**Sept. 5**

Multitasking - Ch 1 & 2

-Batch processing

-Multitasking

Three Design Issues

1. System Boot

-OS is just a set of programs, how does it initially start?

Bootstrap program(loader) locates OS kernel and moves it into main memory to start execution.

Bootstrap program is in ROM; needs no initialization, cannot easily be infected.

Bootstrap runs diagnostics, initializes registers (but not the IR) and other stuff, and starts the OS

1. Protecting OS from apps

-How to prevent apps from corrupting OS?

Processors include a hardware mode bit to identify if the system is in *user* or *kernel* mode

Trap instruction changes signifying bit from user to supervisor/kernel mode

1. System call API

**Sept 7**

*Recap:*

-System boot stages: POST>BIOS>MBR-Primitive Loader>Secondary stage boot loader

-Processor mode bit; changed w/ trap instruction (from 1/user mode to 0/kernel and back to 1)

-Trap table

Classes of Exceptions

CPU interrupts can be caused by traps (intentional; i.e sys calls) faults(potentially recoverable error; i.e. div by zero) Hardware interrupts(sig from I/O) and aborts(nonrecoverable; i.e. disk crash)

**Sept 12th**

Recap:

IO devices consist of mechanical components and device controllers

Device controller starts: Idle, Working, Busy

Three IO strategies: Direct IO w/ polling (CPU waits for device to be idle to give cmd, and waits for completion), Direct IO w/ interrupts(doesn’t wait to give cmd), and DMA w/ interrupts.

DMA = Direct Memory Access, bypasses CPU for data transfers; adds specialized DMA controller (like a simple processor) to operate the memory bus w/o the help of the main CPU

/proc contains files that represent the state of the kernel; allows apps and users to peer into the kernel’s view of the system.

/proc is a virtual file system

**Sept 14th.**

Reading , CH 1, 2, 13, Moodle documents on filesystem and LKM

Standard C library is not available on kernel mode; use printk() instead of printf()

dmesg (display message or driver message) prints buffer of the kernel

-Loadable kernel modules: Win = kernel-mode driver, Linux = LKM, VM = DKM

-LKMs can be loaded and unloaded from kernel on demand at runtime; dangerous because they can add third party code to an already-running kernel. Device drivers are a major source of errors in operating systems because of this. See */lib/modules* for all the LKMs or use *lsmod*

-Kernel modules are written in C; and they must be used with the same kernel they are built for

-Key operations:

int init\_module(void)

~stuff is initialized here on loadup, returns 0 if all good

Void cleanup\_module(void)

~free up resources here on unload

Hello World example:

#include stuff

MODULE\_AUTHOR(...)

MODULE\_LICENCE(...)

int init \_module(void){ printk(KERN\_ALERT “helloworld”);return 0;}

void cleanup\_module(void){printk(KERN\_ALERT “byedudes”);return 0;}

-Building kernel module:

obj-$(CONFIG\_FOO)+=foo.o

~This tells kbuild that there’s an object called foo built from foo.c (or foo.s)

-Loading kernel module: insmod

Sudo insmod helloworld.ko

insmod loads LKM into kernel and invokes its initialization routine (also called init\_module)

-Unload the module with rmmod (“rmmod helloworld.ko”) which would print “byedudes”

-Dependent modules are automatically loaded or unloaded.

Assignment 2 will consist of writing an LKM to act as a device driver.

**Recap:**

Difference between threads and processes.

IPC Message Passing

Pipe, stream of bytes, piped[0] read end, piper[1] write end

Anon pipes only exist between 2 processes, named pipes persist across processes

**Readers Writers problem**

Writers & readers can’t access database at the same time

Semaphores

Typedef struct{

Int val, PID \*list[]

} semaphore

Use it with wait() and signal() to sleep and wakeup processes.

writer(){

wait(s)

Write db -critical section-

signal(s)

}

Reader is similar but there can be multiple readers at the same time.

Only call wait for reader if there is a write process

Int rc =0

reader(){

If rc==0 wait(db)

rc++

//read

rc--

**Midterm Review**

Syntax

-Drawing parse trees from sentences, is the grammar ambiguous or not (ambiguous if more than one parse tree)

-Given a grammar and given a sentence, does that grammar parse that sentence?

Evaluation

-Judgements that are defined by inference rules

-Making derivation trees from rules

**Lecture slides**

Before multiprogramming, OS’s used a form of sequential execution called \_\_\_\_

>batch programming

Switching CPU from one program to another is a \_\_\_\_\_\_

>Context switch

Matching: Cooperative multitasking, Preemptive multitasking

1)Programs quickly and voluntarily yield CPU before they’re done.

2)OS forces programs to give up CPU

How does the OS force rapid switching?

>Periodic use of timer interrupts to transfer control back to the CPU so a scheduler can be used to determine the next program to run.

What are the first and second procedures that run during startup?

>The primitive loader, then the secondary stage bootloader

What is the value of the supervisor mode bit?

>0

What is the processor mode bit set to when the trap() instruction is done executing?

>1

What are the three states of a device controller?

>Working, Idle, and Finished

How much space do the virtual files in the /proc/ directory take up?

>Listed as zero byte size, but actually contain a lot of information

What is used to represent the processes of the OS?

>Process Control Block

Test Semaphore Problem - Trains!

Trains!

Hello, fellow railroad engineer. All our trains run on computers now so we need to implement some way to coordinate trains that go through that one-way tunnel over there.

==\ A B /===

WEST === A ============ B==== EAST

As you can see the tracks merge in the middle so trains can’t be going through at the same time or we’ll all die.

Trains must wait at points A and B until the track is clear. Trains coming from the WEST are called TW, trains coming from the EAST are called TE.

Trains request access to the track when they arrive at points A or B. TW must wait until A signals that the tracks are free for them to use. TE must wait until B signals the track is free fro them to use. Now get to coding, code monkey!

####SOLUTION####

(Disclaimer: may be partially or wholly incorrect)

Semaphore A\_Sem = 0, B\_Sem = 0 //Tracks are initially set as open b/c no trains are present

Void request\_A(){

Wait(A\_Sem);

}

Void release\_A(){

Signal(A\_Sem);

}

Void request\_B(){

Wait(B\_Sem);

}

Void release\_b(){

Signal(B\_Sem);

}

**Programming Assignment 4**

*Readings: Chapters 1 - 13*

-Indexed allocation vs File Allocation Table

-Indexed has faster random access

-How large should the index block be?

---Large file > large index block; paying price for smaller files

-Multilevel indexed allocation: top level index block points to second level index block, etc.

---has the advantage of smaller files not needing to allocate excess space.

---accesing smaller files takes just as long as big files.

-Making changes to filesystem:

---uses temp storage in secondary storage

---Asynchronous writes get saved in buffer and then pushed to the disk

------This removes removes disk I/O wait time, allows disk to schedule writes more efficiently

---Synchronous writes are used when response is expected immediately

-Fault recovery

---consistency checker fsck in UNIX, chkdsk in MSDOS; check that overall structure of filesystem is correct

---In the case of a file system failure: log-based recovery

------References OS log of all operations on filesystem.

------Journal mode(slowest, most verbose), Ordered mode(only metadata logged, not file data, common Linux default), Writeback mode(only metadata, not guarantee that file data is written

before file data, fastest, least safe)

**Review Material for the Final Exam**

*December 14th | Exam is Monday Dec. 18 from 4:30 - 6:00 PM*

Contents

Memory Management; lec set 12 - 15

-What are Base and limit registers?

-How does Address Binding work? Know different types (compile time, load time, run time)

-Memory Allocation (First (faster), worst (slower, less fragmentation), best (slower, more fragmentation), next fit)

-Page Table Implementation (hierarchical (each process has own pg tbl), inverted)

-Virtual Memory(locality(ps is likely to access page again), handling page faults)

-Working set theory(use thresholds to figure out how many frames to allocate)

Mass Storage; lec set 16

-Disk Structure(magnetic disks with platters, tracks, sectors, and R/W arm, three components of access time)

-Algorithms to minimize seek time(FCFS, SSTF(go to the closest sector), SCAN/LOOK(elevator algorithm), C-SCAN(move in one direction, goes all the way back when done \*services outer layers just as much as inner layers, unlike SCAN\*))

-Flash memory characteristics

-RAID characteristics

File System; lec set 17

-Sharing files(symbolic links)

-Mounting filesystems

-Virtual File Systems(Allows user to access files from different types of filesystems)

-Implementation(make use of secondary storage and main memory(as a cache). Use of system-wide OFT(contains one entry for each file that has been opened), per-process OFT(has a file table for each process), and cache directory(directory cached in memory for when it is very large). Use of system calls such as open, read, close, write, seek)

-File allocation; how to decide where to store a file on a disk and what type of allocation to use(contiguous(has problem of handling slow growth), linked(uses pointer to connect disk blocks, but corruption can ruin all links and random access is expensive), and FAT(keeps disk block pointers in table))

Security and Protection; lec set 18, 19

Virtual Machines; lec set 20

-Page replacement algorithms: FIFO, OPT, LRU

-Calculating access time; TLB