What you really want to know

Those concerned about the Comp 190 workload (and grades) should find this handout interesting reading. It should give you a pretty good idea about the instructor's expectations.

First, you will find copies of last year's first midterm. Second and third, you'll find a copy of the second and fifth homework assignments from last year. Right now, the exams questions and homework assignment may make no sense to you. Don't worry, in a couple of months you'll understand it.

By the way, the undergraduate course grades for Comp 190 in Spring 1989 were 7 A's, 12 B's, 3 C's, 1 D, and 1 F and in Spring 1988 were 11 A's, 12 B's, 3 C's, and 2 F/AB's.

OLD Midterm 1–February 24. 1989

Closed book section (64 points)

The exam is to be turned in at 2:50 pm. Work the closed book section first and turn it in before you consult your books and notes to work on the open book section. For the closed book section, write your answers on the exam itself. For the open book section, write your answers on separate pieces of paper.

University regulations require that you sign the following pledge on the first page of your turned-in exam.

I have neither received nor given any unauthorized aid on this exam. ____________

Problem 1. (24 points–4 points each)

Give short definitions (one or two phrases or sentences) of the following terms.

critical region

delayed write

locking

path name

reference count

user mode

Problem 2. (16 points–4 points each)

Give a brief description of what the following UNIX system calls do at the user level. *You don't need to describe the implementation!*

`chmod`(filename, mode)

`fork`()

`link`(filename1, filename2)

`pipe`(fdptr) *–don't worry about which end is which*

Problem 3. (4 points)

What are the two different Unix operating systems often mentioned in class?

Problem 4. (4 points)

Suppose a C program starting with the following header:

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

is compiled and the compiled code is stored in the file `now`. If the command:

```
% now is the time
```

is executed, what are the values of `argc` and the array elements of `argv`?

Problem 5. (4 points)

Suppose `P[0]` is a file descriptor that refers to the read end of a pipe. Under what circumstances will the system call

```
read(P[0], buff, buff_size)
```

return zero?

Problem 6. (4 points)

List the following six items in order of size (smallest to largest):

a. cylinder group	d. data block
b. track	e. cylinder
c. file system	f. disk

Problem 7. (4 points)

Pick a random *data* block of a file system. If the file system is *consistent,* what is the largest number of inodes that *could* point to that block?

Problem 8. (4 points)

Suppose the file `/foo/onu` has file access permissions 0666, *i.e.,* readable and writable to all users. Under what circumstances will the system call

```
open("/foo/onu", O_RDONLY)
```

return zero?

old Midterm 1–February 24, 1989

Open book section (36 points)

The exam is to be turned in at 2:50 pm. The closed book section should be turned in before you open your books and notes to work the open book section. For the open book section, write your answers on separate pieces of paper.

Problem 1. (24 points)

I'm logged into `napoleon`. (Honestly, I am). When I type the line:

```
% ls -id /bin /tmp
```

which displays the inode numbers of `/bin` and `/tmp`, `napoleon` responds with the following two lines:

```
110 /bin
  2 /tmp
```

Immediately, I type the line:

```
% ls -i /
```

which displays the inode numbers of all files and subdirectories referenced within the root directory. Among the many lines printed are the following:

```
110 /bin
168 /tmp
```

Why are different inode numbers given for "`/tmp`" in these two cases? Why is only one inode number given for "`/bin`" in both cases?

What data blocks were read to execute the two `ls` commands? Draw how the inodes for `/`, `/bin`, and `/tmp` refer to each other either by direct pointers or by indirect references through directory files and mount table entries.

Problem 2. (12 points)

Assume the following:

- A data block is 4000 bytes long.
- An integer is 4 bytes long.
- An inode can hold 10 direct block references.
- A file 1000000 bytes long has been opened for reading at descriptor 7.
- The system call `lseek(7, 100023, 0)` has just been successfully executed.

What happens when a read of 100 bytes is attempted on file descriptor number 7? You only need explain how the kernel determines which data blocks must be brought into the cache to accomplish the read.

old Homework #2 (10 points)

Due, Monday, February 6, 1989

Write a program `makefiles` which takes two arguments: the first, a file name prefix P, and the second, an integer n, and then creates n files $P.0$ through $P.n-1$ each containing the line:

```
I enjoy programming in C.
```

To complete this assignment you will need to use the system calls `open` and `write` and the C library fucntions `atoi` and `sprintf`.

You will need to use `cc`, the C compiler, to compile your program. By the way,

```
% rm haha.*
```

will remove all files whose names start with `haha.`

Rules of engagement

Turn in a printout of your program.

You must do all the typing required to accomplish this assignment. Any other help is permitted with the following exceptions: (1), you may not copy anyone else's program to complete this assignment, and (2), all notes you bring to the terminal with you must have been written by you.

You may find running your first C program to be painful.

old Homework #5 (20 points)

Due, Friday, March 31, 1989

Your program takes a single argument P.

First, it prints a message stating whether or not P is executable by its owner. (Just assume P is a file readable by you. You don't need to check that.) If is not executable, then stop. Otherwise, continue.

Second, it prints P's magic number and a message stating if the magic number is good or bad. If it is bad, then stop. Otherwise, continue.

Third, it prints out all the names in P's symbol table. (All executable object files (see figure 7.20, page 219) have a section containing a symbol table.)

In order to make life bearable, the file `/unc/brock/home5/read_sym.o` contains a compiled C subroutine `read_symbol` that when called with an integer `f` and a pointer `sym_name` to a character string buffer will:

(1) read the symbol in the executable object file opened at file descriptor `f` beginning at the present offset,
(2) place the ASCII name of that symbol in the buffer `sym_name`,
(3) move the offset of file `f` to the beginning of the next symbol, and
(4) return 0, if successful, and -1, otherwise.

So, all you've got to do is figure out how to seek `f` to the beginning of the first symbol and how many symbols to read. Incidently, `/unc/brock/home5/read_sym.c` contains the source for `read_symbol`.

The program `/unc/brock/home5/home5` is my *compiled* solution. You can run it to get some idea of what your program's output should be. By the way, the C program that produced `home5` was 43 lines long.

Warning

Not much code needs to be written but you've got a major task. You must understand the format of executable object files on `napoleon.` Most of the information you need will come from the manual page for `a.out` and Figure 7.20 (p. 219) of your textbook. Look at both of these carefully before you start.

You are also going to have to become a C guru. You'll need to know how to use structures of included files and how to combine compiled modules with your own code. Hint – I used the following:

```
cc -g -o home5 home5.c read_sym.o |& more
```

to compile my program.

Rules of engagement

Turn in a printout of your program.

You may work in groups of two *except* both members of the team cannot work for the Department of Computer Science *and* team members must split the work, both intellectual and grunt, equally.