You are kindly requested to submit your answers in PDF to the 120 assignments below via BlackBoard.

Each answer must consist of an explanation of less than 100 words. Your explanation should demonstrate that you have fully understood both the question and your own answer. To get an idea of what we expect by way of written answers, see the model answers to assignments #4, #12 and #14.

The final mark for the operating systems component of the module will normally equal the number of assignments that have been completed correctly, divided by 10, and capped at 10. There will be no marks for incorrect answers and you cannot resubmit answers a second time.

Student assistants and / or the lecturer will regularly interview you about assignments of their choice so that they can give you feedback on your progress.

You may expect to be interviewed four times (i.e. about once a week during the four weeks that the operating systems course runs).

An interview may take up to an hour, but if you master the material the interview will be short.

The interviewer needs some time to study your work, so you will have to hand in your work a reasonable amount of time before the interview.

During the interview you must be able to show how you obtained your answers. To do this, you will need your laptop and your Pi, in working order. The interviewer will only provide an extension lead. A headless Pi configuration (i.e. your Pi, an Ethernet cable and your laptop) would be best. If you require a network connection, cables or equipment that turns out to be unavailable or malfunctioning during the interview then you may not be given a second chance for the interview.

The interviewer will ask you to explain in depth your answers, asking you many questions beginning with “Why”. This means that you should be well prepared. For example, prior to the interview read the assignments and your answers again. Make sure that you can produce all the data, files, programs, shell scripts, and photos that you made when you worked on your answers.

The interviewer may ask you to compile and run your programs (both on your Pi and on your laptop), or to make small changes to the programs to explore your understanding.

The interviewer may also ask you to explain some of the relevant theory with pencil and paper.

If you do not satisfy the interviewer, for example because you cannot explain why you took certain decisions, then the assumption will be that you have cut some corners and your answer to the particular assignment will not be accepted. Each time an interviewee fails to satisfy the interviewer, the mark will be reduced by 1 point.
You must complete at least the 55 mandatory (= starred) assignments.

If your point total is below the pass mark (55) then you will have to sit the final test and your mark will be determined by the final test. Most of the final test questions will be familiar to you if you have at least tried all 120 assignments.

Even if your point total exceeds 55, you still have the right to sit the final test. In that case, the final mark will be the maximum of the test mark and the assignment mark.

If you disagree with the interviewer then you should contact the lecturer or the module coordinator.

Ten top tips

1. If you have a Windows laptop then you are advised run Linux on it. You have at least three options:
   a. Set your laptop up as a dual boot machine with a Linux partition.
   b. Run Linux in a virtual machine, using VirtualBox or VMware.
   c. Install Cygwin; this is not really Linux, but close enough.
   A Mac does not run Linux either, but MacOS is close enough.

2. If you do not know what a particular Linux command “X” does, try the Linux command “man X”.

3. Many of the assignments require you to search the Internet for information; this takes time and requires an Internet connection.

4. Keep every scrap of information (files, notes, pictures, URLs) and make sure that you know where to find the information for use at the interviews.

5. Treat your Pi with respect so that it will work when you need it. In particular put it in a protective case.

6. Please do not rely on your home router to give your Pi an IP address, because then you will have to bring your router to the lectures as well. Imagine a lecture room with 70+ Wi-Fi routers, all competing for wireless channels! Instead use your laptop as a bridge to the Internet.

7. Make sure that you allow enough time to work on the assignments; some take more time than others. For example #115 takes 8 hours of CPU time.

8. Please keep a record of the time taken to answer the questions so that you can answer question #120.

9. Please try to complete about 30 questions each week, so that you can keep pace with the course.

10. Ask for help if you are stuck during the lectures or the labs, but not via email, and not at the last moment.

Chapter 0 Linux

1. Go to [http://www.raspberrypi.org/help/](http://www.raspberrypi.org/help/) and read about the Pi. Explain what you find exciting about it and what you would like to build (or have
already built) using your Pi. You will be invited to pitch your own favourite Pi project at the last lecture.

2. Go to the Scintilla Stores in the EDU café and buy a Raspberry Pi, an SD card, a casing, a suitable power supply, and an Ethernet cable. Include a list of what you bought, and how much you paid in your answer.

3. Follow the instructions on https://www.raspberrypi.org/help/noobs-setup/ to set up the SD card for your Pi. Read the tips shown on the screen during the install. Boot the system, which will take you into raspi-config. Change the password on the default account “pi”. The first time you see the login prompt, take a picture of your hardware also showing the login prompt on the screen. Include the picture with your answer and not how long it took to set up.

4. Login as pi and create a new account with the user name that you also use on your laptop using the command “sudo adduser <<user name>>”. Explain why it is a bad idea to rely on default passwords.

Answer: It is a bad idea to rely on default passwords since millions of Pi users, and probably the rest of the world know these as well. Hence everybody can use a Pi account with a default password unless the passwords are changed to something that the owner can remember and that is hard to guess.

5. Explain why you cannot log in to the root account on the Pi.

6. Connect your Pi to the Internet and follow the instructions to update it here: https://www.raspberrypi.org/documentation/raspbian/updating.md
Then install the following packages that we will be using later: iftop htop atop nmap netcat. Describe in your answer, which commands you executed and what the purpose of each of the commands is.
7. Set up your Pi so that you can use it as a “headless device”, i.e. so that you can use it without a display, keyboard and mouse. You should be able to connect to your Pi from your laptop via an SSH connection over an ordinary Ethernet cable. Look up on the Internet how to do this (Google “raspberry pi headless”). Note the steps of what you did until the first time you managed to connect to your Pi using the ssh command from your laptop.

8. * Explain how to turn your Pi off properly. What could happen if you turn it off by unplugging the power supply?

9. Use the visudo command to put the account that you created in the sudo file. Explain the advantages and disadvantages of giving your usual account this privilege.

10. With a monitor connected to your Pi you can see some interesting messages appearing on the screen when the system boots. Search the Internet for the command that you can use to see some of these messages on a headless device, and inspect the log files in the directory /var/log on your Pi. Is there something interesting to be seen?

11. * Use the ping command to compare the round trip time of the connection from your laptop to your Pi to the round trip time of your laptop to a server of your choice (e.g. “ping demeter.ewi.utwente.nl”). Explain what the difference is in round trip times and why you think one connection is faster than the other.

12. Is the current date and time set correctly on your Pi? If yes, explain how you achieved this. If not, use the date command to set the date and time of your Pi correctly. Explain what you did to get the correct date and time.

   Answer: The Pi does not have a hardware clock. Instead, it uses the NTP protocol to learn what time it is and then sets the system time accordingly. However, in a headless setup the Pi may not be able to connect to an NTP server so the date and time will not be updated properly. Therefore I have to set the date and time manually like so sudo date -s “10 May 2014 17:50”

13. * Make a backup of the Pi by copying a full image SD card onto your laptop. List the steps that you took to make the backup. How large is the backup?

Chapter 1 Hardware

14. Download the file Download.zip from the course web site and put it into your home directory on your Pi. Then unzip the file and you will have access to all the C and Java programs and shell scripts that you will need to answer the questions below. Explain how you transferred the file onto your Pi.

   Answer: I have used sftp to the static IP address of my Pi because sftp runs over the same port as ssh (i.e. 22). Then I used the sftp command “put Download.zip” to put the zip file on my Pi.
15. * Discover which has the faster CPU: your laptop or your Pi. Compile BenchCPU.c on both machines like so “gcc BenchCPU.c” and measure how much time it takes to execute like so “time ./a.out”. Then change the value of N in the program text and recompile until the execution takes about 10 seconds. Do this on your laptop and on your Pi. Explain the differences. Does the printf statement print the value that you expect? If not explain why.

16. * Depending on the type, the Pi has 4 or 5 status LEDs. Try to figure out what the LEDs indicate by observing the LEDs when the Pi is powered off, when it is booting, when the Ethernet cable is unplugged, and when it is waiting for a command. Explain your findings.

17. Discover which has the faster memory system: your laptop or your Pi. Compile BenchMem.c on both machines and measure how much time it takes to execute. Then change the value of N in the program text and recompile until the execution takes about 10 seconds. Do this on your laptop and on your Pi. Explain the differences. What are you actually measuring in both cases: the performance of the memory or that of the SD card?

18. * Search the Internet which command will report the temperature of your Pi. Use the command to measure the temperature of the Pi once every second when the Pi is otherwise idle and when the Pi is running executing the BenchCPU program. (Hint create a shell script with a loop, and use the command “sleep 1″ to wait for a second). Explain your experiment, with a graph plotting temperature against time for the two experiments.

19. Compile and run Uname.c. If the host name is the default of the Raspbian distribution, change the hostname to something specific (use raspi-config / advanced options / A2 Hostname). Note the steps as part of your answer.

20. Compile and run Vname.c and explain the differences and similarities of Uname.c and Vname.c, as well as their outputs.

21. Connect a USB stick to the Pi. Then run the dmesg command to see what the kernel has done in response. Then type the lsusb command to list the USB devices that the kernel knows about. Explain your findings.

22. * [Stallings 1.13] A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20 ns are required to access it. If it is in main memory, but not in the cache, 60 ns are needed to load it into the cache, and then the reference is started again. If the word in not in main memory, 12 ms (NOT ns) are required to fetch the word from disk, followed by 60 ns to copy to cache, and then the reference is restarted. The cache hit ratio is 90% and the main memory hit ratio is 60%. What is the average time in nanoseconds required to address a referenced word on this system?

23. [Stallings 1.14] Suppose a stack is to be used by the processor to manage procedure calls and returns. Can the program counter be eliminated by using the top of the stack as a program counter?
Chapter 2 Overview

24. * Compile Echo.c and run it as follows “strace ./a.out hello world”. Explain what you see. Which system calls does the main function of the Echo program itself execute, and which system calls are executed by the system to load the program? How did you figure this out?

25. * Compile and execute AddressSpace.c. List the address ranges and discuss what the difference is between (a) the data and the BSS segment, (b) malloc and alloca, (c) the stack and the heap.

26. When you execute the compiled AddressSpace.c several times, is the output always the same? Explain why (not).

27. Compile Wrap.c and run it with strace like so “strace ./a.out Wrap.c junk”. Explain which statements in your code are responsible for which lines in the trace and how you worked this out.

28. [Stallings 2.4] Consider a round-robin scheduler with an idle time of 0.25 µs between any two time-slices (irrespective of whether the same job continues, or whether job switching occurs). If three jobs requiring execution times of 6 µs, 3 µs, and 8 µs are executed, what is the throughput? Assume that a time slice of 1µs is allocated in each round.

29. * [Stallings 2.6] A multiprocessor with eight processors has 20 attached tape drives. There are a large number of jobs submitted to the system, and each requires a maximum of four tape-drives to complete execution. Assume that each job starts running with only three tape drives for a long period before requiring the fourth tape drive for a short period toward the end of its operation. Also assume an endless supply of such jobs.
   a. Assume the scheduler in the OS will not start a job unless there are four tape drives available. When a job is started, four drives are assigned immediately and are not released until the job finishes. What is the maximum number of jobs that can be in progress at once? What is the maximum and minimum number of tape drives that may be left idle as a result of this policy?
   b. Suggest an alternative policy to improve tape drive utilization and at the same time avoid system deadlock. What is the maximum number of jobs that can be in progress at once? What are the bounds on the number of idling tape drives?

Chapter 3 Processes

30. * Compile Fork.c and run it with strace like so “strace ./a.out”. Do you see the trace of both the child and the parent? If not figure out how to trace both processes (Hint: try “man strace”). Which system call is responsible for creating the child process? Explain your findings.

31. Run the compiled version of Fork.c in the background (without strace) like so “./a.out &”. Immediately run the top command to see which processes your “./a.out” command is creating. Were you in time to see anything? If not, change Fork.c so that you can indeed see what is
happening. Explain what you did and the results, including a screen shot of the top command running.

32. * Compile Signal.c like so “gcc -ggdb Signal.c”. Then start the debugger with the program by typing “gdb ./a.out”. Run the program by typing “run”. When the program crashes, type the gdb command “display pointer” to show the value of the variable pointer. Explain why the program has crashed. Use the quit command to end the gdb session.

33. Change the initializer “(int *) NULL” on the declaration of the variable “pointer” in Signal.c such that the program won’t crash. Explain how you achieved this, and include the source of your modified program.

34. Change Signal.c such that another signal than SIGSEGV is caught. Then find a way of generating the signal (either from within the program or from the shell) and demonstrate that the handler catches the signal. Explain how you achieved this, and include the source of your program.

35. Write a new program that combines Fork.c and Signal.c such that the parent process sends a signal that is handled by the child process. Explain how you achieved this, and include the source of your program with your answer.

36. [Stallings 3.10 older edition] UNIX is unsuitable for real-time applications because a process executing in kernel mode may not be pre-empted. Elaborate.

Chapter 4 Threads

37. * Compile MyThread.c like so “gcc -lpthread MyThread.c” and run it with strace like so “strace ./a.out”. Then change N to discover the maximum number of threads that can be created on your laptop and on your Pi. Which system call is responsible for creating the threads? Explain your findings.

38. Explain in detail the purpose of the type-cast “*((int *)” in the program MyThread.c.

39. * Do the threads created by MyThread.c always print their output in the same order? (Hint try running this command several times “./a.out >>junk” and inspect the file junk). Explain your answer.

40. Change the first for-loop in MyThread.c, so that it counts down instead of up. Is this guaranteed to change the output of the program? Explain your answer.

41. Compile MyThread.java like so “javac MyThread.java” and run it like so “time java MyThread”. How much longer does it take to execute the Java version than the C version? Explain why.

42. Does ForkThread.c always produce the same output? Explain your answer.

43. Change ForkThread.c so that it always produces the same output. Explain how you achieved this, and include the source of your program.
44. * [Stallings 4.10] Suppose a single application is running on a multicore system with 16 processors. If 20% of the code is inherently serial, what is the performance gain over a single processor system?

Chapter 5 Concurrency

45. Execute the following shell command pipeline “cat index.html | sort | uniq -c | sort -rn | pr -t”. Explain what each stage in the pipeline does and where there is potential for concurrency. (Hint: use the “tee” command to capture what flows through each pipe).

46. * Compile the Java program Count.java like so “javac Count.java” and execute it with assertions enabled like so: “java -ea Count”. Does the program produce the same result each time you execute it? Explain the results.

47. * Now turn on the semaphore when you run the compiled version of Count.java like so “java -ea Count x”. Does the program produce the same end result in the variable ctr each time you execute it? Is there still concurrency possible? Explain the results.

48. Compare the sources of Count.c and Count.java, and the execution results both with the semaphore turned on. Do both programs produce the same final value for the variable ctr? Explain the results.

49. * Compile Count.c and execute it with “strace -o foo -ff ./a.out”. Study the traces of the three threads and explain which system call is involved in the implementation of a semaphore.

50. Compile Spinlock.c and execute it. Then compile it again with the compilation option “-DYIELD” like so “gcc -DYIELD -lpthread Spinlock.c” and execute it again. Explain the differences between the two outputs.

51. Can you rewrite Spinlock.c with processes instead of threads? Explain your answer.

52. * Compile ProdCons.c and execute it. Then change the initialisation of the Spaces semaphore from N to 1. Compare the results and explain how closely synchronised the two threads are in both cases. Don’t forget to undo the change after you are done.

53. * In ProdCons.c, change the initialisation of the Spaces semaphore from N to “N+1”. Explain what happens. Don’t forget to undo the change after you are done.

54. In ProdCons.c, remove the statement “sem_wait(&Spaces);” and explain what happens. Don’t forget to undo the change after you are done.

55. In ProdCons.c, remove “sem_wait(&Elements);” and explain what happens. Don’t forget to undo the change after you are done.

56. * Compile ProdManyCons.c and execute it once like so “./a.out” and then execute it a second time with a command line option “x” like so “./a.out x”. Compare the outputs and explain the differences.
57. * Read the mq_overview manual page and follow the instructions on “Mounting the message queue file system” (don’t forget to sudo). Explain what you did and what the result is of “ls -l /dev/mqueue”.

58. * Compile MesPass.c like so “gcc -lpthread -lrt MesPass.c” and run the program. Then compile it again like so “gcc -DNONBLOCK -lpthread -lrt MesPass.c”. Then run the program. Explain the differences between the two outputs.

59. Look up in the mq_overview manual page where you can find the maximum number of messages in a queue. If this is less than 12 change it to 12. Explain how you achieved this. Does your change survive a reboot?

60. Find the header file with the definition of the “struct mq_attr”. Explain where you found it and which field is the maximum number of messages.

61. Change the source of MesPass.c so that the maximum number of messages of the queue is N×P. Recompile the non-blocking version of the program and report and explain why there are no more “send failed” messages. (Hint if the program deadlocks, then remove the queue manually like so “rm /dev/mqueue/myqueue”)

62. * [Stallings 5.1] Give an example to illustrate the following quote: “In a race condition, the loser wins”.

Chapter 6 Deadlock

63. Compile the program Detect.c like so “gcc -std=c99 Detect.c” and run it. Then change either the resources or the claims of a process such that none of the processes are deadlocked. Make sure that you do not violate any of the invariants of the program. Explain your choice.

64. * Compile the program Philosophers.c and write a shell script that runs ./a.out until it deadlocks. Kill the program and run it again with a command line option like so “./a.out x”. Will this version ever deadlock? Explain the differences in deadlock behaviour and include the shell script in your answer.

65. * Assume that N=∞ in the program Philosophers.c. Is it possible that one of the philosophers might starve? If so, describe and explain a scenario. If not, explain this.

66. * Compile and run the program Asymmetric.c. Explain why this version of the philosopher’s problem cannot deadlock

67. * [Stallings 6.17] A hungry philosopher first picks up his left fork; if his right fork is also available, he picks up his right fork and starts eating; otherwise, he puts down his left fork and repeats the cycle. Does this protocol prevent deadlock? What, upon finding the left fork unavailable, the philosopher waits a random amount of time and repeats the cycle?
Chapter 7 Memory

68. * Compile ProcessLayout.c and run it. Then change the value of N from 3 to 2 and compare the two outputs. Explain the differences, focusing on which parts of the memory are shared by the threads and which parts are unique to each thread.

69. * Execute pmap with a pid of your choice. Explain why you never see anything mapped to memory address 0 (not to be confused with Process ID 0).

70. Compile the program StackLayout.c and run it on your laptop and on your Pi like so: ".\a.out >junk; sed -f junk <junk". Draw the stack frames of the two machines side by side. Explain the differences and similarities.

71. * [Stallings 7.12] Consider a simple paging system with the following parameters: 2^31 bytes of addressable physical memory; page size of 2^10 bytes; 2^26 bytes of logical address space.
   a. How many bits are in a logical address?
   b. How many bytes in a frame?
   c. How many bits in the physical address specify the frame?
   d. How many entries in the page table?
   e. How many bits in each page table entry (assume each page table entry includes a valid/invalid bit).

Chapter 8 Virtual Memory

72. * Compile Getrusage.c, execute it like so ".\a.out" and like so ".\a.out x" and explain the differences.

73. Compile the program Getrusage.c like so “gcc -DDIRECT Getrusage.c”, run it, and compare the results with the previous version. Explain the differences.

74. * Compile Mmap.c and execute it as follows ".\a.out Mmap.c foo". Explain how the program copies the file Mmap.c to the file foo.

75. Execute the compiled version of Mmap.c like so ".\a.out Mmap.c bar x". Explain the difference between the files foo and bar.

76. * [Stallings 8.13] In discussing a page replacement algorithm, one author makes an analogy with a snowplough moving around a circular track. Snow is falling uniformly on the track and a lone snowplough continually circles the track at constant speed. The snow that is ploughed along the track disappears from the system.
   a. For which of the page replacement algorithms discussed in Section 8.2 of the book is this a useful analogy?
   b. What does this analogy suggest about the behaviour of the page replacement algorithm in question?

Chapter 9 Scheduling

77. Compile the program loop.c like so “gcc -c -o loop.o loop.c” then compile LoopTest.c and link loop.o like so “gcc LoopTest.c loop.o”. Then execute
the program and change N until it takes about 10 seconds CPU time. What is N on your Pi?

78. * Compile and execute ThreadSched.c on your Pi and on your laptop. If one of these machines has multiple cores, then you should see a difference in the output compared to a single core machine. Explain the difference.

79. * Compile SchedXY.c twice like so: “gcc -DX=SCHED_FIFO -DY=80 SchedXY.c -o FIFO80 loop.o” and “gcc -DX=SCHED_RR -DY=80 SchedXY.c -o RR80 loop.o”. Then type “sudo ./RR80&” twice and watch the output of the htop command for a while. Then type “sudo ./FIFO80&” whilst the two RR80 processes are still running and watch the output of the htop command again. Explain what the differences and similarities are of the two scheduler-classes.

80. * Compile the program Nice.c like so “gcc loop.o Nice.c” and run it in the background like so: “./a.out >junk&”. Immediately type top and correlate what you see with the output of the program in the file “junk”. Explain your findings.

81. [Stallings 9.13] An interactive system using round-robin scheduling and swapping (using only time slice pre-emption), tries to give guaranteed response to trivial requests as follows: After completing a round robin cycle among all ready processes, the system determines the time slice to allocate to each ready process for the next cycle by dividing a maximum response time by the number of processes requiring service. Is this a reasonable policy? Explain your answer.

82. * [Stallings 9.15] In priority-based process scheduling, the scheduler only gives control to a particular process if no other process of higher priority is currently in the ready state. Assume that no other information is used in making the process scheduling decision. Also assume that process priorities are established at process creation time and do not change. In a system operating with such assumptions, why would using Dekker’s solution (see Section A.1 of the Book) to the mutual exclusion problem be “dangerous”? Explain this by telling what undesired event could occur and how it could occur.

Chapter 11 I/O

83. Extract the files from the gzipped tar ball “wiringPi-f18c8f7.tar.gz”, compile and install the library and the tools. Test the install by “gpio readall” and explain what this command does.

84. * Connect GPIO pin 17 to the anode of a low power LED and connect the cathode of the LED via a 330 Ohm resistor to GND pin. Never use the 5.0V pin! Write a shell script that will make the led blink. (Google “wiringPi LED” for suggestions). Give the shell script and include a photo of your circuit with your answer.

85. Connect one pin of a light dependent resistor (LDR) via a 1KOhm resistor to the 3.3V pin and connect the other pin of the LDR via a 10uF capacitor to the GND pin. Never use the 5.0V pin! Connect GPIO pin 18 to where the LDR and the capacitor are connected. Write a shell script that
will measure how long it takes for the capacitor to charge, depending on
the ambient light intensity. (Google “wiringPi LDR” for suggestions).
Include the shell script and a photo of your circuit with your answer.

86. Compile Blink.c like so “gcc -lwiringPi Blink.c” and run the program like
so “sudo ./a.out”. Explain in what scheduler class this program runs, and
whether the led would keep blinking as intended on a really busy
system. Try to make the system busy, for example by running a few
processes that hog the CPU.

87. * Execute the shell script sparse.sh like so “sh -x sparse.sh >&
sparse.log&” and compare how long it takes for the Sparse.c program to
create its output file to the time needed by the cat program. Make a
graph showing the relationship between the time consumed as a
function of the size of the files. Include the graph with an explanation in
your answer. Keep the file sparse.log in case you are asked questions
about it during the interview.

88. * Execute the shell script fadvise.sh like so “sh -x fadvise.sh >&
fadvise.log&” and keep the fadvise.log in case of questions during the
interview. Make a table of the execution times and the number of cached
disk blocks of the two runs of the Fadvise.c program and the two runs of
the cp command. Explain the differences.

89. Search the Internet for a C-program that uses the mmap library call to
access the GPIO pins of your Pi. Explain how the C program accesses the
device and include the URL in your answer.

90. * [Stallings 11.7] How much disk space (in sectors, track, surfaces) will
be required to store 307,000 128-byte logical records if the disk is fixed
sector with 512 bytes/sector, with 96 sectors per track, 120 tracks per
surface, and 16 usable surfaces? Ignore any file header record(s) and
track indexes, and assume that records cannot span two sectors.

91. [Stallings 11.8] Consider the disk system described in [Stallings 11.7]
(above), and assume that the disk rotates at 720 rpm. A processor reads
one sector from the disk using interrupt-driven I/O, with one interrupt
per byte. If it takes 1.5 \( \mu \)s to process each interrupt, what percentage of
the time will the processor spend handling I/O (disregard seek time)?

Chapter 12 Files

92. * Execute the shell script ln.sh like so “sh -x ln.sh >& ln.log&” and
interpret the output of the ls commands to explain the differences and
similarities between the files a, b, and c.

93. Execute the shell script mkdir.sh like so “sh -c mkdir.sh >&mkdir.log&”
and compare how long it takes for the Mkdir.c program to create its
output to the time needed by the rm program to remove the output
again, both as a function of the depth of directory structure. Make a
graph and include this with an explanation in your answer.

94. * Modify the program Readdir.c to also output the file size. Explain how
you achieved this and include the source code of your solution in your
answer.
95. * Compile the program Pipe.c and run it a few times. Explain (i) why the output is different each time the program is run, and (ii) why there are four calls to the library call “close(.)” in the program.

96. Compile the program Fifo.c and run it a few times. Explain the similarities and differences between Pipe.c and Fifo.c and show how you can recognize a named pipe with the ls command.

97. Type “sudo fdisk -l”. The insert a USB stick in one of the USB ports of your Pi and type “sudo fdisk -l” again. Explain where in the file system the USB drive can be found.

98. * Create a mount point for the USB stick by typing “sudo mkdir /media/usb” and mount the USB stick, with “sudo mount -t vfat /dev/sda1 /media/usb”. Does “ls -l /media/usb” show the files on your USB stick? Then compile the program Mount.c and run it on your home directory like so “./a.out $HOME” and on the USB stick like so “./a.out /media/usb”. Explain the differences and similarities of the two outputs.

99. Explain what to do before the USB stick can be unplugged safely.

100. Insert a USB stick of at least 1GB **whose contents can be destroyed.** Mount the stick on /media/usb, compile the C program Fadvise.c and execute it twice: once with the output file on the SSD card of your Pi and once on the USB stick. Explain which of the two devices is faster.

101. Use mkfs.ext4 to reformat the 1GB stick like so “sudo mkfs.ext4 /dev/sda1”. **This will destroy its contents.** Mount the stick on /media/usb like so “sudo mount -t ext4 /dev/sda1 /media/usb” and run the compiled version of Fadvise.c once more with the output on the USB stick. Explain which of the two file systems is faster.

102. List the file systems does on your Pi? Explain the purpose of each file system.

103. Run GParted live [http://gparted.org/index.php](http://gparted.org/index.php) on a PC to make the largest partition on your SD card smaller. Describe what you did and what the advantages are.

104. * [Stallings 12.13] Consider the organization of a UNIX file as represented by the inode (Figure 12.16 in the Book). Assume that there are 12 direct block pointers, and a singly, doubly, and triply indirect pointer in each inode. Further, assume that the system block size and the disk sector size are both 4K. Moreover, the disk block pointer is 32 bits, with 8 bits to identify the physical disk and 24 bits to identify the physical block within a disk.
   a. What is the maximum file size supported by this system?
   b. What is the maximum file system partition (that is, the number of addressable bytes) supported by this system?
   c. Assuming that the file inode is already in main memory, how many disk accesses are required to directly access the byte in position 13,423,956?
Chapter 14 Threats

105. Compile and execute the program Thompson.c. Explain the differences and similarities between the program source text and its output.

106. * Compile the program Smash.c and run it in the debugger. Explain why and how the program overwrites the stack; in particular what is the best place to set a breakpoint?

107. * Compile Smash.c but this time with stack protection enabled like so “gcc -fstack-protector-all Smash.c”. Explain how this detects the program from overwriting the stack. Why would stack protection not be enabled by default?

108. Execute the following sequence of commands “sudo sh” then “echo -n 1 >/proc/sys/kernel/print-fatal-signals” and finally “exit”. Then execute the compiled version of Smash.c again, followed by dmesg and explain what the the command d that you typed and what kernel reports about the crash.

109. Run the setuid.sh shell script like so “sh -x setuid.sh >& setuid.log&”. Then log in onto a different account and execute “echo Foo | /tmp/submit”. Explain the differences and similarities between the output of /tmp/submit run on both accounts and the output of “ls -l /tmp/db” run on both accounts.

110. * How many programs are there on your Pi with setuid root? Show the command(s) that you executed in order to find the answer and explain why such programs are potentially dangerous.

111. [Stallings 14.5] Assume that passwords are selected from four-character combinations of 26 alphabetic characters. Assume that an adversary is able to attempt passwords at a rate of one per second.
   a) Assuming no feedback to the adversary until each attempt has been completed, what is the expected time to discover the correct password?
   b) Assuming feedback to the adversary flagging an error as each incorrect character is entered, what is the expected time to discover the correct password?

112. * [Stallings 14.7] The question arises as to whether it is possible to develop a program that can analyse a piece of software to determine if it is a virus. Consider that we have a program D that is supposed to be able to do that. That is, for any program P, if we run D(P), the result returned id TRUE(P is a virus) or FALSE(P is not a virus). Now consider the following program:
   
   program CV :=
   {
   -. main-program :=
   {  if D(CV) then goto next
   else infect-executables;
   }
   next:
   }
In the preceding program, infect-executables is a module that scans memory for executable programs and replicates itself in those programs. Determine if D can correctly decide whether CV is a virus.

Chapter 15 Security

113. Compile and execute the C program Getpwent.c. Then write a program very similar to Getpwent.c that reads the shadow password file. Print the outputs of both programs side by side and explain the differences and similarities, focusing on the passwords. Include the source of your program in your answer.

114. * Compile the C program Lastlog.c and execute it with the user names on your Pi. How big is “/var/log/lastlog”? Explain what the output of the program is and how it can read such a big file in so short a time.

115. Extract the files from the gzipped tar ball “john-1.7.9.tar.gz”, then follow the instructions in doc/INSTALL to compile the program. To test the password cracker, add a new user “test” with password “test”, and execute “sudo ./unshadow /etc/passwd /etc/shadow >crack.txt” in the appropriate directory. To test whether “John the ripper” works, edit crack.txt and save the line for the test account in a separate file test.txt and call “./john test.txt”. This should be quick. Then call “./john crack.txt”. If it takes less than 8 hours to break all your passwords change them so that they are at least twice as hard to break. Explain either why your passwords could not be broken, or which strategy you used to harden your passwords.

116. * Use the nmap command to discover which ports are open on your Pi. Then use netcat to retrieve the welcome banner from each of the open ports. Explain your findings.

117. Execute “sudo badblocks -nv /dev/sda1” to check whether an old USB stick has any issues. Explain your findings.

118. [Stallings 15.7] It was stated that the inclusion of the salt in the Unix password scheme increases the difficulty of guessing by a factor of 4096? But salt is stored in plaintext in the same entry as the corresponding cipher text password. Therefore these two characters are known to the attacker and need not be guessed. Why is it asserted that the salt increases security?

119. * [Stallings 15.13] Assume a system with N job positions. For job position i, the number of individual users in that position is Ui and the number of permissions required for the job position is Pi.
   a. For a traditional DAC scheme, how many relationships between users and permissions must be defined?
   b. For a RBAC scheme, how many relationships between users and permissions must be defined?

120. * Congratulations, you have reached the end of the list of assignments. Explain what you think of this way of examining the operating systems course. How long did it take you to complete the answers?