit method for determining the placement of a process being swapped in from disk will often perform more poorly than the First-Fit method.

(b) Suppose you have a 16-bit computer (i.e. addresses are 16 bits) that supports virtual memory, where the page size is 8KB. Suppose also that a process is running, for which the page table looks like:

| 1 ... 7 |
| 0 ... 1 |
| 1 ... 4 |
| 1 ... 1 |
| 0 ... 2 |
| 0 ... 3 |
| 0 ... 0 |
| 1 ... 3 |

where the leftmost column is the present/absent bit, the rightmost column is the page frame number, and the first entry of the table appears in the top row of the drawing.

Furthermore, suppose the TLB looks like:

| 1 ... 3 | 1 |
| 0 ... 2 | 5 |
| 1 ... 7 | 3 |

where the leftmost column is the valid bit, and the rightmost two columns (from left to right) are the page number and the page frame number.

i. If the running process issues the virtual address 7A18 (expressed here in hex), what is the corresponding physical address (in hex) generated by the MMU? Consider the individual bits carefully and show your work.

Since the page size is 8KB = 2^{13} bytes, the rightmost \( \log(2^{13}) = 13 \) bits of an address are used as the offset into a page. Thus, the remaining 3 leftmost bits of a virtual address specify the page number. The virtual address 7A18 hex corresponds to 0111 1010 0001 1000 in binary, so the leftmost three bits are 011 binary = 3 and the offset bits are 1 1010 0001 1000. Since the first entry in the TLB is valid (the valid bit is 1) and indicates that page 3 is found in page frame 5, a TLB hit occurs and the leftmost three bits of the physical address would be 5 = 101 binary. The entire physical address is constructed from the page frame number, 101, and the offset, 1 1010 0001 1000, so the result is 1011 1010 0001 1000 binary = BA18 hex.

ii. In the translation process for the above virtual address, was there a TLB miss? Explain.
No. As explained above, a TLB hit resulted because the first entry of the TLB was valid and contained the specified page number.
iii. In the translation process for the above virtual address, was there a page fault? Explain.
   No. Since there was a TLB hit, the page table didn’t even need to be accessed and no page fault resulted.

iv. Give a virtual address for which there would be both a TLB miss and a page fault. Explain.
   Any virtual address whose page number, i.e. the leftmost three bits, are not found in a valid TLB entry nor indicated as being present in the page table would suffice. For example, since there is no TLB entry for page 1 and the present/absent bit for page 1 in the page table indicates that page 1 is not in memory, a virtual address within page 1, i.e. whose leftmost 3 bits are 001, would result in both a TLB miss and a page fault. For example, a virtual address in page 1 is 0010 0000 0000 0000 = 2000 hex.

v. Give a virtual address for which there would be a TLB miss, but not a page fault, and show the corresponding physical address. Explain.
   Since page 0 is in memory in page frame 7 and there is no corresponding entry in the TLB, any virtual address in page 0 will result in a TLB miss but not a page fault. For example, the virtual address 0001 1111 1111 1111 = 1FFF hex would be mapped to 1111 1111 1111 1111 = FFFF hex.

(c) How does the OS become involved when a page fault occurs and what does it do to handle the page fault?
   The page fault is a trap, thus the trap handler code is automatically executed by the hardware when the trap occurs. The page fault handling code will block the process that issued the virtual address and issue a disk read request in order to bring the requested page into memory. Once the disk read request is satisfied, as signalled by a disk interrupt, the OS updates the page table to reflect that the page is now in memory and unblocks the process and puts the process on the ready queue.