

5/3/1/0 1. What value does **fopen()** return if the file does not exist?

2. Match the file open mode to the description:

10/8/4/0

r ___

(a) Open file for writing, create the file if it doesn't exist or truncate it if it does

r+ ___

(b) Open the file for reading and writing, reading occurs at the start of the file, writing occurs at the end of the file

w ___

(c) Open the file for writing and reading, create the file if it doesn't exist or truncate it if it does

a+ ___

(d) Open file reading reading and writing, do **not** create the file if it doesn't exist and do **not** truncate it if it does.

w+ ___

(e) Open the file reading, do **not** create the file if it doesn't exist and do **not** truncate it if it does.

10/8/4/0

3. Label all the things wrong with this program below and describe to the right: (**hint: Don't forget about error checking**)

```
#include <stdlib.h>

int main( int argc, char * argv[]){
    file * stream;
    stream = open("file.txt", "r");
    fprintf(stream, "Hello World");
    return 0;
}
```

a

b

c

4. Write the corrected code from question 3:

5/3/1/0

5. Consider the type below, fill in the **fwrite()** statement to write that type to a file:

```
typedef struct{
    long acctnum;
    double bal;
    char acctname[1024];
} acct_t;
```

```
int main(int argc, char *argv[]){
```

```
    acct_t acc;
    acct.acctnum=123456789011;
    bal=1000000000000; //I'm rich!
    strcpy(acctname, "Adam Aviv");
```

```
    FILE * out = open("acct.dat", "w");
```

10/8/4/0

```
    fwrite(  );
```

```
    fclose(out);
}
```

6. Consider a file, **accts.dat**, which stores 1000 accounts formatted like above. Complete the **fread()** command to read all those accounts in:

```
int main(int argc, char * argv[]){
```

```
    acct_t accts[1000];
```

```
    FILE * in = open("accts.dat", "r");
```

10/8/4/0

```
    fread(  );
```

```
    int i;
    for(i=0;i<1000;i++){
        printf("%1 (%f) -- %s\n", accts[i].acctnum,
            accts[i].bal,
            accts[i].acctname);
    }
```

```
    fclose(in);
```

```
}
```

5/3/1/0

7. Explain how the OS provides abstraction and isolation via the System Call API.

5/3/1/0

8. Match the OS system resource to the action. (match all that apply)

- | | | |
|------------------------|-------|--------------------------------------|
| Device Management (1) | _____ | Writing to a file |
| Process Management (2) | _____ | Reading user input from the terminal |
| Memory Management (3) | _____ | Adjusting the break point |
| File Management (4) | _____ | Executing a program |

5/3/1/0

9. Why are certain operations in an OS *privileged*? What is the Operating System protecting us from?

5/3/1/0

10. What is the kernel? And why must it be trusted?

5/3/1/0

11. What section of the man pages are system call found and in and what sections are library functions in?

5/3/1/0 12. Open the manual page for **read()** and **fread()** (), which is the system call and which is the library function? How did you determine this?

5/3/1/0 13. What is the difference between **malloc()** and **sbrk()** from a system programmer perspective? Why is one a system call and one a library function? (APUE discusses this)

5/3/1/0 14. Explain a *context switch* with respect to the kernel-space, user-space and system calls.

5/3/1/0 15. What is a **trap**? How does it relate to context switching?

5/3/1/0 16. Find the man page for the system call **open()**, what is the man command you need to access it? Explain why you can't just type **man open**?