Sample Questions

PC Hardware, Assembly Language, System Calls

****Question 1****

1. On x86, if the `EAX` register holds the value 0x712ab211, what value does the `AH` register have?

2. In the following assembly program, what is the value of the `EAX` register when the `done` label is reached?

```
        start:
            mov $0, %eax
            jmp two

        one:
            mov $1, %eax

        two:
            cmp %eax, $1
            je done
            call one
            mov $10, %eax

        done:
            jmp done
```

****Question 2****

Below is the code for `fetchint` and `argint` in xv6:

```
    // Fetch the int at addr from the current process.
    int
    fetchint(uint addr, int *ip)
    {
      if(addr >= proc->sz || addr+4 > proc->sz)
        return -1;
      *ip = *(int*)(addr);
      return 0;
    }

    // Fetch the nth 32-bit system call argument.
```

```
int
argint(int n, int *ip)
{
  return fetchint(proc->tf->esp + 4 + 4*n, ip);
}
```

Suppose we removed the check `(addr >= proc->sz || addr+4 > proc->sz)` (which, as you will recall, is there to guard against malicious user-space programs trying to crash the kernel or read memory they're not supposed to). Now, finish the following snippet of a malicious user-space program written in assembly so that it will crash the xv6 kernel:

```
// Your code here
mov    $0x6, %eax   ; kill(int pid) is system call 6
int    $0x40        ; execute system call interrupt
```

****Question 3****

Using the xv6 system calls below, write C code that creates a file named `hello.txt` and puts the string `hello world` into it. You do not need to write out the include statements or even a proper main function; just include the operations needed to open, write to, and close the file.

```
#define O_RDONLY  0x000
#define O_WRONLY  0x001
#define O_RDWR    0x002
#define O_CREATE  0x200

int open(char *filename, int mode);
int write(int fd, void *buf, int sz);
int close(int fd);
```

Memory Management and Virtual Memory

****Question 1****

(Tanenbaum Ch 3, Problem 5)

What is the difference between a physical address and a virtual address?

****Question 2**
******

(Tanenbaum Ch 3, Problem 4)

Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 MB, 4 MB, 20 MB, 18 MB, 7 MB, 9 MB, 12 MB, and 15 MB. Which hole is taken for successive segment requests of:

* 12 MB

* 10 MB

* 9 MB

for first fit? Now repeat the question for best fit.

****Question 3 ****

Why is the principle of locality crucial to the use of virtual memory?

****Question 4 ****

What does TLB stand for and what is it's purpose?

****Question 5 ****

Consider the following C program:

```
Int X[N];
Int step = M;  /* M is some constant */
For (int i=0; I < N; i+= step)  X[i] = X[i] + 1;
```

a) If this program is run on a machine with with a 4KB page size and 64 entry TLB, what values of M and N will cause a TLB miss for every execution of the inner loop?
b) Would your answer in part (a) be different if the loop were repeated many times? Explain.

Processes, Threads, and Scheduling

****Question 1****

(Tanenbaum Ch 2, Problem 40)

Round-robin schedulers normally maintain a list of all runnable processes, with each process occurring exactly once in the list. What would happen if a process occurred twice in the list? Can you think of any reason for allowing this?

****Question 2****

1. (Tanenbaum Ch 2, Problem 14) The register set is generally considered to be a per-thread rather than a per-process item. Why? After all, the machine has only one set of registers.

2. (Tanenbaum Ch 2, Problem 15) Why would a thread ever voluntarily give up the CPU by calling *thread_yield*? After all, since there is no periodic clock interrupt, it may never get the CPU back.

****Question 3****

Describe the conditions that need to occur for a *priority inversion* bug to happen.

Drivers and I/O

****Question 1****

What problem does double buffering solve?

****Question 2****

Suppose a printer prints one character at a time, and issues an interrupt when it is ready to print another. An interrupt handler for this device might look like:

```
    // count: total bytes to be printed
    // p: the data buffer containing data to print
    // i: the index of the next byte to be sent to the printer
    if (count == 0) {
        unblock_user();
```

```
    } else {
        *printer_data_register = p[i];
        count = count – 1;
        i = i + 1;
    }
    acknowledge_interrupt();
    return_from_interrupt();
```

In this code, the interrupt is not acknowledged until after the next character has been output to the printer. Could it have equally well been acknowledged right at the start of the interrupt service procedure? If so, give one reason for doing it at the end. If not, why not?

****Question 3****

The rate at which a 300 dpi scanner produces data is 1 MB/sec. An 802.11b wireless network has a maximum transmission rate of 900KB/s. Can documents be sent out on the network as fast as they are scanned? Why or why not?

Concurrency

****Question 1****

Suppose that we have an atomic compare-and-swap instruction that atomically compares a variable with some value and swaps them if they are not equal:

```
    int compare_and_swap(int *var, int val);
```

Write implementations of `void acquire(int *lock)` and `void release(int *lock)` that use this instruction to implement a *spin lock* (that is, a lock that loops until it is able to acquire exclusive access to the lock). Note that we will assume here that each lock is represented by a global integer variable.

```
    void acquire (int *lock) {
        // your code here

    }

    void release (int *lock) {
        // your code here

    }
```

****Question 2****

Recall the parallel hashtable implementation from Homework 4:

```
#define NUM_BUCKETS 5     // Buckets in hash table

typedef struct _bucket_entry {
    int key;
    int val;
    struct _bucket_entry *next;
} bucket_entry;

bucket_entry *table[NUM_BUCKETS];

// Inserts a key-value pair into the table
void insert(int key, int val) {
    int i = key % NUM_BUCKETS;
    bucket_entry *e = (bucket_entry *) malloc(sizeof(bucket_entry));
    if (!e) panic("No memory to allocate bucket!");
    e->next = table[i];
    e->key = key;
    e->val = val;
    table[i] = e;
}
```

Suppose we have two threads inserting keys with `insert()` at the same time. Describe the exact sequence of events that results in a key getting lost.

****Question 3****

Consider the following allocation and request matrices, where E is the vector representing the resources of each type that exist in the system and A is the vector representing the resources currently available.

```
E = (3, 5, 4)        A = (0, 4, 2)

     Current
Allocation Matrix    Request Matrix
   [ 1  0  1 ]         [ 0  2  0 ]
   [ 1  1  1 ]         [ 2  0  0 ]
   [ 1  0  0 ]         [ 1  1  4 ]
```

Is this system deadlocked? (Show how you arrived at that answer)

Filesystems

Question 1

Suppose we have a non-journaled filesystem that uses i-nodes, and a file delete operation that consists of the following actions:

1. Mark the i-node for the file as free in the filesystem bitmap.
2. Mark the data blocks for the file as free in the filesystem bitmap.
3. Remove the directory entry for the file from the directory.

Now suppose that we have a crash after step 2.

1. Describe a scenario where this results in file data being corrupted.
2. How would a filesystem checker like `fsck` that runs at boot detect and fix this condition?

Question 2

In the xv6 logging filesystem, filesystem operations are grouped into transactions, where each transaction consists of the following operations:

1. Write each modified block to the log area, along with its eventual destination.
2. Write a commit record.
3. For each entry in the log, copy the block to its final destination.
4. Clear the log.

For each of these steps, describe what would happen if the system crashed during that step, saying what xv6 would do when it reboots and how this would guarantee that the transaction is carried out atomically (that is, every operation is carried out, or none of them are).

Question 3

Suppose we have a filesystem with a block size of 512 bytes and an i-node defined as follows:

```
    #define BLOCKSIZE 512

    struct inode {
        short type;           // File type
        short major;          // Major device number (T_DEV only)
        short minor;          // Minor device number (T_DEV only)
        short nlink;          // Number of links to inode in file system
        uint size;            // Size of file (bytes)
        uint blocks[32];
        uint indirect;
    };
```

That is, it has 32 direct block pointers and one indirect block pointer. What size (in bytes) is the largest file we can create using this system?

Security

****Question 1****

Consider the following program:

```
    int main(int argc, char **argv) {
        char magic[4];
        int winner = 0;

        // Copy command line input into magic var
        strcpy(magic,argv[1]);
        // Do secret computation to check for magic value
        if (((magic[0] * 0x2115) + (magic[1] * 1222) ^ (magic[2] << 3)) ==
0xbeef)
            winner = 1;

        if (winner) printf ("You win!\n");
        else printf("You lose\n");

        return 0;
    }
```

When run, the stack layout for the `main()` function looks like:

```
0x1000 magic[4]
0x1004 winner
0x1008 saved EBP
0x100c return address
```

1. This program has a buffer overflow. Find it, and use it to give an input (i.e. a value for `argv[1]`) that will cause the program to print "You win!".

2. Which of the following (if any) would prevent the problem from being exploited?

   * DEP (Data Execution Prevention)

   * ASLR (Address Space Layour Randomization)

   * Stack canaries

****Question 2****

1. Explain how adding a *salt* to a password makes password cracking more difficult.

2. Why would we want a password hashing algorithm to be slow?