2 (10 minutes). In Linux, does an older process (i.e., a process with a larger elapsed time) always have a lower process-ID? If so, explain why; if not, give a scenario illustrating a counterexample.

2. Not always,
    a. Counter example:If it wraps around then not necessarily

3 (10 minutes). Is it possible for a Linux-based multiprocess application to have deadlock and livelock simultaneously? If so, explain how it could occur; if not, explain why it's not possible.

1. Deadlock, circular dependency on waiting
    a. at least two processes waiting, and holding a part of the resources
    b. Live lock:
    c. Is it possible to have these two things simultaneously
    d. key: linux-based multiprocess
        i. 2 pairs of processes
            1. 1st pair is dead locked, second pair has nothing to do with first pair has a livelock

4 (15 minutes). Is it straightforward to implement the POSIX 'link', 'unlink', and 'rename' system calls on a FAT32 file system? For each of the three system calls, explain how or when you can easily implement the call, or explain why or when it can't easily be done.

1. POSIX link/unlinnk/rename is based on inodes, FAT32 file system does not have inodes can we implement these three things
    a. What does a POSIX link does? refers to a hard lock, you have existing inode that represents a file that wants to access another file, you increase the link count in the file that is being accessed
    b. FAT, a file can only be in one place and you don't have inode. each file is a file and no associated inodes which means there is no way to use a common place to hold the number of links that a file has
    c. for Link: NO
    d. What about Unlink: it is basicaly deleting the file so is can be implemented
    e. Rename: in inode, you change the dir-name in the directory. Within the same directory you have to change the name field because the file is only in one directory

5 (15 minutes). The FAT32 file system stores file time stamps like this:

field name	value
DIR_CrtDate	file creation date
DIR_CrtTime	file creation time
DIR_CrtTimeTenth	file creation fractional time (tenths of a second)
DIR_WrtDate	date of last write
DIR_WrtTime	time of last write
DIR_LstAccDate	date of last access

In contrast, the Solaris tmpfs file system (which is a RAM-based file system) stores inode change time, last-modification time, and last-access time each to a resolution of 1 nanosecond. Give arguments for why the FAT32 variable-resolution time stamp approach might lead to better overall performance for an application. Conversely, give arguments for why the Solaris approach, with uniformly high resolution, might lead to better performance overall.

1. FAT32 file system with time stamps
    a. Pros for FAT32
        i. Lots of fields to fill for FAT
        ii. Why less frequent update on time gives better performance
            1. You write to a lot within a second, you just to change write time once
                a. in Solaris file system you have to update 5 times
            2. Don't have to update that frequently
        iii. Cache can keep track of file dates so look up is really easy and you don't have to write to the disk (which takes a lot of time) frequently
    b. Pros for Solaris

i. Not that good: writing a lot of files means less fields than FAT32 needs to fill
ii. If you want to synchronize files
iii. Makefile needs to compile the source file with the o file. If the source file is newer you want to update the o file, if it is older than the object file you don't need to update it
    1. If they're made at the same time then just update
    2. If we use a dating timing system, then we have to update the entire o file every single time we run make

6 (20 minutes). Suppose we are implementing the 'cat' command, which simply copies standard input to standard output, and we want to optimize the special case where standard input and standard output are both pipes. In this special case is there any way that 'cat' can connect the input pipe directly to the output pipe and then exit, so that the 'cat' process itself need not constantly issue 'read' and 'write' system calls but can rely on the kernel to move the data from the source to the destination? If so, give the system calls that 'cat' should call to arrange this. If not, explain why not, and explain how you'd change the Linux kernel to support this optimization (or if such a change wouldn't be easy, explain why not).

1. Can we glue two pipes together so we don't have to use cat
    a. Do current Linux systems support this?
        i. No, so why?
            1. Expensive, because kernel has to manage both pipes?
                a. No, process managing the pipes is more expensive
            2. Why do we want to do this in the beginning?
                a. Copy things from source to destination
                    i. you can just use one pipe
            3. There is no demand for this, example would be that the main purpose is to copy one thing from source to destination we can use one pipe.
            4. Other disadvantages?
            5. What if we need this? How can we optimize it to work better? What are the benefits of having two pipes glued together?
                a. you have two pipes, so you need two locks to prevent race condition
                b. if they're glued together, then it saves on one lock

1. The 10 page tables entries point to page tables themselves then they point to others
    a.

10	10	12
used		
points down there	used	
		used
	down here!	LOTS OF STUFF

    b. PAE, uses a three level way uses 64 bit address

8 (20 minutes). The indirect blocks of Unix System V file systems seem to have a function similar to that of the intermediate page directories in multiple-level address maps. Explain the main differences in implementation goals and strategies between the two schemes.

1. Indirect blocks of Unix System V file systems
    a. Indirect blocks of Unix System V file
        i. You can waste indirect pointers and its not a big deal
        ii. Small files can be stored with 10 blocks only... good utilization
        iii. we usually put the meta data of the file in the beginning
            1. very quickly access those
            2. Looking for high resolution pictures, then we can just read the first blocks of the file and see if it has it, if it doesn't then we can skip it
    b. Intermediate page directories in multiple-level address maps (one is virtualization don't know which one)

10 (24 minutes). Explain how access is controlled to the entities identified by the following values in Linux user programs. In particular, explain what happens when an invalid access is attempted via one of these values.

File descriptors
File names
Inode numbers
I/O buffer addresses
Process IDs
Thread IDs

1. File Descriptor: token / handler that represents a file that we want to read or write to
    a. lives in the proccess file table, lives in the kernel and the user cannot access this, so file control
2. File names
    a. we have a directory dir drwxr-xr-x (underlined means we can still see things that we can't work with"
    b. r means can access, w can change the files inside, and x means we can read (see it)
        i. file1 -rwxr-xr-x
        ii. file2
    c. how do we access file 1,
    d. we need linux file access permisions
        i. drwxr-xr-x for directory
        ii. for files
        iii. -rwxr-xr-x
        iv. if we want to use file 1, the access is not granted by -rwxr-xr-x
            1. we can only see "-rwxr-xr-x" if we "ls file 1"
            2. but if we ls file1 then we already have access to file name
        v. If we want to use the file1, then access is granted by the rights of the directory
            1. this is just accessing the file name, not changing
3. Inode numbers
    a. can a user open an arbitrary inode number
        i. no, linux doesn't support accessing a file using the inode number, kernel does not let you do this
4. I/O buffer addresses
    a. Input buffer: when you type, the input goes into a buffer and the kernel reads it
    b. Output buffer: when you want to print your information goes into a output buffer and goes out together
    c. I/O buffer exists in the kernel, user prorams don't know how to attach to that buffer
5. Process IDs
    a. With an inode number you can't do anything, but with process ID, you can do something
        i. like send a signal to the process...
        ii. can you send a signal to an arbitrary process?
            1. send kill signal to arbitrary processes
            2. You can send kill signals to an arbitrary PID but the kernel will probably seize it
            3. but sending a kill signal to a child process is fine
6. Thread IDs
    a. This time we are in the same process
    b. There is no protection against threads joining each other within the same process, but there is a boundaries between thread IDs you can't act as other threads

But rather we should talk about the Cloud computing security in the last lecture

- a. If you have a virtual machine in the cloud and you want to access it, it is possible that people spoof as that virtual machine. You think you are giving information to the correct one, but actually sending to a malicious one. Open ended questions
- b. Big problem with huge weighting on cloud computing?
- c. Not sure about the real problem, but wanted to test about cloud computing