CS140
Operating Systems and Systems Programming

Final Exam. Summer 2006.
By Adam L Beberg.
Given August 19th, 2006.

Total time = 3 hours, Total Points = 335

Name: (please print)_____

In recognition of and in the spirit of the Stanford University Honor Code, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature:_____

- This exam is closed notes and closed book.
- No collaboration of any kind is permitted.
- You have 3 hours to complete the exam.
- There are 24 questions totaling 335 points. Some questions have multiple parts. Read all parts before answering!
- Please check that you have all 13 pages.
- Before starting, write your initials on each page, in case they become separated during grading.
- Please print or write legibly.
- Answers may not require all the space provided. **Complete but concise answers are encouraged.**
- SCPD students: If you wish to have the exam returned to you at your company, please check the box.

Page 2	/10
Page 3	/30
Page 4	/30
Page 5	/25
Page 6	/25
Page 7	/35
Page 8	/40
Page 9	/35
Page 10	/40
Page 11	/30
Page 12	/35
TOTAL	/335

SCPD?	

INSTRUCTIONS

Near the beginning of the course, if not the first lecture, I said that even the "Hello World" application involved millions of lines of code to make run. Now that the course is over, and with your experience with Pintos, you should understand what all of those lines of code are doing.

This is your chance to demonstrate that you understand operating systems. If you watched the lectures, did the assignment, and read the materials, all of these questions should be easy. This exam is 1/3 of your grade, so show us you know the material. You should have much more time then you need to do a great job.

All questions will refer to the source code on the LAST page of this exam as "the code". Feel free to rip that page off and refer to it, and use it as scratch paper, you do not need to turn that page in (we have a copy).

READ ALL THE QUESTIONS FIRST! Unlike most tests, this is not just a suggestion. In general you're asked what happens when things go smoothly, then about when things go a different way, each questions follows from the one before. If you're not about to be asked about an edge case, or an important issue, go ahead and mention it however.

BEGIN CS 140 FINAL EXAM

1. [10] You take the code and run `` `gcc -c final.c -o final.o` `` to compile the code. What stuff is in the final.o file? Exact names/number are not needed, you did that in the midterm.

2. [10] (a) Next you link final.o along with the standard C library. What does the linker spit out if the C library is a static library - libc.a?

   [5] (b) What unused code will end up in the program, and why?

3. [10] Same as question #2, but with a dynamic stared library - libc.so, how is this program different?

4. [5] List 3 things in the ELF header(s) of the program.

5. [20] (a) Now you run `final`. What does the operating system do up to the point where you add it to the ready list? Assume lazy loading like in the project. (completely ignore the details of the file system for now, we'll get to that)

   [5] (b) And if we weren't so lazy?

6. [5] When you run final, all physical memory is already in use. What happens?

7. [5] List 2 situations where the OS will not let the program run for you at that moment, but would let you if the situation was not happening.

8. [5] (a) Finally everything is going according to plan, you're ready. When do you get to run?

   [5] (b) Why could this be a while?

   [10] (c) If this was a typical type of program on the machines you administered, what scheduling algorithm would you want to use? Justify your choice.

9. [5] (a) Which lines of code *always* generate a systems call? List #'s.

   [5] (b) Which lines of code *might* generate a system call? List #'s.

10. [10] (a) Assume timeslices end right after every line of code by using the Pyschic Timeslice Algorithm™. If we're using an exponential feedback algorithm, what's happening to our priority as we go through the program. Use lower to mean low priority in your description, not a low queue number, and higher to mean higher priority.

1	8
2	9
3	10
4	11
5	12
6	13
7	14

    [5] (b) What if we're billing all time spent in the system call time to the process, would this change your answers? Which ones?

11. [20] (a) Now back to that file system were ignoring. If we're running on the Elaine machines and using an ACL based access control system, what happens when the open() in line 5 is run? Describe all the structures and OS ideas involved. (now you can ignore the syscall details)

[10] (b) What if we were in a Capabilities based system? Also, What happened at login to make this possible?

[5] (c) What if the file isn't openable, what's different between how ACL and Capabilities handle it?

12. [10] Now we're in a typical UNIX file system with inodes. What's going to be happening as far as figuring out where the blocks are as we read (many) blocks due to line 7.

[5] (b) … and with an extent based file system?

13. [10] How would this be different if before line 7 there was a call to mmap().

14. [10] (a) By now we know what disk blocks we need and have a long list, and so do other processes (maybe they have short lists). What do we tell the disk to do so everyone is happy?

[5] (b) What does SCSI do that IDE doesn't that's makes this a non-issue.

15. [10] After we start this program, the system comes to a standstill, and becomes unbearably slow. In what ways could `BIG_NUMBER` be involved in this.

[5] (b) Is this really avoidable at all if `BIG_NUMBER` is really big? How? Your answer should not involve violence ;)

16. [10] (a) Is a log based file system likely to help this processes performance or hurt it or not matter? Why?

[10] (b) What would happen differently in line 7 and in line 9 in the case of a log based file system?

17. [5] How do we battle latency and bandwidth limits in disk systems?

18. [10] (a) Alice is sending Bob a message in a public key cryptosystem. What does she do in which order.

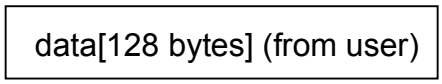
    [5] (b) What does Bob do when he gets the message?

    [5] (c) If Alice and Bob have never talked directly to each other in person or by phone, is this system secure?

19. [10] (a) What happens during the connect() call in line 12?

    [5] (b) If it was a UDP socket instead?

20. [20] (a) Draw a diagram of what the packet sent in line 13 will look like going across the wire. You don't need to remember all the exact header fields, just the ones we highlighted in class. You _DO_ need every extra chunk added between line 13 and the wire. Label each addition as belonging to the end-to-end, network, or link layer. Here is a start: (since the page is only 8.5 inches wide, some arrows may be useful)



21. [10] What 4 packet problems does TCP fix, and what does it use to solve them all.

22. [5] (a) Why do wireless networks normally use encryption in the link-layer, in clear violation of the end to end principle.

    [5] (b) Why _doesn't_ encryption happen in the link later, since we should all obviously be using it for everything now that we're all being watched at all times.

23. [10] (a) Buffer overflows… what stupid thing do programmers keep doing even though it's 100% obvious when you look at the code.

    [5] (b) What did the hardware folks do (or stopped doing for some reason) that made this a problem in the first place?

24. [10] Pick 2 lines of code that happen differently (but act the same of course) if we're in a Virtual Machine Monitor. Briefly, what's different?

END CS 140 FINAL EXAM

Assume during the "..." that any undeclared variables are declared, any variable that need initializing are initialized, and all function calls succeed. Those details are not important.

Line #	final.c
	`int main( int argc, char * argv[] )`
	`{`
1:	`  int file;`
2:	`  int socket;`
3:	`  char packet[128];`
4:	`  void * matrix = NULL;`
	`  ...`
5:	`  file = open( "/tmp/foo.txt", "r+" );`
	`  ...`
6:	`  matrix = malloc( BIG_NUMBER );`
7:	`  read( file, matrix, BIG_NUMBER );`
8:	`  do_matrix_invert( matrix );`
9:	`  write( file, matrix, BIG_NUMBER );`
10:	`  free( matrix );`
	`  ...`
11:	`  socket = socket( PF_INET, SOCK_STREAM, 0 ); /*TCP*/`
12:	`  connect( socket, &address, sizeof(address) );`
13:	`  send( socket, &packet, sizeof(packet), 0 );`
14:	`  len = recv( socket, &packet, sizeof(packet), 0 );`
	`  ...`
	`}`