CSCI3240 Exam 2 Study Guide

Chapter 3 Practice problem.

(Problem 1-10 are practice problem in the book.)

1. Practice problem 3.10
2. Practice problem 3.11
3. Practice problem 3.18
4. Practice problem 3.20
5. Practice problem 3.24
6. Practice problem 3.26
7. Practice problem 3.36
8. Practice problem 3.38
9. Practice problem 3.41 (A and B)
10. **Struct Alignment:** Consider the following C struct declaration:

```
typedef struct {
     char a;
     long b;
     float c;
     char d[3];
     int *e;
     short *f;
     } foo;
```

a) Show how foo would be allocated in memory on an x86-64 Linux system. Label the bytes with the names of the various fields and clearly mark the end of the struct. Use an X to denote space that is allocated in the struct as padding. (Each cell represents 1-byte location).

b) Rearrange the elements of foo to conserve the most space in memory. Label the bytes with the names of the various fields and clearly mark the end of the struct. Use an X to denote space that is allocated in the struct as padding. (Each cell represents 1-byte location).

11. **Struct Access:** Consider the following C structure declaration:

```
struct my_struct{
     char a;
     long b;
     short c;
     float *d[2];
     unsigned char e[3];
     float f;
};
```

a) Show how `my_struct` would be allocated in memory on an x86-64 Linux system. Label the bytes with the names of the various fields and clearly mark the end of the struct. Use an X to denote space that is allocated in the struct as padding. (Each cell represents 1-byte location).

b) How many bytes is the smallest possible struct containing the same elements as my_struct?

12. Pointers and Array

Declaration	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
`int A1[3]`									
`int *A2[3]`									
`int (*A3)[3]`									
`int (*A4[3])`									

Declaration	An			*An			**An			***An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
int A1[3][5]												
int *A2[3][5]												
int (*A3)[3][5]												
int *(A4[3][5])												
int (*A5[3])[5]												

- **Cmp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by `sizeof`**

13. What is the difference between virus and worm?

14. Explain buffer overflow attacks. How do we avoid them?

Chapter 5: Optimizing Program Performance

(Problem 1-2 are practice problem in the book.)

1. Practice Problem 5.2
2. Practice Problem 5.3
3. What are the generally useful optimization techniques in C?
4. Why is Loop Unrolling technique effective?
5. Discuss two optimization blockers.
6. Write the optimized versions of the following C codes:

a.

```
void myfunction(double *a, double *b, long i, long n){
    long j;
    for (j = 0; j < n; j++)
      a[n*i+j] = b[i];
}
```

b.

```
for (int i = 0; i < n; i++) {
  int ni = n*i;
  for (j = 0; j < n; j++)
    a[ni + j] = b[2];
}
```

c.

```
void myFunction (int *a){
for (int i=0; i< getLength(a); i++){
    a[i] = i + myfunction2(a[0]);
}
```

Chapter 11: Network Programming

1. Explain the difference between client and the server. Provide examples of some well known servers.
2. What does an internet protocol do?
3. What is LAN?
4. Convert the following 32-bit addresses into dotted decimal notation:
   a. 0xff3240ff
   b. 0x00ff1234
5. Convert the following addresses in dotted decimal notion to 32-bit hex form.
   a. 129.184.254.200
   b. 253.183.199.130
6. Explain Domain Name System. Why is it used?
7. Explain Socket Interface. What are the functions involved in the client and the server side? Briefly describe the job of each function.

Chapter 12: Practice Problems

8. What are the possible output sequences from the following program:

A. Circle the possible output sequences:

- abc
- acb
- bac
- bca
- cab
- cba

```
int main() {
    if (fork() == 0) {
        printf("a");
        exit(0);
    }
    else {
        printf("b");
        waitpid(-1, NULL, 0);
    }
    printf("c");
    exit(0);
}
```

9. Consider the following code sample.

```
int global_x = 0;
int main(int argc, char *argv[]){
    global_x = 17;
    /* Assume fork never fails */
    if(!fork()) {
        global_x++;
        printf("Child: %d\n", global_x);
    }
    else {
        wait(NULL);
        global_x--;
        printf("Parent: %d\n",
global_x);
    }
    return 0;
}
```

A. What is printed by this program?

B. How might the output change if we remove the call to wait?

10. Using the following code, fill each entry in the table with “Yes” or “No”. Note: header removed to save space.

```
int g = 0;
// The function to be executed by all threads
void *myThreadFun(void *vargp) {
    int *myid = (int *)vargp;
    static int s = 0;
    ++s; ++g;
    printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}
int main(){
    int i;
    pthread_t tid;
    for (i = 0; i < 3; i++)
        pthread_create(&tid, NULL, myThreadFun, (void *)&i);

    pthread_exit(NULL);
    return 0;
}
```

Variable Instance	Referenced by main thread?	Referenced by peer thread 0?	Referenced by peer thread 1?	Referenced by peer thread 2?
g				
s				
i.m				
tid.m				
myid.p0				
myid.p1				
myid.p2				

tid.m implies local variable declared in main function.
myid.p0 implies myid local variable declared in stack of peer thread 0
myid.p1 implies myid local variable declared in stack of peer thread 1
myid.p2 implies myid local variable declared in stack of peer thread 2

11. Consider the C program below. For space reasons, we are not checking error return codes, so assume that all functions return normally.

```c
int main () {
    if (fork() == 0) {
        if (fork() == 0) {
            printf("9");
            exit(1);
        }
        else
        printf("5");
    }
    else {
        pid_t pid;
        if ((pid = wait(NULL)) > 0) {
            printf("3");
        }
    }
    printf("0");
    return 0;
}
```

Circle the string that can be a possible output of the program.
93050 53090 50930 39500 59300

12. Consider the C program below. For space reasons, we are not checking error return codes, so assume that all functions return normally.

What are the outputs?

Parent: i = ________________________

Child: i = ________________________

```c
int i = 0;
int main () {
    int j;
    pid_t pid;
    if ((pid = fork()) == 0) {
        for (j = 0; j < 20; j++)
            i++;
        }
    else {
        wait(NULL);
        i = -1;
    }
    if (i < 0)
        i = 10;
    if (pid > 0)
        printf("Parent: i = %d\n", i);
    else
        printf("Child: i = %d\n", i);
    exit(0);
}
```

13. Define the following terms:

a. deadlock
b. race condition
c. starvation

14. Which of the following is true about races?

(a) A race occurs when correctness of the program depends on one thread reaching point "a" before another thread reaches point "b".
(b) Exclusive access to all shared resources eliminates race conditions.
(c) Race conditions are the same as deadlocks.
(d) All race conditions occur inside loops, since that is the only way we can interleave processes