OS Course

Erick fredj – professor at Rutgers

Call him eric

Nachum is the TA

Syllabus in Hebrew

Use multiple sources of info and combine the best

16 lectures

Books

Use the newest versions of the silberchatz book

And the linux kernel book too

- For dev and research

Red hat

Etc

Wsl 2 – windows ubuntu vm

Homework is 5% - **mandatory**

Final exam is 85%

Mid term exam is 10%

Try more than 80% average… prerequisites enforced, exercises pretense mandatory

---

He is writing the midterm – Nachum writes moed a and another teach writes the 2nd exam

Midterm is to help us

Understand , not just repeat

Os (2019) slide

Benefits of certain OSes over others

Linux is OSS – can observe and read the code

Windows is totally closed

OS programming

- How to program and use the OS in programming
    - From all the languages we can apply and optimize software and design
- Know how to use the OS = more efficient programming

Parallel OS and storage parallel OSes = state of the art OS

DEO – department of energy

- IO library adios 2
- Exa scale computers
    - 1000's of CPUS
        - Each cpu has 64 nodes etc
        - And GPUS
            - And combining nodes together (if we have time)

Nuclear bomb sims

6g and 7g connections – manage all info quickly

Boats need computing too

Manage production of electricity

Structures:

- Lectures
    - Basic info
    - Examples form unix
- Ex
    - Linux os code (not applied) – only theoretic exercises not actual ones
- HW
    - Program with kernel of OS and course, dry exc… ask the TA
    - Prove knowledge – **think and work alone**

What is an OS?

- Software layer for managing and hiding computer hardware details
    - Way to get to hardware easily
- Provide apps with dedicated & powerful VM abstraction CS is art of abstraction
    - OS is a layer – abstract
    - Illusion
- Manage system resources and share them between processes, programs, and users
    - OS is a manager too
    - User can be another computer/machine/device

Good manager – we don't feel like someone manages us

If it interferes between programs and hardware = not good manager

Friendly and efficient

Should be smooth and transparent

Role of OS

User layer = role to run the app

- Use hardware as much as possible w/o problems

Ensure correctness

- Mem limits
    - Each app has limit of memory
    - Don't want to mess
        - Pinny getting to my memory
- Priorities
    - Most important apps to run
    - My app is more important than eden's
- Stability
    - Sure the system runs correct and stable
    - Nothing missing or no issues

Convenience

- Hide details
    - Hide hardware layer
- Coordination
- Sys calls
- File system
    - Storage , where we store
    - Information

Management needs correct - be sure made right decision

Allow user to run app without taking care of difficult management

If someone takes care of everything, we need to ask for permission to use memory pieces for example

- Can destroy optimization/speed up of apps

Be careful not to be too strict, because nothing will run

Windows ME – on market for 1 week

- Best and worst OS
- Slow , nothing running correctly

App wants all resources:

- CPU time
- Memory
- Files
- I/O
    - Storage
    - Input/output

- Clock
    - Clock cycle of computer

Ram, storage, USB, etc

OS gives app **illusion** of a complete system of its own

- Exclusivity

Sharing a blanket in the winter

- Each app wants the blanket for itself

CPU = blanket

Shared resources , both people think they have the system themselves

DOS OS – all program 1 app to run and other has no right to run

Realtime system

- Each app can have it at anytime to need
- That's a different approach

Run a bunch of apps…

Development of OS

- Batch jobs – bunch of ops running one after another
- Mainframe computing
- IBM 24x7 IBM s/360

.bat file

Define a bunch of ops or tasks one after another

- Check for updates for apps

Another one to turn on projectors

.sh for linux

Newer os, first lime in parallel

2nd approach

- Timesharing expensive and fast hardware, cheap manpower
    - Interactive time sharing: **unix**
        - Illusion that running at same time
        - Multi-user operations
            - Each had a piece of cpu time

Cheap and slow hardware – expensive manpower

- PC – used ms dos , windows
    - Each app run and used all resources, finished then next app
        - Windows uses multi user with time sharing approach

Every move, gives us illusion of all windows working at same time

Present and future

- Cheap hardware, great computer power
    - Windows NT - OS/2 (mac and linux)
    - Multi-cores
        - Multiple tasks at cpu
        - parralelization (no longer illusion)
    - Basic resource sharing
        - Disks
        - Printer

SIMD

- Single Instruction Multiple Data
- Vector multiplication
- 1 clock , multiply 2 vectors

MIMD

- Multiple Instruction Multiple Data
- Matrix multiplication (plus/multiply)

2 configs for computers and important for new developments

High speed networking

- Grid computing
    - Connect each laptop to high-speed network and broadcast tasks for each computer
    - Parallel computers
    - Matrix multiplication or even dot product of vector
        - Basics in graphics, rotation of picture with pixels
            - Define image and compose elsewhere and each domain performs rotation
                - Gets subdomain
- Web storage
    - Store code in cloud
        - Save in parallel on different devices

Optimization

- Go deep in order to get reason/cause of delay or speedup of program

Near future

- Virtualization
    - Disconnect OS from hardware
    - Multiple VMs on physical machine
        - Need a OS on each VM
            - Can even create a grid of VMS and run them in parallel on same task
    - Integrated high speed networks
        - Cloud computer
            - High demand
        - HPC – high performance computation
            - Simulations of cars, planes, new designs of medicine
    - Dedicated OS – software apps
        - On specific subject
            - Not very demanding but performs specific ops
            - Washing machine
                - Runs a small computer with a manager (OS) – dedicated to perform specific task that is optimized
            - Cars have it too
- Minimization and assimilation
    - Mobile phone as computer system, netbooks
    - Computing everywhere – pervasive / ubiquitous computing
    - Phones don't need a desktop OS
        - Android OS
            - Specific tasks and management, don't need everything

Fuzzy logic approach

- Byte operations with electricity
    - 1/0 manipulations
        - t/f
    - we use maybes'
        - not certain – ubiquitous – quantum computing
            - uncertainty on calculation
            - also AI certain level of knowledge , not T/F

Computer Structure

- OS behavior dictated by hardware on which its running
    - Command set, special components

Hardware simplifies or complicates OS tasks

- Old computers didn't support virtual memory
- Modern computers contain multi-core and multi process support
    - Right hardware to manage

Computer price relies on hardware and if its smart or not

OS simplify **management** of hardware – needs info to do so

Hardware mechanics for OS

- Hardware clock
- Atomic sync operations
    - Hardware helps OS to perform specific task and isolates access of other tasks from same thread
    - Syncs different apps at same time
    - Hardware – busy don't disturb
    - Or smart algo to give knowledge about piece being busy
- Interrupts
    - Signal produced by hardware telling OS, ready for info or to give
- Sys calls
    - What is System Call in Operating System? A system call is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.Sep 10, 2022
- IO control ops
    - Printer or file, know if access file or not
- Mem protection
    - This mem belongs to certain task
- Secure
    - Don't want app to be mingled with mem of another app at same time
    - Access home directory, don't mess with different users
- Protected actions
    - OS does action, should be protected
        - Can perform OP or not
            - Moving info from one place to another in memory

Proected commands

- OS is middleware
- Some machine commands only allowed for OS
    - Access I/O
        - Disks
        - Network cards
        - Printer
    - Modification of memory access data structures
        - Page table
            -
        - TLB
            - Cache
            - Table look buffer
            - Access memory fast and chose right mem

- Update special modes
  - Status for interrupt handling priority
    - Different layers (hardware, app, etc)
      - User = app layer = user role / mode
      - OS – hidden layer = user doesn't have direct access
        - Uses sys call to perform hidden commands that only OS can perform
      - Move form user to kernel/OS mode
        - Need to specify where the task/worker is sitting
          - Way to jump from one to another
          - In user or kernel (special command on Io for example)
            - Higher priority than the user mode
              - Every kernel task will run on cpu before user task
    - Halt command
      - Stopping
      - Shut down computer
      - OS performs specific operation to do so

Work in protected env

- Arch support 2 situations
  - Kernel
    - Protected ops
  - User
    - Space for user
- Status is saved with protected specific register as status bit
  - Specific value for kernel or Os mode
  - User programs in user
  - Os in kernel
  - Status = 0 for user mode

Processer executes protected commands only in kernel mode

1+2 op

Form user to kernel mode, kernel performs the ops in that mode

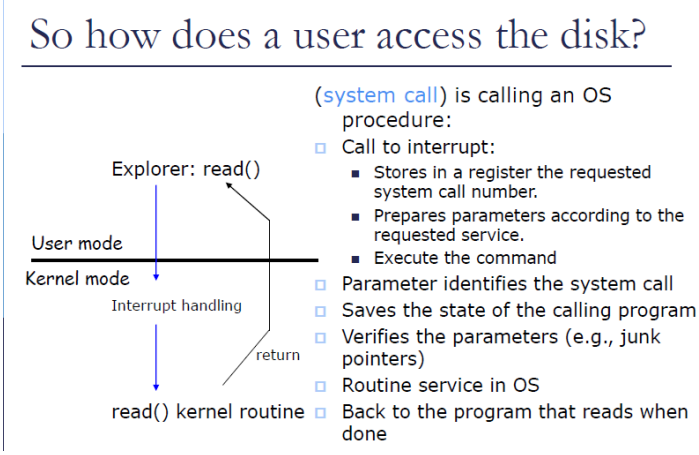
Loads 1 in one register and 2 in other then performs operation them gives info back to user

Back and forth between user and kernel modes

- Uses hardware easily

Modes

Read – system call – interrupt to move information



Move this system call – in user mode in code and will allow us via interrupt kep by OS and performs and executes request from application

Interrupt – stops the os and tells it what to do (hardare)

OS active software responding to interrupts

Interrupts guide actions for the OS – how it treats requests

Read in kernel reaches hardware and extracts info then returns to read call in user mode

Read calls fn in kernel mode

Talked about status

- Register saving status in user mode – 0 in register
- Then applied from user to kernel – status becomes 1
    - No catch from interup and apply read op in kernel mode
        - Management is back and forth in code

Example: Linux on Intel processors

- Initialization:
    - Interrupt Descriptor Table (IDT) with handles for each type of interrupt.
    - Vector 128 (=0x80) corresponds to system calls.
    - Kernel has priority 0, priority user code 3.
- IDT entry, corresponding to vector 128, contains:
    - Pointer to a kernel code that handles system calls.
    - Approvals call with priority 3 to initiate a call to the interrupt.
- In performing system call, user process
    - Stores in a register ***eax*** the requested system call number:
    - Prepares additional parameters according to the requested service.
    - Executes "int 0x80" command (software initiated interrupt).

Interrupt table

- Vector on 128 interupts – basic ones to catch
- Challenge for us
    - Dec to hex and binary
        - Basic info
        - **Do the operations by ourselves no calculator**
            - Important for test

Apps and interrupts are high priority – run in kernel mode

Eax is the name of register for intel

- Location and name
- Other CPUS differ
- Run and execute whats in eax

protect parallel tasks and information

Hackers can access our program then do what they want in the program

Protect different users from different programs

- Word running – same program by 2 users

Register for each program

- Each program defined by base and limit address
- Info between those address belong to that specific program

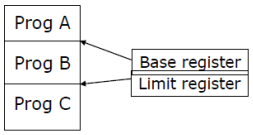
Basic protection

Virtual memory

- Another form of protection

Memory Protection

- The OS needs to protect user program from each other (with or without malicious intent)
- The OS needs to protect itself from user programs
    - And about the programs that use it.
- Simple method: base register, limit register for each program.
    - Self protected.
- Virtual memory



The Operating System lifetime

- Besides the boot the system, get into the kernel just for an event.
- The kernel defines how each event is handled
    - Part is determined by the processor.
- Interrupts and Exceptions:
    - Interrupts are caused by hardware components (clocks, end / end)
    - Exceptions (trap) come from the software (explicit command, page fault)

boot of system – OS doesn't run, the boot is meant to run the program to load the OS

That's the lifetime of the program

Boot system then dies

Kernel can write to interrupts, manage and catch different requests from device

Hardware interrupts – act or start, end an operation

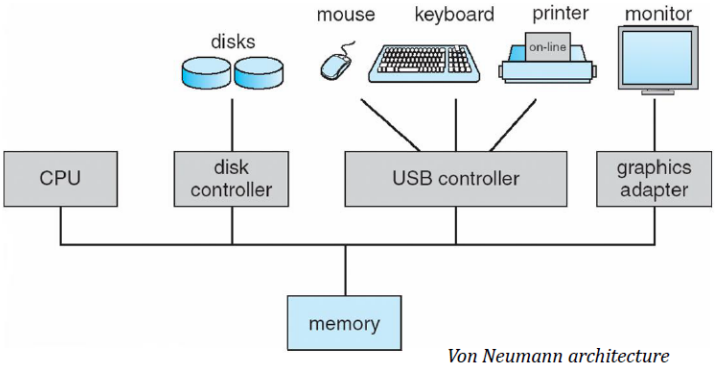
Or exception – interrupt from the software – not specifically from hardware

- Any program have exceptions is software related
- Trap = exception for unix

Example:

- Move from 1 memory to another one
    - We don't have this info
        - Will be exception, the OS (program) realized that he doesn't have this info
            - So defined trap
            - Segmentation fault / page fault depending on how memory is managed

Computer System Organization



Von Neumann architecture

new ones are not von neumann

CPU and memory = main element for any computers – connected via bus

- Bring information
- Cpu with 32 bit
    - 32 lines to move
    - Define the addresses
        - Each lime has electricity
        - 2 states – 0 or 1 from each line
    - 4 billion addresses or 2^32 possible addresses (all powers of 2)

1 kilbyte = 2^10

1 mb = 2^20 (2^10 * 2^10)

1 gb = 2^30

Recall these for the exam

1 kb of memory = 2^10

Can put different devices on the bus

Each device has a controller

- Specific roles
- Control the devices
- May be replaced by a small computer with os to control device
    - Helps OS to manage computer

Cpu acts on memory, bring mem info through bus to cpu to perform math op or manipulation of memory and then finish to bring back to memory

Bus has lots of traffic

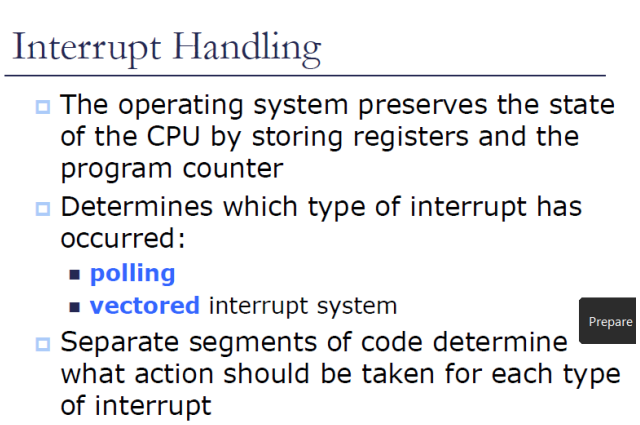
CPU/OS to manage the traffic

Devices to use and connect

- Bring info from disk to memory then form memory to cpu -> memory -> back to disk

Apply device = applies interrupt

Guides OS to perform OP and move info from one place to another



To help OS, we need ot know how to catch this info

What interept happens

Polling – continuisly samples vector interrupt

IDT table – 128 vector interup and continuously sampling 128 quite fast

Vectored interrupt system

- Different , if we have interrupt .. flag turns up and this info goes to vector directly and deal with interrupt connects with flag
- Direct access to interrupt on vector
- Hash table where interrupt is hashed and then acted on by vector
- Vector O(1) to interact with
- Polling
    - Passes over all interrupts
        - O(n) much slower

Different parts/segments in code with different action defined by interrupts

Seg to load info and seg to manipulate info

Different parts of code

OS is management layer

What we need to think of

Task management

- Cpu or specific device
- Memory manahgement
- Second memory = usb drive or hard drive or cloud or external drive
- Main mem = ram
- File system = give names of files (illusion)
- Security
    - Protections for system, user ,program, tasks
- Multiple users – need to manage
- Way to access commands of OS from user
    - For unprotected commands
        - **We call this the shell (UI)**
            - Terminal – CMD
                - Call different OS commands
                - Windows dir, cd, ping to ip
                    - Protected commands but access via shell
                - And unix course

Operating System Components

In addition…
- boot
- backup
- …
- browser

- processes
- memory
- I / O
- secondary memory
- File system
- Security
- Manage user accounts
- User interface (shell)

Additional stuff

Boot – BIOS

- Something to call the OS

Backup system

- SSD is problematic
    - No flag raised if something is wrong

Browser

- For network
- Chrome etc

old OS

1 layer for kernel / OS

- Advantages
    - More in sync
    -
- Cons

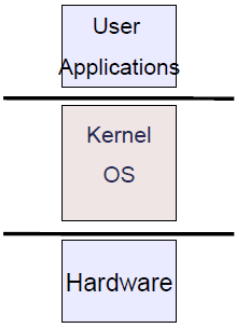
Operating System Organization

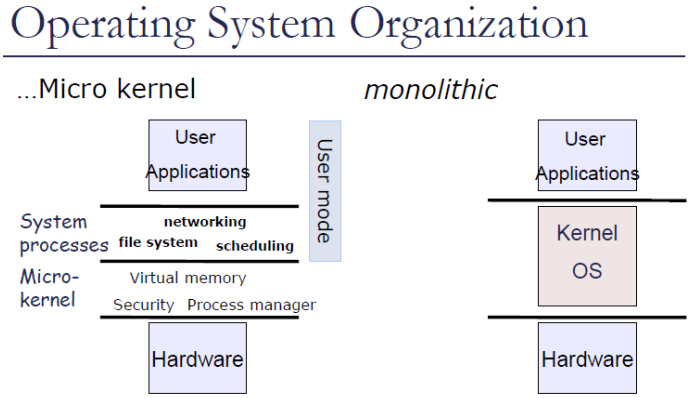
✓ Cheap communication inside the Kernel *monolithic*

× It is hard to understand

× It is difficult to change or add components

What is the alternative?





- Hard to change 1 part of it

DOS was monolithic

Comms – kernel has process, mem, network, etc

No need for comms because all in 1 place

Messy program , hard to maintain

Micro kernel OS

- Opposite
- Smaller kernel
- Main component

Van neumman

- With micr kernel

But on top of it is system process

- Schedule tasks on system

System process connect, cpu, virtual me, etc…

2 places in kernel mode

Moved network, file system and scheduling to user mode

- Pros
  -
- Cons
  - Using system processes get more flavors/components
  - Slowly increase the size by system process on user mode
  - Run system process and additional components
  - Increase OS size
  - Reduce size of user application

Buy for running applications not for the OS

Computers with micro kernels

- Basic
- But we have a new printer, where to bring driver
  - Call networking in system process
    - Bring driver but install in user space
    - Less kernel panics / bsods
    - But less space for the user

OS after years slow down… then erase and reinstall

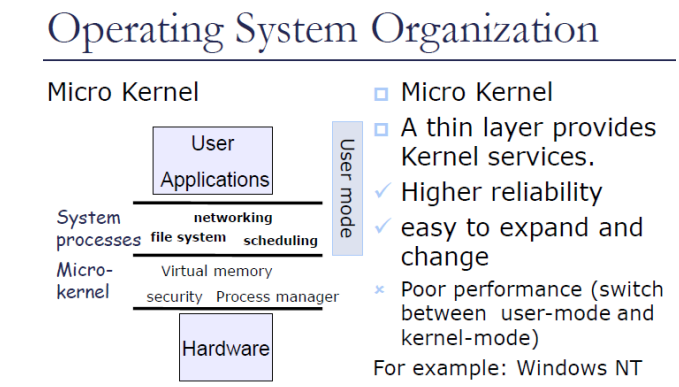
every time you look at something, look at good and bad…nothing is perfect

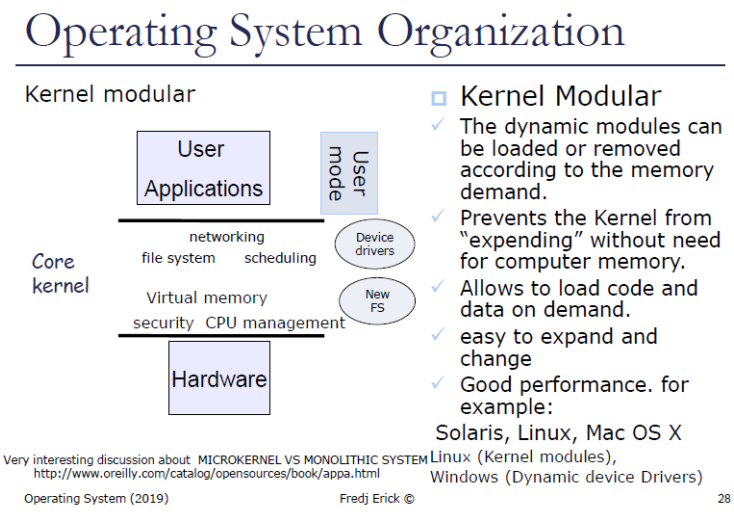
windows NT is an example

micro kernel is more flexible than monoloithic

easier to modify

not always more reliable





Linux approach

Kernel is thin layer between user and hardware

Each component is a module

Like objects in c++ , easily expand and manipulate

Its in 1 core kernel

Also dos advantages – one big program

- Main function of program, and write everything in main
- Huge program with everything

Krnel packages are abstract and dynamic

Communication isn't so expensive

Easily load and run modules

Good performance – same as 1 big layer

New windows uses approach too

Solaris, linux, os x

http://www.oreilly.com/catalog/opensources/book/appa.html

new File Systems

- Ntfs
- Fat
- Exfat etc
- Refs for windows
    - Different file systems defines the file size like block size