Thatcher Lane
CS 321
Assignment 3
02/23/18

1.20)

Fork()- In case of fork condition no memory allocation is there thus the resources are limited

Exec()- this could easily fail if the path name of the file that one wants to execute is incorrect, causing there to be more symbolic links within the translation of the path or the file

Unlink()- If the user does not have write permissions to the unlink file, or if the unlink file does not exist this could fail.

1.28)

When a system call is executed it puts the current running process on hold in order to begin the system call. The process that was previously running is saved to registers with its status being saved so that it can be resumed. In the case of system calls, after the execution of a system call, it is possible that instead of the earlier process, a new process can be started, resulting in further delays of the previous operation. With this in mind it is important for programmers to know which library wall will result in system calls so that they can prevent speed deterioration. So this means that it is important for programmers to know what will result in a system call because it could fundamentally change the logic of the program.

2.11)

A single thread process is incapable of forking while waiting for input from a keyboard. While waiting for input the process is put on hold, which prevents fork() from being used. After the input is received the process resumes and fork can be used. But because a fork cannot be created while the process is on hold it is impossible to have the same problem as we find in the multi thread program.

2.12)

If a user level thread were used for this process then the read operation being performed by this thread will block all the other threads from operating for the duration of this read, causing there to be no real implementation of multi threading. If we were to use kernel threads they would not effect or block all other threads from working, because they do not interact with user level threads. This means that unless all the kernel threads are blocked, other threads might continue operating. This would preserve the multi threading of the server. So it would be appropriate

for the person making this web server to implement kernel threads instead of user level threads.

2.13)

A single threaded server is better in situations like; a single assistance number for an area can keep the records of telephone numbers for a large number of people. The directory contains a number with a corresponding name. For the fastest access of the information, the information must be stored in the memory of the server. This means that it is more efficient to use single threading because when doing block read and write you must do them sequentially, which eliminates the need for multi threading.

2.15)

The main reason that a thread would ever voluntarily give up the CPU is that it's job depends on another thread. If thread #1 is waiting on thread #2 to do something, thread #1 will never be able to finish its job if thread #2 never receives the CPU. This means that thread #1 must call the thread_yeild() in order to finish its task.

2.28)

If the simulated multiprocessor is multi-programmed then it is very likely that a race condition could occur. A process, say processA is running and accesses a shared variable (x), and then a clock interrupt starts a new processB, which then access x also. If the accessing of the variable x does not happen in the correct order, then a race condition can occur.

2.32)

```
P(semaphore){
        mutex = down()
        Count = count -1
        If(count>=0){
                Mutex=up()
                return
        }
        mutex = up()
        blocking = down()
}

V(semaphore){
        mutex = down()
        Count = count+1
        If(count>0){
```

```
            Mutex = up()
            Return
        }
        blocking = up()
        mutex = up()
}
```