CS350 Final Review

For what reasons might a process fail to open a file?

1. The file does not exist.
2. The user process does not have the correct permissions.
3. The filename format is incorrect.
4. The file is empty.

For what reasons might a process fail to open a file?

1. The file does not exist.
2. The user process does not have the correct permissions.
3. The filename format is incorrect.
4. The file is empty.

1. open will fail if you do not use the option O_CREAT (File Systems 9).
   Note: fopen can fail with "r" or "r+".
2. Check the permissions before opening the file (File Systems 29).
3. Read the directory's data block to see if the file exists (File Systems 28).

A 5400rpm disk has 2^10 tracks with capacity 2^32 bytes. Each track has 2^8 sectors.   Maximum seek time is 20ms.

1. What is the capacity of each track?
2. What is the capacity of each sector?
3. How long does it take for a full rotation of the disk?
4. How long does it take to read 10 consecutive sectors?
5. How long does it take to seek 400 tracks away from the current head position?

A 5400rpm disk has 2^10 tracks with capacity 2^32 bytes. Each track has 2^8 sectors.   Maximum seek time is 20ms.

1. What is the capacity of each track? 2^32/2^10 = 2^22 bytes
2. What is the capacity of each sector? 2^22/2^8 = 2^14 bytes
3. How long does it take for a full rotation of the disk?

   5400/60 = 1/x -> x = 11.1ms

   See IO 15-16

4. How long does it take to read 10 consecutive sectors?

   (10/2^8)11.1ms = 0.436ms

   (faction of full rotation) x (time for full rotation)

5. How long does it take to seek 400 tracks away from the current head position?

   400/2^10*20ms = 7.8125ms

   (fraction of tracks) x (max seek time)

Enumerate the disk operations when executing “ls /usr/bin/local”. Assume local is a directory.

Assumption:

When you read from a directory you do not update its last access time.

Enumerate the disk operations when executing “ls /usr/bin/local”. Assume local is a directory.

1. Read / i-node
2. Read / data block to retrieve usr i-number
3. Read usr i-node
4. Read usr data block to retrieve bin i-number
5. Read bin i-node
6. Read bin data block to retrieve local i-number
7. Read local i-node
8. Read local data block

File Systems slide 28 (repeated twice).

Give one advantage and one disadvantage of chaining.

Give one advantage and one disadvantage of chaining.

Advantages: simple, easy to implement. Works well for sequential file access.

Disadvantages: very slow non-sequential access.

File Systems slide 36.

Give a pseudocode implementation of a function that adds two global arrays (A+B=C) of the same length such that there is one thread for each element of the array.

Give a pseudocode implementation of a function that adds two global arrays (A+B=C) of the same length such that there is one thread for each element of the array.

```
void AddKernel( void * data, long int idx )

{

        C[idx] = A[idx] + B[idx];

}

void SumArrays()

{

        For i = 1 to length(A)

                thread_fork( “add”, null, AddKernel, null, i );

}
```

Since only the ith thread would access C[i], locks are not needed. See Concurrency slide 11.

Suppose MIPS had an atomic instruction SwapNotEqual such that:

```
int SwapNotEqual( int &x, int y )
{
    If ( x != y )
        Int tmp = x;
        x = y;
        return tmp;
    return y;
}
```

Similar to x86's xchg (Synch slide 16) except
1. order of arguments different
2. uses a reference rather than a pointer
3. swap only if different

Implement a spinlock using SwapNotEqual.

Suppose MIPS had an atomic instruction SwapNotEqual such that:
Implement a spinlock using SwapNotEqual.

int lock = 1; // lock is free, let 0 mean the lock is NOT free.

```
While ( !SwapNotEqual( lock, 0 ) ) {}
```

Similar to x86's xchg (Synch slide 16) except
1. true means unlocked

Reason:

If lock = 1; then SwapNotEqual(1, 0) sets lock to 0 and returns 1. !SNE = 0, so no more looping.

If lock = 0; then SwapNotEqual(0, 0) returns 0. !SNE = 1, so continue looping.

The Linux Completely Fair Scheduler assigns each thread a weight that is proportional to what share of CPU time the thread should receive. Suppose two threads A and B have weights 10 and 11 respectively.   A has an actual runtime of 5 and B has an actual runtime of 10. If the total thread weight is 100, which of thread A or B will run next?

The Linux Completely Fair Scheduler assigns each thread a weight that is proportional to what share of CPU time the thread should receive. Suppose two threads A and B have weights 10 and 11 respectively. A has an actual runtime of 5 and B has an actual runtime of 10. If the total thread weight is 100, which of thread A or B will run next?

Virtual Runtime Thread A = AR(TOTAL/W) = 5(100/10) = 50

Virtual Runtime Thread B = AR(TOTAL/W) = 10(100/11) = 90.9

Since VR(A) < VR(B) Thread A will run next.

See Scheduling slides 21-23.

Long-running, non-interactive computation threads that do not block will filter down to the lowest priority queue in the multi-level feedback queue scheduling algorithm. If there are significantly more high-priority and blocking threads running, these lower priority threads may not run. How does the multi-level feedback queue method ensure these threads do not starve?

Long-running, non-interactive computation threads that do not block will filter down to the lowest priority queue in the multi-level feedback queue scheduling algorithm. If there are significantly more high-priority and blocking threads running, these lower priority threads may not run. How does the multi-level feedback queue method ensure these threads do not starve?

Periodically takes all threads in all queues and puts them into the highest-priority queue.

Solaris does this once per second.

Scheduling Slide 10.

What are the three primary types of device registers?

What are the three primary types of device registers?

1. Command
2. Status
3. data

See Devices and I/O slide 3.

A prankster placed an easter egg in a major OS. If you type “this os sucks” at the command prompt a kernel-privilege program executes and V’s every semaphore in the kernel exactly once. Does the program succeed in V’ing every semaphore, or, does it panic when it tries to V? Why?

A prankster placed an easter egg in a major OS. If you type “this os sucks” at the command prompt a kernel-privilege program executes and V’s every semaphore in the kernel exactly once. Does the program succeed in V’ing every semaphore, or, does it panic when it tries to V? Why?

The code succeeds in V’ing every semaphore because semaphores do not store/check if the thread that P’d is the thread that V’d.

See Sychronization slides 28, 32.9

A device driver writes some data to a data register and then writes to a command register. When the device is done the requested operation the OS needs to acknowledge the completion before any other threads can use the device. Give the pseudocode for the driver and handler.

A device driver writes some data to a data register and then writes to a command register. When the device is done the requested operation the OS needs to acknowledge the completion before any other threads can use the device. Give the pseudocode for the driver and handler.

```
Driver
    P( device semaphore )
    Write to data register
    Write to command register
Handler
    Acknowledge completion to device
    V( device semaphore )
```

See slide 7 in Devices and I/O.

On-demand paging only loads program data into the address space when it is needed. When is this detected?

If we use the Clock Replacement Algorithm for pages, what must the kernel do when choosing a victim?

On-demand paging only loads program data into the address space when it is needed. When is this detected?

If we use the Clock Replacement Algorithm for pages, what must the kernel do when choosing a victim?

- Detected on TLB miss

We only mentioned on-demand paging briefly this term, Virtual Memory slides 42 and 53. You could have a bit in each page table entry to track if the page has been loading into RAM or not.

- Until a victim is found (an entry whose use bit is not set); kernel sets the use bit of non-victims to 0

See line 2 of the pseudocode on Virtual Memory slide 59.

A system uses 48bit virtual addresses and 64bit physical addresses. Pages are 64kb (2^16) in size.

1. What is the maximum size of physical and virtual memory?
2. How many pages are there in virtual memory? How many frames in physical memory?
3. How many bits are needed for the page offset?
4. If a PTE is 16 bytes (2^4), how many fit on a page?
5. How many levels are needed for the multi-level page table if each table must fit onto a page?

A system uses 48bit virtual addresses and 64bit physical addresses. Pages are 64kb (2^16) in size.

1. What is the maximum size of physical and virtual memory?

   2^64 bytes, 2^48 bytes

   See slide 2 of Virtual Mem.

2. How many pages are there in virtual memory? How many frames in physical memory?

   Pages = 2^48/2^16 = 2^32; Frames = 2^64/2^16 = 2^48

   See slide 20.1 of Virtual Mem

3. How many bits are needed for the page offset? 16

   See slide 16 of Virtual Mem.

4. If a PTE is 16 bytes (2^4), how many fit on a page? 2^16/2^4 = 2^12

   See slide 31 of Virtual Mem.

5. How many levels are needed for the multi-level page table if each table must fit onto a page?

   (48-16)/12 = 32/12 -> 3 levels [8 bit directory index][12 bit page index][12 bit page index][16 bit offset]

This is from a class exercise but take
(number of bit in a virtual address) - (the bits on the offset) = bits for page number
ceiling ((bits for page number) / (PTE per page)) = levels, small one directory, rest page tables