Study Guide for CS311 Final Exam - Fall 2013

The final is comprehensive

Yes, I know this is a long study guide (about 430 questions or 31 pages). It is shorter than the book however.

The final exam will be (mostly) based from the required reading material in the class. You will not be asked to write big chunks of code. I don't think a proctored final is good setting to write big chunks of code, especially without reference material. The final will be composed of 75-85 questions from required reading material and lectures from the **entire** class. Approximately, 1/3 of the questions (20-25) will be from the first 4 weeks of the class (the material also covered by the midterm). The first ump-teen pages of this document are exactly the same as the review material for the midterm. I include it hear for simplicity.

A question to ask yourself is:

> If Jesse has identified a section/topic for a review question, that probably means that he views that section/topic as (choose one):
>
> a. Significant
> b. Captivating
> c. Exhilarating!!!
> d. All of the above

If you selected d, then we understand each other.

Questions will be of the forms:

- true/false,
- multiple choice,
- short answer, or
- longer answer.

My expectation for a "short answer" question is at most 1 to 2 sentences (more than 2 may be okay, but a lengthy paragraph won't be helpful). For many of the short answer questions, 1 or 2 words would be sufficient. In rare cases, a single character might actually be the right answer. If you see a question "Compare and contrast an artichoke with an anchovy.", then you are looking at the wrong test. I will be grading short answer questions by hand,

so if the correct answer is "file descriptor" and you write "Fïlé DeScRiPtOr", I'll probably get it (but let's not go there). I don't count off for spelling, at least not yet.

With "longer answer" questions you will be asked a question and you need to provide a conclusion and give 3-4 specific examples from the class that support your conclusion. I don't need a fully formed essay worthy of a Pulitzer, a Tony, an Emmy, an Oscar, or even a Razzie, but state your position and give justification for it. These are synthesis questions. I'm more interested in how you choose the support than whether I happen to agree with your conclusion. If I ask you which do you prefer chocolate ice-cream, vanilla, or pistachio, and you go with vanilla and I happen prefer chocolate, but you give 4 good reasons that vanilla is better, that's fine. But we know chocolate is still better. You won't find the longer questions in the study guide.

I am not going to ask you to reproduce giant figures or multi-page tables from TLPI (even I know that is cruel). I am not going to ask you to describe all of the bzillion flags and options that can be given for the `open()` command. However, in the review questions, I may ask about a couple specific ones. That means I may ask about that specific one on the final.

That said, the list of review questions should (select the **best** one):

a. Make an dazzling study guide
b. Make you look good in the nerd herd
c. Make a good fly swatter
d. Make Republicans and Democrats hold hands and sing Kumbaya
e. Make you think that cleaning your room is fun
f. a, b, c, and e, but not d.

This is a year-three (junior level) university undergraduate class in computer science. Material can be hard. Tests can be hard. Reviewing these questions is just downright smart. Combining that with re-reading the identified sections (especially from the book, TLPI) is even smarter-er. All that with reviewing the notes that you've taken from the reading material and lectures (because you have taken notes) is the best of all.

Though taking the final exam is a strictly solo (and proctored) activity, studying for the midterm need not be. Make it a par-tay.

Python Questions

1. The Python programming language got its name from:
    a. http://docs.python.org/2/tutorial/appetite.html
2. Typically, a Python script will end with what letters (the file extension):
    a. http://docs.python.org/2/tutorial/modules.html
3. In Python, a string slice is represented as:
    a. http://docs.python.org/2/tutorial/introduction.html
4. What is the value of x after the following 2 expressions:

```
s = 'this is a ' + 'long ' + 'string'
x = s[10:14]
```

    a. Try it out
5. In Python, a list assignment can be represented as:
    a. a = [‘abc’, ‘def’, 123, 456.789]
    b. a = (‘abc’, ‘def’, 123, 456.789)
    c. a = {‘abc’, ‘def’, 123, 456.789}
    d. a = ‘abc’, ‘def’, 123, 456.789
    e. a = [“abc”, “def”, 123, 456.789]
    f. lists are not a natural Python structure
    http://docs.python.org/2/tutorial/introduction.html (section 3.1.4)
6. After executing the following Python code fragment, the value of x is:

```
x = 0
for y in range(-2, 10):
  x += y
```

    a. Try it out
7. The value of x after the following statement is:

```
x = range(10)
```

    a. http://docs.python.org/2/tutorial/controlflow.html (section 4.3)
8. The value of x after the following code fragment is:

```
x = 0
for y in range(-10,-100, -30):
     x += y
     break
```

    a. Try it out
9. The Python statement that can be used when a statement is required syntactically but the program requires no action is called ____.
    a. See http://docs.python.org/2/tutorial/controlflow.html, section 4.5
10. Python command line arguments are stored in:
    a. http://docs.python.org/2/tutorial/controlflow.html (section 2.1.1)
11. Python uses what to determine code blocks?

    b. Python tutorial 3.2, lecture 3
12. In Python, in order to access the Python time functions you need to use the what line of code?
    a. Lectures 3 and 4

Chapter 1 Questions

1. The UNIX operating system was initially developed at what institution(s)?
    a. TLPI 1.1
2. For this class, GNU stands for:
    a. TLPI 1.2.1, pg 5
3. The first version of Unix was developed by (person's or persons' name)
    a. TLPI pg 2
4. For this course, POSIX stands for:
    a. TLIP page 11
5. A single vendor controlled the development of Unix?
    a. TPLI pg 1
6. The Linux operating system was developed by (person's or persons' name)
    a. TLPI 1.2.2
7. The C Programming Language was developed by (person's or persons' name)
    a. TLPI 1.1
8. Give an example of a recursively-defined acronym.
    a. TPLI 1.2.1
9. What does BSD stand for?
    a. TLPI 1.1

Chapter 2 – Fundamental Concepts

1. Some of the tasks performed by the Unix kernel include (select all that apply)
    a. TLPI 2.1
2. The typical login name for the super-user on a Unix system is:
    a. TLPI pg 26
3. The typical segments for a Unix process include (select all that apply):
    a. TLPI pg 31
4. A "software interrupt" is another name for:
    a. TLPI pg 37
5. The two types of time on a Unix system are:
    a. TLPI pg 40
6. On a Unix system, the /etc/passwd file contains what information (identify 3):
    a. TLPI 2.3
7. On Unix, the root directory to the file system is named what?
    a. TLPI 2.4

8. A relative path name is (mark all that apply)
    a. Location of a file relative to a process's current working directory
    b. Must begin with a ~
    c. Starts from the /home directory.
    d. Distinguished from an absolute pathname by the absence of an initial slash
    e. Specifies the location of a file with respect to the root directory
    f. Always begins with .. or .
    g. TLPI 2.4
9. An instance of an executing program is called a:
    a. TLPI 2.7
10. The "text" segment of a process contains:
    a. TLPI 2.7
11. Define a *realtime application*, as described in TLPI.
    a. TLPI 2.18
12. Describe the some differences between kernel mode and user mode for a CPU.
    a. TLIP 2.1
13. Describe some of the characteristics of a privileged process.
    a. TLPI 2.7
14. What are the 2 commonly used meanings for *operating system*?
    a. TLPI 2.1
15. In this class, the word *kernel* means what?
    a. TLPI 2.1
16. What is a login shell?
    a. TLPI 2.2
17. Identify 3 things that you'll find in the /etc/group file on a Unix system.
    a. TLPI 2.3
18. What is the base of the file system hierarchy called on Unix?
    a. TLPI 2.4
19. Every directory in the Unix file system contains what 2 entries?
    a. TLPI 2.4
20. What is the *current working directory*?
    a. TLPI 2.4
21. What is a file descriptor?
    a. TLPI 2.5
22. What 3 open file descriptors does a process started by a shell normally inherit?
    a. TLPI 2.5
23. The "parent of all processes" on a Unix system is called what?
    a. TLPI 2.7
24. What distinguishes a *daemon* process from other processes on Unix?
    a. TLPI 2.7

Chapter 3 – System Programming Concepts

1. When a Unix system call fails, it sets a global integer called:
    a. TLPI section 3.4
2. Define what a system call is.
    a. TLPI 3.1
3. Identify 3 characteristics of how system calls work:
    a. TLPI 3.1
4. Successful system calls and library functions always reset errno to 0.
    a. True or False
    b. TLPI 3.4
5. What does a call to perror() do?
    a. TLPI 3.4
6. All library functions return a -1 on error and sets errno to indicate a specific error.
    a. TLPI 3.4
7. Identify 2 reasons why it is better to use system data types rather than standard C types for things such as process id, user id, and file offsets.
    a. TLPI 3.6.2
8. Identify 3 system data types
    a. TLPI Table 3-1

Chapter 4 – File I/O: the Universal I/O Model

1. Match the file descriptors with their common name:
    a. 0 _____
    b. 1 _____
    c. 2 _____
    TLPI section 2.5, table 4-1
2. What does read() return, when successful?
    a. TLPI section 4.1
3. When using the system call read(), how do you know when you've reached the end of the file?
    a. TLPI 4.4
4. What does a call to the system function write() return, when successful?
    a. TLPI section 4.5
5. A successful call of the open() system call returns _____.
    a. TLPI section 4.1
6. After performing all I/O, we should free the file descriptor and its associated resources using _____ system call.
    a. TLPI 4.6
7. For each open file, the kernel records a *file offset*, which is the location in the file at which the next *read()* or *write()* will commence.

a. TLPI 4.7

8. That a program can typically be used with any type of file without requiring code that is specific to the file type, is called _____.
   TLPI 4.2
9. When using the lseek() system call, a value of SEEK_CUR for the whence parameter will adjust the file offset
   a. TLPI 4.7
10. A lseek() call with a pipe will _____
    a. TLPI 4.7
11. What does the error code EACCES indicate?
    a. TLPI 4.3.1
12. What does the error code ENOENT indicate?
    a. TLPI 4.3.1
13. A call to the read() function always places a NULL character at the end of the buffer.
    a. TLPI 4.4
14. What causes "file holes"?
    a. TLPI 4.7
15. On Linux, file holes a common cause of a full disk.
    a. TLPI 4.7
16. Under what conditions is a file's nominal size larger than the amount of disk storage it utilized?
    a. TLPI 4.7
17. Describe the O_APPEND flag for the open() system call.
    a. TLPI 4.3.1
18. Describe the O_NOATIME flag for the open() system call.
    a. TLPI 4.3.1
19. A call to *read()* may read less than the requested number of bytes. For a regular file, the probable reason for this is that _____
    a. TLPI 4.4

Chapter 5 – File I/O: Further Details

1. The concept that the kernel guarantees that all of the steps in a system call are completed as a single operation, without being interrupted by another process or thread is called _____
   TLPI pg 90
2. What system call would you use to change the access mode and open status flags on an open file (file descriptor)?
   a. TLPI 5.3
3. The pread() call performs just like the normal read() call, except pread() only works on pipes.
   a. TLPI 5.6

4. The pwrite() system call allows a multi-threaded application to use a shared open file descriptor and multiple threads can simultaneously perform I/O on the same file descriptor without being affected by changes made to the file offset by other threads.
    a. TLPI 5.6
5. A call to truncate() or ftruncate() always results in a 0 size file, assuming you have access rights on the file.
    a. TLPI 5.8
6. To ensure that a process is the creator of a file, combine the _____ flag with the O_CREAT flag when calling open().
    a. TLPI 5.1
7. The kernel maintains a system-wide table of all *open file descriptions.* Included in the information the kernel maintains is (list 3):
        TLPI 5.4
8. Two common system calls the duplicate file descriptors are:
    a. TLPI 5.5
9. An easy way to create a temporary file that will be automatically deleted when the application exists is to use what function?
    a. TLPI 5.12
10. Describe what a race condition is.
    a. TLPI 5.1
11. There is a one-to-one correspondence between a file descriptor and an open file.
    a. TLPI 5.4
12. Describe what the dup2() system call does.
    a. TLPI 5.5
13. What 2 system calls can be used for scatter-gather I/O?
    a. TLPI 5.7
14. Describe the /dev/fd directory.
    a. TPLI 5.11
15. Describe the mkstemp() system call.
    a. TLPI 5.12

Chapter 10 – Time

1. Define process time.
    a. TLPI chap 10, intro
2. On a Unix system, the Epoch began:
    a. TLPI 10.1
3. What is the Year 2038 problem?
    a. TLPI 10.1
4. A call to ctime(ctime(const time_t *timep)) will return:
    a. TLPI 10.2.1

5. Time zone information is compiled into the Unix kernel
    a. TLPI 10.3
6. In this class, the term “locale” refers to:
    a. TLPI 10.4
7. Describe the two components of process time:
    a. TLPI 10.7
8. The software clock on a Unix system measures time in:
    a. TLPI 10.6
9. What does the time() system call return?
    a. TLPI 10.1
10. Describe broken-down time.
    a. TPLI 10.2.2
11. Describe the system call asctime().
    a. TPLI 10.2.3
12. Describe how the TZ environment variable can be used.
    a. TLPI 10.3
13. Describe the adjtime() system call.
    a. TLPI 10.5

Chapter 15 – File Attributes

1. Three system calls to retrieve information about a file are:
    a. TLPI 15.1
2. The difference between the stat() and lstat() calls is that the lstat() call will always resolve symbolic links down to the actual file.
    a. TLPI 15.1
3. The struct stat{} returned by a call to the stat() functions contains a member st_nlink. This value represents _____
    a. TLPI 15.1
4. The struct stat{} returned by a call to the stat() functions contains a member st_blocks. This value represents _____
    a. TLPI 15.1
5. Identify the 3 different time stamps for a file.
    a. TLPI 15.2
6. What is the system call to change the permissions on a file or directory?
    a. TLPI 15.4.7
7. What is the system call to change the ownership of a file?
    a. TLPI 15.3.2
8. The execute permission on a directory is also called ____ permission.
    a. TLPI 15.4.2
9. Having execute permission on a directory, but not read permission allows you to:
    a. TLPI 15.4.2

10. Describe the access() system call:
    a. TLPI 15.4.4
11. In modern UNIX implementations, the sticky is on directories does what?
    a. TLPI 15.4.5
12. The *fstat()* system call always succeeds, if provided with a valid file descriptor.
    a. TLPI 15.1
13. Describe the S_ISFIFO() test macro.
    a. TLPI 15.1, table 15-1
14. How do you find the optimal block size for I/O on files on a file system?
    a. TLPI 15.1
15. When a new file is created, its user ID is taken from the ____
    a. TLPI 15.3.1
16. Describe the lchown() system call.
    a. TLPI 15.3.2
17. The file permissions mask divides the world into three categories (identify all three).
    a. TLPI 15.4.1
18. A file has the following permissions: -rwxr-x--x. Describe the permissions for the file.
    a. TPLI 15.4.1
19. Describe the meaning of having your umask value set to 022.
    a. TLPI 15.4.6
20. Describe the shell command: chmod u+w,o-r myfile
    a. TLPI 15.4.7

Chapter 12 – System and Process Information

1. To get system name information, you can call this system function ____
    a. TLPI 12.2
2. The /proc file system can be used to answer the following questions (check all that apply):
    a. How many processes are running in the system and by whom they are owned
    b. What files a process has open
    c. What files are locked and which processes have the locks
    d. What sockets are being used on the system
    e. What color the system is
    f. If the user prefers emacs or vim
    g. TLPI 12.1
3. The /proc file system contains a directory for each active process in the system.
    a. TLPI 12.1.3

Chapter 20 – Signals: Fundamental Concepts

1. Signals are queued.
    a. TLPI 20.12
2. The occurrences of a signal can be reliably counted.
    a. TLPI 20.12
3. Delivery of a signal is blocked during the execution of its handler.
    a. TLPI 20.12
4. Events that cause a signal to be sent to a process include:
    a. TLPI 20.1
5. If you want to reset the disposition of a signal to its default, there is a constant that can be used in place of a handler function. What is it?
    a. TLPI 20.3
6. A program can set one of the following dispositions for a signal: (check all that apply)
    a. The signal is forwarded to the kernel.
    b. The *default action* should occur.
    c. The signal is *ignored.*
    d. The signal is promoted.
    e. A *signal handler* is executed.
    f. The signal is reentrant
    g. The signal is atomized
    TLPI chapter 20
7. On Linux, the standard signals are numbered ____
    a. TLPI chapter 20.1 and 20.2
8. The value of errno will not change within a signal handler.
    a. TLPI chapter 20
9. All signals can be caught or blocked.
    a. TLPI chapter 20
10. What event is represented by the SIGINT signal?
    a. TLPI 20.2
11. What system calls can be used to change the disposition of a signal (establish a new signal handler)?
    a. TLPI 20.3
12. Why is sigaction() preferred over signal() for establishing a signal handler?
    a. TLPI 20.3
13. What system calls can be used to send a signal to a process?
    a. TLPI 20.5
14. When using the kill() system call, what occurs when the pid to which the signal is sent is equal to 0?
    a. TLPI 20.5
15. What is the purpose of using the kill() system call with a sig argument as 0 (the null signal)?

    a. TLPI 20.6
16. What system call can be used to explicitly add signals to, and remove signals from, the signal mask?
    a. TLPI 20.10
17. What system call will suspend a process until a signal is received?
    a. TLPI 20.14
18. Describe the SIGCHLD signal.
    a. TLPI 20.1
19. Describe the SIGSEGV signal.
    a. TLPI 20.1
20. What system call can be used to determine which signals are pending for a process?
    a. TLPI 20.11

Chapter 21- Signals: Signal Handlers

1. What can cause signals to "disappear" from a process?
    a. TLPI 21.1.1
2. A function that can safely be simultaneously executed by multiple threads of execution in the same process, is called:
    a. TLPI 21.1.2
3. A function that employs only local variables is guaranteed to be reentrant.
    a. TLPI 21.1.2
4. Explain why malloc() and free() are not reentrant.
    a. TLPI 21.1.2
5. Calling printf() from within a signal handler can result in (list 3):
    a. TLPI 21.1.2
6. What 2 characteristics are required to make a function async-signal-safe?
    a. TLPI 21.1.2
7. A function where the implementation guarantees to be safe when called from a signal handler is called:
    a. TLPI 21.1.2
8. When using errno within a signal handler, how do you avoid making the signal handler nonreentrant?
    a. TLPI 21.1.2
9. A global variable shared between the main program and a signal handler should be of what type?
    a. TLPI 21.1.3
10. What is special about the C system type sig_atomic_t?
    a. 21.1.3
11. Incrementing or decrementing a C variable of the system type sig_atomic_t is guaranteed to be atomic.
    a. TLPI 21.1.3

12. The function that causes a program to terminate and produce a core dump is:
    a. TLPI 21.2.2
13. Identify 3 functions required to be async-signal-safe by POSIX.
    a. 21.1.2, table 21-1
14. Describe the abort() function.
    a. TLPI 21.2.2

Chapter 24 – Process Creation

1. What system call does a parent process use to create a new child process?
    a. TLPI 24.1
2. What system call is used by a parent process find the termination status of a child process?
    a. TLPI 24.1
3. If a parent process calls wait() and no child process has yet exited, the parent process will ____
    a. TLPI 24.1
4. After creating a child process using *fork()*, which runs first, the parent process or the child process?
    a. TLPI 24.2
5. When successful, for the parent, *fork()* returns the _____ of the newly created child process.
    a. TLPI 24.2
6. When successful, for the child, *fork()* returns _____ (what value).
    a. TLPI 24.2
7. A child process can find the process id of its parent making what system call?
    a. TLPI 24.2
8. When a fork() is performed, the child receives duplicates of all of the parent's file descriptors.
    a. TLPI 24.2.1
9. If a call to fork() fails, the return value is ___
    a. TLPI 24.2
10. Identify 2 distinguishing differences between fork() and vfork().
    a. TLPI 24.3
11. Unlike fork(), the child from a call to vfork() is guaranteed to be scheduled first, before the parent process.
    a. TLPI 24.3

Chapter 27 – Program Execution

1. A successful call to one of the `exec()` functions returns what value?
    a. TLPI 27.1

2. The `exec()` set of commands cannot be used to execute shell scripts (interpreter scripts).
    a. TLPI 27.3
3. By default, all file descriptors opened by a program that calls *`exec()`* remain open across the *`exec()`* and are available for use by the new program.
    a. TLPI 27.4
4. Describe the close-on-exec flag.
    a. TLPI 27.4
5. When `dup(), dup2(), or fcntl()` is used to create a duplicate of a file descriptor, the close-on-exec flag is always cleared for the duplicate descriptor.
    a. TLPI 27.4
6. During a call to an `exec()` function, the kernel maintains the dispositions of all handled signals (installed signal handlers remain in place).
    a. TLPI 27.5
7. Describe 3 conveniences of using the system() function.
    a. TLPI 27.6
8. After an execve() call, the process ID remains the same.
    a. TLPI 27.1
9. Which of the exec() functions make use of the PATH environment variable to search for the executable file?
    a. TLPI 27.2.1
10. Why is the current working directory normally excluded from the PATH of the root (superuser) account?
    a. TLPI 27.2.1

---------- Midterm ----------

Chapter 25 – Process Termination

1. In the call _exit(int status), although defined as an *int,* only the bottom __ bits of *status* are actually made available to the parent.
    a. TLPI 25.1
2. By convention, a termination status of _____ indicates that a process completed successfully
    a. TLPI 25.1
3. Before a call to *_exit()* completes, calls are made to exit handlers (atexit() and onexit()) and stdio buffers are flushed.
    a. TLPI 25.1
4. Exit handlers are way for a process to perform cleanup before exiting.
    a. TLPI 25.3
5. Registered exit handlers are always called before a process terminates.
    a. TLPI 25.3
    b. NEED TO FIX QUIZ
6. A process is ALWAYS successfully terminated by _exit().
    a. TLPI 25.1
7. Describe 2 actions that are performed by a call to exit()?
    a. TLPI 25.1
8. What is an exit handler?
    a. TLPI 25.3
9. A process can only have a single exit handler registered.
    a. TPLI 25.3
10. A child process created with fork() inherits a copy of its parent's exit handler registrations.
    a. TLPI 25.3
11. What function calls can be used to register exit handlers?
    a. TLPI 25.3
12. Performing a return without specifying a value, or falling off the end of the main() function is equivalent to calling exit() from main()?
    a. TLPI 25.1
13. Identify 5 actions that occur during both normal and abnormal termination of a process.
    a. TLPI 25.2
14. When are exit handlers NOT called?
    a. TLPI 25.3
15. When multiple exit handlers are registered for a process, in what order are the functions called when exiting?
    a. TLPI 25.3
16. What happens to the remaining exit handlers when an exit handler fails to return?
    a. TLPI 25.3
17. What are 2 limitations of exit handlers registered with atexit()?

a. TLPI 25.3

Chapter 26 – Monitoring Child Processes

1. On a successful call to wait(), if no (previously unwaited-for) child of the calling process has yet terminated, the call ___ until one of the children terminates.
    a. TLPI 26.1.1
2. Child processes are reaped using what system call?
    a. TLPI 26.4
3. A common way of reaping dead child processes is to establish a handler for the ____ signal.
    a. TLPI 26.4
4. In order to kill a zombie process you must _____
    a. TLPI 26.2
5. A successful call to wait() returns what value?
    a. TLPI 26.1.1
6. What are the limitations of the wait() call that are addressed by waitpid()?
    a. TLPI 26.1.2
7. The status value from wait() or waitpid() allows what termination events to be determined for the child process?
    a. TLPI 26.1.3
    b. reword
8. The principle difference between wait3() and wait4() from waitpid() is that _____
    a. TLPI 26.1.6
9. What process becomes the parent of an orphaned child process?
    a. TLPI 26.2
10. What happens to a child process that terminates before its parent process has had a chance to perform a wait()?
    a. TLPI 26.2
11. What signal is sent to the parent process whenever a child process terminates?
    a. TLPI 26.3
12. What is an example of when a call to wait() returns a -1?
    a. TLPI 26.1.1
13. Describe how waitpid(pid, …) performs then the value of pid is greater than 0.
    a. TLPI 26.1.2
14. What does the WNOHANG option do when part of the options argument to waitpid()?
    a. TLPI 26.1.2
15. When the macro WIFSIGNALED(status) to dissect a wait status value,

what can it tell you?
    a. TLPI 26.1.3
16. What can (eventually) happen if a large number of such zombie children are created?
    a. TLPI 26.2
17. Under what condition can a signal handler for SIGCHLD still leave zombie process on a system?
    a. TLPI 26.3.1
18. What happens when the disposition of SIGCHLD is set to SIG_IGN?
    a. TLPI 26.3.3
19. How is SIGCHLD treated uniquely among signals?
    a. 26.3.3

Chapter 44 – Pipes and FIFOs

1. When TLPI refers to a pipe as a byte stream, it means that _____
    a. TLPI 44.1
2. Data in a pipe can be randomly accessed using lseek()?
    a. TLPI 44.1
3. Attempts to read from an empty (but open) pipe will, ____
    a. TLPI 44.1
4. If multiple processes are writing to a single pipe, then it is guaranteed that their data won't be intermingled if ____
    a. TLPI 44.1
5. When writing to a pipe, the call to write() will block until:
    a. TLPI 44.1
6. What does "pipes can be used between 'related' processes" mean?
    a. TLPI 44.2
7. The system calls normally used to perform I/O on pipes are:
    a. TLPI 44.2
8. The popen() command can be used to write to **and** read from an executed command.
    a. TLPI 44.5
9. Other names for FIFO's are:
    a. TLPI 44.7
10. The principle difference between a pipe and a FIFO is _____
    a. TLPI 44.7
11. Once a FIFO has been created, it can be only be opened by a "related" process (subject to file permission checks).
    a. TLPI 44.7
12. When a process opens one end of a FIFO, it blocks if the other end of the FIFO has not yet been opened.
    a. TLPI 44.9

13. What is the oldest method of IPC on the Unix system?
    a. TLPI 44.0
14. What does the statement "Pipes have limited capacity" mean?
    a. TLPI 44.1
15. A successful call to pipe() returns ____
    a. TLPI 44.2
16. Give an example of what can happen when the unused pipe file descriptors are not closed.
    a. TLPI 44.2
17. What is the return value from popen() when an error occurs?
    a. TLPI 44.5
18. Why should pclose() be used and fclose() not be used on the file stream pointer returned from popen()?
    a. TLPI 44.5
19. Like the system() call, what conveniences does popen() provide?
    a. TLPI 44.5
20. By default, block buffering is not applied to the file stream created by popen().
    a. TLPI 44.6
21. When a FIFO is closed, any outstanding data within it is ____
    a. TLPI 44.7
22. The shell command to create a FIFO is ____
    a. TLPI 44.7

Chapter 46 – System V Message Queues

1. The primary distinguishing characteristic between IPC using pipes and messages queues is:
    a. TLPI chap 46, intro
2. When using message queues, it is possible to write only part of a message.
    a. TLPI 46.2.1
3. Like pipes, the data in message queues can only be read in the order in which it was written.
    a. TLPI 46.2.2
4. The function call for sending a System V message is:
    a. TLPI 46.2.1
5. On success, a call to msgsnd() returns what value?
    a. TLPI 46.2.1
6. The function for receiving a System V message is:
    a. TLPI 46.2.2
7. On success, a call to msgrcv() returns what value?
    a. TLPI 46.2.2
8. How do you use a System V message queue as a priority queue?
    a. TLPI 46.2.2

9. When using System V message queues, what does MSGMNI represent (the description, not the actual value).
    a. TLPI 46.5
10. What will the command "ipcs –q" show you?
    a. Check the man page
11. The handle returned by a call to the System V msgget() is actually just a file descriptor.
    a. TPLI 46 Intro
12. What does the error code E2BIG indicate when returned from a call the System V function msgrcv()?
    a. TLPI 46.2.2

Chapter 52 – POSIX Message Queues

1. What does the statement "POSIX message queues are reference counted." imply about deletion of POSIX message queues?
    a. TLPI chap 52, intro
2. The call to place data into a POSIX message queue is:
    a. TLPI 52.1
3. The call to read a POSIX message queue is:
    a. TLPI 52.1
4. The POSIX message queue call mq_unlink() will ___
    a. TLPI 52.1
5. When sending POSIX messages, messages are ordered within the queue in descending order of priority.
    a. TPLI 52.5.1
6. If a POSIX message queue is full, then a mq_send() will ___
    a. TLPI 52.5.1
7. Describe what POSIX message queue call mq_receive() does with a queue when called.
    a. TLPI 52.5.2
8. If a POSIX message queue is currently empty, then a mq_receive() call ____
    a. TLPI 52.5.2
9. Describe the POSIX message queue call mq_notify() ___
    a. TLPI 52.6
10. How many processes can be registered to receive notification about a new message on a POSIX message queue?
    a. TLPI 52.6
11. If process A is blocked on a mq_receive() on a POSIX message queue and process B registers for notification on the same message queue, which process will receive the next message?
    a. TLPI 52.6

12. When using POSIX message queues, what does MQ_OPEN_MAX value represent?
    a. TLPI 52.8
13. When a POSIX message queue is created, using mq_open(), what 2 mq_attr fields determine the maximum amount of memory the kernel allocates?
    a. TLPI 52.4
14. When a new message is added to the queue, it is placed _____ any other messages of the same priority.
    a. TLPI 52.5.1
15. Describe the abs_timeout argument to the POSIX message queue calls mq_timedsend() and mq_timedreceive().
    a. TLPI 52.5.3
16. If a POSIX message queue already contains messages at the time of a call to mq_notify(), a notification will occur only after the queue is emptied and a new message arrives.
    a. TLPI 52.6
17. Once a process has called mq_notify() for a POSIX message queue, it will continue to receive notification for new messages on that queue.
    a. TLPI 52.6
18. An advantage of POSIX message queues over System V message queues is
    a. TLPI 52.9
19. An advantage of System V message queues over POSIX message queue is
    a. TLPI 52.9

Chapter 29 – Threads: Introduction

1. Threads allow a single process to perform multiple tasks concurrently.
    a. TLPI 29.1
2. Threads in a process do not share global memory.
    a. TLPI 29.1
3. Which is faster, creation of a new process with fork() or creation of a new thread with pthread_create()?
    a. TLPI 29.1
4. Some of the attributes that threads within a process **share** are:
    a. TLPI 29.1
5. Some of the attributes that are **distinct** for each thread are:
    a. TLPI 29.1
6. When using Pthreads, each thread has its own errno variable.
    a. TLPI 29.2
7. On failure, all Pthreads functions return _____
    a. TLPI 29.2
8. When a new thread is created, which will run first, the new thread or the old thread?

    a. TLPI 29.3
9. To determine if 2 thread ids are the same, the ____ function calls should be used.
    a. TLPI 29.5
10. You can rely on a thread id being unique across all processes in a system.
    a. TLPI 29.5 (Linux yes, others no)
11. The *pthread_join(thread_id,status)* function blocks until the thread identified by *thread_id* _____.
    a. TLPI 29.6
12. What can result from not calling pthread_join() on completed threads?
    a. TLPI 29.6
13. What is the call used to "join with any thread" in a process?
    a. TPLI 29.6
14. When you don't care about a thread's return status; you simply want the system to automatically clean up and remove the thread when it terminates, you say the thread is _____
    a. TLPI 29.7
15. Identify 3 of the ways in which a thread (from PThreads) can terminate.
    a. TLPI 29.4
16. Why should the retval value used in a call to pthread_exit() not be located in the calling thread's stack?
    a. TLPI 29.4
17. Identify 1 reason why thread IDs are useful within multi-threaded applications.
    a. TLPI 29.5
18. Why should pthread_equal() be used to compare thread IDs?
    a. TLPI 29.5
19. Calling pthread_join() for a thread ID that has been previously joined can ____
    a. TLPI 29.6
20. Identify some points that can influence a choice to build an application as multi-threaded or multi-process.
    a. TLPI 29.9

Chapter 30 – Threads: Thread Synchronization

1. In TLIP, the term *critical section* is used to refer to a section of code that accesses a shared resource and whose execution should be
    a. TLPI 30.1
2. What does it mean that mutex locking is advisory?
    a. TLPI 30.1
3. The word mutex is short for
    a. TLPI 30.1
4. A mutex has 2 states, what are they?

a. TLPI 30.1

5. When accessing a shared resource, each thread employs a three step protocol for accessing the resource:
    a. TLPI 30.1
6. Before a statically allocated mutex can be used, it should be initialized with the value:
    a. TLPI 30.1.1
7. If another thread currently has a muetx locked, a call to pthread_mutex_lock() will _____
    a. TLPI 30.1.2
8. On Linux, if a thread tries to lock a mutex for which it already has a lock, the default behavior is to:
    a. TLPI 30.1.2
9. To dynamically initialize a mutex, this call should be used:
    a. TLPI 30.1.5
10. Initializing an already initialized mutex results in
    a. TLPI 30.1.5
11. An automatically allocated mutex does not need be destroyed before its host function returns.
    a. TLPI 30.1.5
12. Describe a PTHREAD_MUTEX_RECURSIVE mutex type.
    a. TLPI 30.1.7
13. A condition variable allows one thread to _____
    a. TLPI 30.2
14. A statically created condition variable must be initialized with the following value before use:
    a. TLPI 30.2.1
15. It is safe to destroy a condition variable even when threads are waiting on it.
    a. TLPI 30.2.5
16. At any moment, at most ____ thread may hold the lock on a mutex.
    a. TLPI 30.1
17. Because of ownership property when locking a mutex, the terms _____ and _____ are sometimes used synonymously for lock and unlock.
    a. TLPI 30.1
18. The calls to lock and unlock a mutex are _____ and _____.
    a. TLPI 30.1.2
19. How does the call pthread_mutex_trylock() differ from pthread_mutex_lock()?
    a. TLPI 30.1.2
20. A thread that uses pthread_mutex_trylock() to periodically poll the mutex to see if it can be locked _____
    a. TLPI 30.1.2

21. What bad kinds of conditions can arise when threads try and acquire tow or more different resources?
    a. TLPI 30.1.4
22. The simplest way to avoid mutex deadlocks is to _____
    a. TLPI 30.1.4
23. Describe the "try, and then back off" strategy for acquiring multiple mutexes.
    a. TLPI 30.1.4
24. It is safe for a thread that owns the lock on a mutex to destroy it, while it is locked.
    a. TLPI 30.1.5
25. Describe the PTHREAD_MUTEX_ERRORCHECK mutex type.
    a. TLPI 30.1.7
26. A condition variable is always used in conjunction with a ____
    a. TLPI 30.2
27. Describe the difference between pthread_cond_signal() and pthread_cond_broadcast().
    a. TLPI 30.2.2
28. Describe the pthread_cond_init() function.
    a. TLPI 30.2.5
29. It is safe to destroy a condition variable when threads are waiting on it.
    a. TLPI 30.2.5

Chapter 33 – Threads: Further Details

1. When a new thread is created, it receives its own stack, whose size is unbounded.
    a. TLIP 33.1
2. On x86-32 Linux, where the user-accessible virtual address space is 3 GB, the default stack size of 2 MB means that we can create a maximum of about ___ threads.
    a. TLPI 33.1
3. Why might a process choose to have a smaller stack size per thread?
    a. TLPI 33.1
4. If an unhandled signal whose default action is *stop* or *terminate* is delivered to any thread in a process, then all of the threads in the process are stopped or terminated.
    a. TLPI 33.2.1
5. When a signal is delivered to a multithreaded process that has established a signal handler, the kernel arbitrarily selects one thread in the process to which to deliver the signal and invokes the handler in that thread.
    a. TLPI 33.2.1
6. By manipulating the per-thread signal masks, an application can control

which thread(s) may handle a signal that is directed to the whole process.
    a. TLPI 33.2.1
7. To send a signal to a specific thread in the same process, use the following call.
    a. TPLI 33.2.3
8. Because a thread ID is guaranteed to be unique only within a process, we can't use pthread_kill() to send a signal to a thread in another process.
    a. TLPI 33.2.3
9. Describe how asynchronous signals can be sanely managed within a multi-threaded application.
    a. TLPI 33.2.4
10. When calling one of the exec() functions within a multi-threaded application (check all that apply):
    a. TLPI 33.3
11. The usual recommendation is that the only use of *fork()* in a multithreaded process should be one that is followed by an immediate *exec()*.
    a. TLPI 33.3
12. When a multithreaded process calls fork(), all the calling threads are replicated in the child process.
    a. TLPI 33.3
13. For programs that must use a fork() that is not followed by an exec(), the Pthreads API provides a mechanism for defining _____
    a. TLPI 33.3
14. What are 2 disadvantages of the many-to-one (user-level) thread modes?
    a. TLPI 33.4
15. What are 2 disadvantages of the one-to-one (kernel-level) thread models?
    a. TLPI 33.4
16. Which thread model does the Linux threading implementations use?
    a. TLPI 33.4
17. If a signal handler interrupts a call to pthread_mutex_lock(), then the call is always automatically restarted.
    a. TLPI 33.2.1
18. Describe the pthread_sigmask() function.
    a. TLPI 33.2.2
19. Describe the pthread_sigqueue() function.
    a. TLPI 33.2.3
20. When any thread calls one of the exec() functions, the calling program is completely replaced. All threads, except the one that called exec(), vanish immediately.
    a. TLPI 33.3
21. Describe the many-to-one (M:1) implementation (user-level threads) model.
    a. TLPI 33.4

22. Describe the One-to-one (1:1) implementations (kernel-level threads) model.
    a. TLPI 33.3
23. When discussion threading models, what does KSE stand for?
    a. TLPI 33.4

PThreads – Tutorial

1. When compared to the cost of creating and managing a process, a thread can be created with much less operating system overhead.
    a. Pthreads tutorial, Why Pthreads?
2. The subroutines which comprise the Pthreads API can be roughly grouped into four major groups: (select all that apply)
    a. Pthreads tutorial, The Pthreads API
3. Describe 4 characteristics of a thread the in the UNIX PThreads environment.
    a. PThreads, What is a Thread?
4. What are some considerations for designing parallel programs?
    a. PThreads, Designing Threaded Programs
5. In a nutshell, thread safeness refers to _____
    a. PThreads, Thread Safeness
6. After a thread has been created, how do you know a) when it will be scheduled to run by the operating system, and b) which processor/core it will run on?
    a. PThreads, a question and answer in Thread Binding and Scheduling
7. When more than one thread is waiting for a locked mutex, which thread will be granted the lock first after it is released?
    a. PThreads, Mutex Variables

Chapter 51 – Introduction to POSIX IPC

1. The three POSIX IPC mechanisms are:
    a. TLPI chap 51, intro
2. On Linux, names for POSIX shared memory and message queue objects are limited to _____ characters (a name not a number).
    a. TLPI 51.1
3. For POSIX IPC mechanisms, calling the unlink function on the object
    a. TLPI 51.1
4. POSIX IPC objects are reference counted. What does this imply?
    a. TPLI 51.2
5. Unlike Unix pipes, POSIX message queues have message boundaries that are preserved, so that readers and writers communicate in units of messages.
    a. TLPI 51 intro
6. POSIX semaphores are allocated in sets, like System V semaphores.

    a. TLPI chap 51 intro
7. One notable advantage that favors of System V IPC mechanisms over POSIX IPC is:
    a. TLPI 51.2
8. The handle returned by the IPC open call is analogous to _____
    a. TLPI 51.1
9. After a POSIX IPC object is unlinked, an IPC open call specifying the same object name will refer to a new object.
    a. TLPI 51.1
10. What does the statement "POSIX IPC objects have kernel persistence" mean?
    a. TLPI 51.1
11. System V IPC objects can be listed with the ipcs command. What is the POSIX IPC equivalent command?
    a. TLPI 51.1

Chapter 53 – POSIX Semaphores

1. SUSv3 specifies two types of POSIX semaphores (identify both):
    a. TLPI 53.1
2. When a child process is created via fork(), the child inherits references to all of the named semaphores open in its parent (at the time of the fork()).
    a. TLPI 53.2.1
3. Open POSIX named semaphores are automatically closed when a process does an exec()?
    a. TLPI 53.2.2, pg 1093
4. The POSIX sem_wait() function:
    a. TLPI 53.3.1
5. If a *sem_timedwait()* (on unnamed POSIX semaphores) call times out without being able to decrement the semaphore, then the call fails with the error:
    a. TLPI 53.3.1
6. If a process attempts to decrease the value of a POSIX semaphore below 0, then, depending on the function used, the call _____
    a. TLPI 53.1
7. The sem_post(sem) function does what?
    a. TLPI 53.3.1
8. The POSIX semaphore operations can operate on multiple semaphores in a set.
    a. TLPI 53.3
9. If a blocked *sem_wait()* call is interrupted by a signal handler, then it
    a. TLPI 53.3.1
10. Like their System V counterparts, POSIX semaphores are a synchronization

mechanism, and a queuing mechanism.
    a. TLPI 53.3.2
11. The _____ function returns the current value of the semaphore.
    a. TLPI 53.3.3
12. The sem_destroy(sem) call is the preferred method to remove both named and unnamed semaphores from the system.
    a. TLPI 53.5
13. It is safe to call sem_destroy() on a semaphore even when processes or threads are waiting on it.
    a. TLPI 53.4.2
14. A POSIX unnamed semaphore can be embedded inside a data structure object.
    a. TLPI 53.4
15. The maximum number of POSIX semaphores that a process may have is (a name not a number):
    a. TLPI 53.6
16. What call marks a POSIX semaphore to be destroyed once all processes cease using it?
    a. TLPI 53.2.3
17. How are the System V semaphore operations different from the POSIX semaphore operations? (identify 2)
    a. TLPI 53.3
18. Describe what happens when a POSIX semaphore value is 0 (zero) and a call to sem_wait() is called on it.
    a. TLPI 53.3.1
19. If a blocked sem_wait() call is interrupted by a signal handler, then it ____
    a. TLPI 53.3.1
20. What function call is used to return the current value of a POSIX semaphore?
    a. TLPI 53.3.3
21. What 2 additional functions are required when using POSIX unnamed semaphores?
    a. TLPI 53.4

Chapter 54 – POSIX Shared Memory

1. POSIX shared memory allows to us to share a mapped region between unrelated processes without needing to create a corresponding mapped file.
    a. TLPI 54.1
2. On a Linux system (kernel 2.4 and higher), the POSIX shared memory is stored mounted as file system under:
    a. TLPI 54.1
3. On a Linux system, the POSIX shared memory objects will persist even after no processes has them open.

    a. TLPI 54.1
4. The POSIX call shm_open(), when successful, returns a
    a. TLPI 54.1
5. Kernel persistence for POSIX shared memory objects means that ___
    a. TPLI 54.4
6. The 2 methods to remove a POSIX shared memory object from a system are:
    a. TLPI 54.4
7. When a new POSIX shared memory object is created, it initially has ____ length.
    a. TLPI 54.2
8. When a shared memory object is extended, the newly added bytes are automatically initialized to -1
    a. TLPI 54.2
9. What are the 2 steps to use a POSIX shared memory object?
    a. TLPI 54.1
10. What function can be called to expand or shrink a POSIX shared memory object?
    a. TLPI 54.2
11. When a new shared memory object is created, its ownership and group ownership are taken from the ___
    a. TLPI 54.2
12. To obtain information about a POSIX shared memory object, we can call fstat() on it.
    a. TLPI 54.2
13. Why should references to locations in shared memory objects be calculated as offsets (rather than pointers)?
    a. TLPI 54.5

Chapter 56 – Sockets: Introduction

1. Sockets are a method of IPC that allow data to be exchanged between applications on different hosts connected by a network.
    a. TLPI 56.0
2. A successful call to socket() returns a
    a. TLPI 56.1
3. Modern Unix/Linux operating systems support at least the following socket communication domains (select all that apply)
    a. TLIP 56.1
4. Message boundaries are not preserved on stream sockets.
    a. TLPI 56.1
5. A stream socket is similar to using a pair of pipes to allow bidirectional communication between two applications.

    a. TLPI 56.1
6. When using datagram sockets, messages may arrive out of order, be duplicated, or not arrive at all.
    a. TLPI 56.1
7. Transmission Control Protocol (TCP) is an example of datagram sockets.
    a. TLPI 56.1
8. Socket I/O can be performed using the conventional *read()* and *write()* system calls.
    a. TLPI 56.1
9. A socket that has been marked to allow incoming connection by calling *listen()* is called
    a. TLPI 56.5
10. If there are no pending connections when *accept()* is called, the call blocks until a connection request arrives.
    a. TLPI 56.5.2
11. When *accept()* returns, it creates a new socket that is connected to the peer socket that performed a connect().
    a. TLPI 56.5.2
12. When a peer application reads from a socket which has been closed, it will receive:
    a. TLPI 56.5.6
13. A datagram socket doesn't need to be connected to another socket in order to be used.
    a. TLPI 56.1
14. With datagram sockets, message boundaries are preserved, but data transmission is not reliable.
    a. TLPI 56.1
15. By default, a socket that has been created using *socket()* is ___
    a. TLPI 56.5
16. In most applications that employ stream sockets, a server application performs the passive open, and the client performs the active open.
    a. TLPI 56.5
17. If an application attempts to write to a stream socket which the peer application has closed, it will _____
    a. TLPI 56.5.4
18. Since the underlying networking protocols may sometimes retransmit a data packet, the same datagram socket messages could arrive more than once.
    a. TLPI 56.6

Chapter 59 – Sockets: Internet Domains

1. A system that is called big endian is one where the most significant byte is stored first (MSB, leftmost)

    a. TLPI 59.2
2. The standard network byte ordering used is called
    a. TLPI 59.2
3. One difference between IPv4 addresses and IPv6 address is that an IPv6 address is 64 bits, not 32 bits like IPv4.
    a. TLPI 59.4
4. The result of a successful call to getaddrinfo() is
    a. TLPI 59.10.1
5. The function inet_aton() means ___
    a. TLPI 59.5
6. The *inet_pton()* and *inet_ntop()* functions are like *inet_aton()* and *inet_ntoa(),* but differ in that they also handle
    a. TLPI 59.5
7. With UDP sockets, if the incoming datagram would overflow the receiver's queue, then the datagram is silently dropped.
    a. TLPI 59.1

Chapter 63 – Sockets: Internet Domains

1. Because of the limitations of both non-blocking I/O and the use of multiple threads or processes, one of the following alternatives is often preferable (select all that apply):
    a. TLPI 63.1
2. The primary dis-advantage of the epoll API is:
    a. TLPI 63.1
3. The primary dis-advantage of the select() and poll() calls compared to the epoll API and signal-driven I/O is:
    a. TLPI 63.1
4. I/O multiplexing, signal-driven I/O, and *epoll* are all methods of achieving the same result, to see whether an I/O system call could be performed without blocking.
    a. TLPI 63.1
5. Level-triggered notification is:
    a. TLPI 63.1.1
6. Edge-triggered notification is:
    a. TLPI 63.1.1
7. Of I/O multiplexing (the select() and poll() system calls), signal driven I/O, and the epoll API, which can support both level-triggered and edge triggered notification?
    a. TLPI 63.1.1
8. When using level-triggered notification, an application can check the readiness of a file descriptor at any time.
    a. TLPI 63.1.1

9. When using edge-triggered notification, after notification of an I/O event, it is not normally necessary to perform as much I/O as possible.
    a. TLPI 63.1.1
10. The 2 system calls that can be used to perform I/O multiplexing are
    a. TLPI 63.2
11. The *select()* system call blocks until _____ (choose all that apply)
    a. TLPI 63.2.1
12. A *select()* system call contains file descriptor sets. Identify them:
    a. TLPI 63.2.1
13. The FS_SET() macro
    a. TLPI 63.2.12
14. On Linux, the maximum size of a file descriptor set (as defined by the FD_SETSIZE macro) is
    a. TLPI 63.2.1
15. When the timeout argument in a *select()* call is set to NULL, the call
    a. TLPI 63.2.1
16. If a call to *select()* returns the value 9, it means that
    a. TLPI 63.2.1