University of California, Irvine
Deparment of Computer Science

JUNE EXAMINATIONS 2008

CS 146
Instructor: Wayne Hayes

Duration — 2 hours

Examination Aids: NO AIDS ALLOWED

Your NAME:

Your STUDENT NUMBER:

Total Marks: 87

NOTE: Questions are printed on both sides of the sheet. There are 11 pages on this exam. Please ensure that you have all 11 before you start. **It is your responsibility to do so.** All questions must be answered in the space provided. **BE BRIEF**, especially in the "short answer" questions.

HINTS

— **READ EACH QUESTION CAREFULLY**.

— Write legibly. An unreadable answer recieves no credit.

— Some of the questions are harder than others. If you can't do one, skip it and return to it later.

— Some questions are worth more than others. Manage your time carefully. *Especially, don't waste your time answering more than what is asked.*

— If you can't recall the name of a command or the letter for an option, make it up and say what it's supposed to do. If it exists, you'll probably get full credit.

— If there seems to be ambiguity in the question, state your assumptions and go from there. If your assumptions are reasonable and don't trivialize the problem, you'll probably get full credit.

15. [8 marks]Define the following terms in the context of concurrency. (If you find it hard to write a technical definition, describe it or give an example.)

Race condition

Deadlock

Process starvation

Critical section

16. [6 marks]**(a)** Describe the permission bits that should be set on an RCS directory and the files inside it, if you and your buddy are to share access to an RCS repository. Assume that you both belong to the group `ugrad`.

(b) If you ask the sysadmins nicely, there is a way they can arrange things so that ONLY you and your buddy can access those files, rather than everybody who is group `ugrad`. What question should you nicely ask the sysadmins?

17. [5 marks]Describe how a buffer overflow security hole can be exploited.

18. [5 marks]What is "security through obscurity"? How safe is it? What precisely is the problem with it?

19. [5 marks]If process A on a Unix system has an open file descriptor on file "foo", is it possible to rename the file (eg., using "mv")? Why or why not?

CONT'D...

Part II: longer answers

20. [10 marks]In an early version of Unix, it was the shell's responsibility to look at the first two characters of an executable file before an `exec(2)` to see if it was a binary executable or a "#!" interpreter file (or, if the bytes were unrecognized, to assume it was a Bourne shell script). If the file was an interpreter file, then the shell would read the remainder of the line (to find out what interpreter to use) and construct the proper `exec(2)` system call to call the interpreter on the interpreter file. There are several major disadvantages to having the shell do this, so the functionality was eventually moved into the kernel. Describe one (1) disadvantage.

21. [10 marks]Below is one of the wrong algorithms for two-process mutual exclusion that we covered in class. Give an execution trace (ie., a precise sequence of events) that demostrates its wrongness.

```
/* Process 0 */
id=0;
while (1) {
    want[id]=1;
    while(want[1-id]) ;
    /* critical section */
    want[id] = 0;
    /* remainder section */
}
```

```
/* Process 1 */
id=1;
while(1) {
    want[id]=1;
    while(want[1-id]) ;
    /* critical section */
    want[id] = 0;
    /* remainder section */
}
```

22. [10 marks]Assume that you want to write a program that truly needs about 1.5 gigabytes of workspace, but you are running it on a machine with only 256 megabytes of memory, and assume that you need to access every byte of the 1.5GB of memory at least 100 times (both reading and writing). Thus, some virtual memory paging is certain to occur. What kind of memory access properties would you want your program to have if it is to run quickly and minimize swapping? What kind of memory access properties would cause the program to run very slowly?

23. [15 marks]At any given instant of time, the *process tree* is a kernal data structure that describes the topology of all the fork system calls that were performed to reach the current state (only live processes exist in the process tree; dead ones are not represented—ignore zombies for this question). Each node in the tree represents exactly one live process in the system. Each node has a well-defined parent (which is the process that forked it), and each node can have zero or more children (which are the processes that it forked, that have not yet exited). The tree changes from instant to instant, as new processes are created and destroyed. The root of the tree, of course, is the `init` process, which has a PID of 1.

Write a Bourne shell script called `find-init` that, give the PID of a process on the command line, traces the "lineage" of that process. That is, it will trace the path through the tree from PID back to `init`. It should print out the line from `ps(1)` for each node that it passes, including the one for PID. Below are some hints about which commands you might want to use. Note: PPID is the parent process ID.

```
$ ps # assume it prints all processes of all users
USER   PID  PPID PRI  NI   VSZ  RSS WCHAN  STAT TTY        TIME COMMAND
root     1     0  16   0  1500  516 -      S    ?          0:00 init
wayne  102   101  10   0  3700  213 -      R    pts/12     0:01 bash
[etc... much output deleted, but the lines deleted include those shown below]
$ ps | grep wayne | grep bash | awk '{print $3}'
101
$ find-init 102
USER   PID  PPID PRI  NI   VSZ  RSS WCHAN  STAT TTY        TIME COMMAND
wayne  102   101  10   0  3700  213 -      R    pts/12     0:01 bash
wayne  101   100  20   0  4300  102 -      W    pts/12     0:00 ssh
wayne  100     1  17   0  5200  104 -      W    pts/12     0:00 sshd
root     1     0  16   0  1500  516 -      S    ?          0:00 init
$
```