Problem 1. (5 points)

a. (1 point) Threads
: The main difference is that threads share memory while processes do not share memory.

b.(2 points)
x is12.   y is 2 - not possible
x is 11. y is 1
or
x is 11. y is 1
y is 12. y is 2

b. (2 points)
x is12.   y is 2
x is 13. y is 1
or
x is 13. y is 2
x is 13. y is 1
or
x is 13. y is 1
x is 13. y is 2
or
x is 11. y is 1
x is 13. y is 2

Noted that we have assumed “x=x+y” as an atomic operation.

Problem 2 (5 points).

a.) Adapted from chapter 6 slides 27 - 31.

1 pt - Multiple semaphore use

1 pt - Works correctly

```
Semaphore* prod = new Semaphore( "prod", QUEUE_SIZE );
Semaphore* cons = new Semaphore( "cons", 0 );

// P is dec, V is inc, locked is 0

// Consumer:
    while( 1 ) {
       cons->P();
       Consume next data item;
       prod->V();
    }

// Producer:
    while( 1 ) {
       prod->P();
       Produce next data item:
       cons->V();
    }
```

b.) Adapted from chapter 6 slide 36.

½ pt - Producer locks

½ pt - Consumer locks

1 pt - Consumer waits on empty queue

1 pt - Producer calls signal (1/2 pt if broadcast was used. Broadcast is unnecessary in this context and far more expensive than signal)

```
Lock* lock = new Lock( "Lock" );
Conditional* cond = new Conditional( "Conditional" );
int s = 0;

// Consumer:
      while( 1 ) {
         lock->Acquire();
         if ( s <= 0 ) {
              cond->Wait( lock );
         }
         Consume next data item;
         s--;
         lock->Release();
      }

// Producer:
      while( 1 ) {
         lock->Acquire();
         Produce next data item:
         s++;
         cond->Signal( lock );
         lock->Release();
      }
```

Problem 3 (5 points).

a. (2 points). TLB speeds up the translation of logical page to physical page. Since TLB is an extremely fast cache, we can find the physical page of a local page much fast than fetching this information from an in-memory page table if available.

TLB hit ratio is typically high because a program often contains loops and accesses consecutive addresses. As a result the same page will be visited repetitively. Even there is a miss at the first time when a page is visited, there is a TLB hit when this page is visited again immediately.

b. (3 points) TLB access = e = 10[ns]; Memory access = 100[ns] ;

Effective access time is   p(100+10) +(1-p)(200+10);

This number <= 220 ns. Thus   p >= 90[%]

Problem 4 (5 points).

a. (1 pt). Internal fragmentation results from the allocation of an entire page to satisfy a memory request of less than an entire page. Under such circumstances, the remainder of the page has been allocated to the process but is unused, and is thus wasted.

b.) Virtual address (64 bits): [26 bits | 12 bits | 12 bits | 14 bits]

1 pt - page offset is 14 bits

1 pt - middle portion is 12 bits

1 pt - max space (2^12 * 2^12 * 2^14 = 2^38)

½ pt - Correct explanation of paging design. Diagrams illustrating how it would work were acceptable, even if the numbers are incorrect

½ pt - Correct explanation of max space

Problem 5 (5 points).

a.(3 point) Best-fit: use partition of size 100K

First-fit: use partition of size 200K

b. (2 points)

External memory allocation is when small pieces of memory are scattered and can't be used to execute a job even the summation of these small pieces is big enough to execute this job.

From the example above, assume a job is asking for 400KB, even the total free space added together exceeds this number, there is no consecutive space available for executing such a job.

Problem 6 (5 points).

a.) (3 pts)

1 pt - Combination of filesystem->Open and AddrSpace. These create a new address space for the process and load the text, data, and BSS segments of the given file into the new address space. The input file is assumed to be in NACHOS object format (NOFF).

½ pt - InitRegisters(). This line 1) initializes the program counter to where the text segment starts, 2) sets the stack pointer register to where the stack is in the address space, and 3) initializes all other registers to 0. Must mention at least one for a correct answer.

1 pt - RestoreState(). Sets the active page table to be that of the process being created.

½ pt - Run(). Switches NACHOS to user mode and starts executing the process.

b.)

1 pt - StackAllocate. Creates a new stack for the thread. Also sets the thread's program counter to execute the function when it gets run (1/2 pt), and saves the argument in the thread's registers (1/2 pt). If only mentioned that this creates a new stack, only ½ pt is awarded.

½ pt - ReadyToRun. Puts the thread on the end of the scheduler's run queue, and marks it as runnable.

½ pt - interrupt->SetLevel. As a pair, the calls turn interrupts on and off, respectively. ReadyToRun assumes interrupts are off for correctness.