EECS 3221 Operating System Fundamentals

Instructor: Prof. Hui Jiang
Email: hj@cse.yorku.ca
Web: http://www.eecs.yorku.ca/course/3221

General Info

- 3 lecture hours each week
- 2 assignments (2*5%=10%)
- 1 project (10%)
- 5 in-class short quizzes (10%)
- In-class mid-term (30%)
- Final Exam (40%) (final exam period)
- In-class
  - Focus on basic concepts, principles and algorithms
  - Examples given in C
  - Brief case study on Unix series (mainly Linux)
- Assignments and tests
  - Use C language

Bibliography

- Required textbook
  - *"Operating System Concepts: 9th edition"*
- Other reference books (optional):
  - *"Advanced Programming in the Unix Environment"* (for Unix programming, Unix API)
  - "*Programming with POSIX threads*" (Multithread programming in Unix, Pthread)
  - "Linux Kernel Development (2nd edition)" (understanding Linux kernel in details)

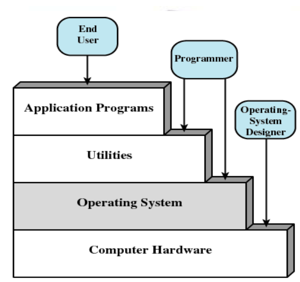
Why this course?

- OS is an essential part of any computer system
- To know
  - what's going on behind computer screens
  - how to design a complex software system
- Commercial OS:
  - Unix, BSD, Solaris, Linux, Mac OS, Android, Chrome OS
  - Microsoft DOS, Windows 95/98,NT,2000,XP,Vista, Win7, Win8

What is Operating System?

- A program that acts as an intermediary between computer hardware and computer users (or user applications).
- OS manages computer hardware:
  - Use the computer hardware efficiently.
  - Make the computer hardware convenient to use.
  - Control resource allocation.
  - Protect resource from unauthorized access.

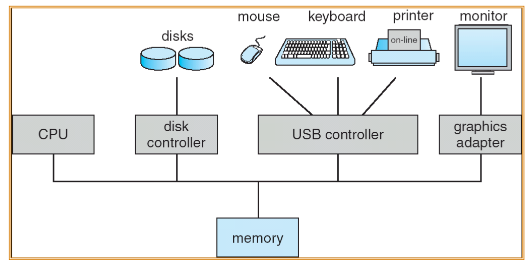
Computer Structure



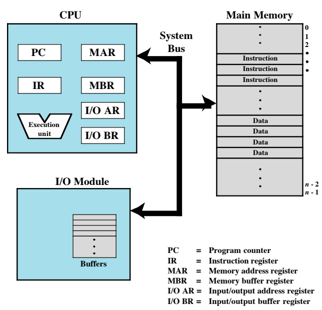
Hardware Review

- Instruction execution
- Interrupt
- Three basic I/O methods
- Storage hierarchy and caching

Computer Hardware

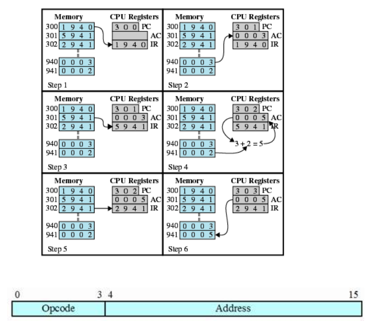


Computer Hardware



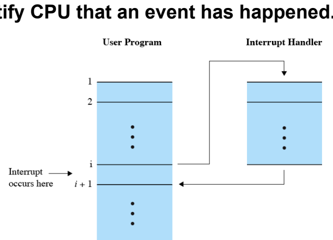
Computer Components: Top-Level View

Instruction Execution

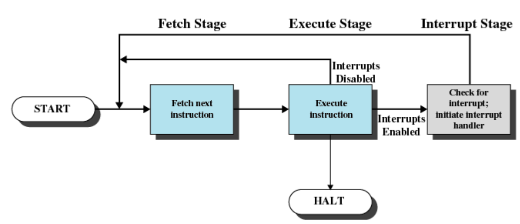


Interrupts

- A hardware signal to interrupt the normal execution sequence of CPU.
- To notify CPU that an event has happened.

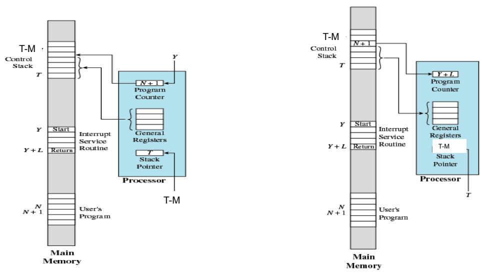


Instruction Cycle with interrupts



Instruction Cycle with Interrupts

Interrupts



(a) Interrupt occurs after instruction at location N

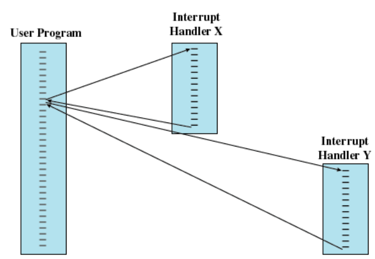
(b) Return from interrupt

Interrupt Handler

- Program or subroutine to service a particular interrupt.
- A major part of the operating system is implemented as Interrupt handlers since modern OS design is always *interrupt-driven.*
- Determines which type of interrupt has occurred:
  - *Polling*
  - Vectored interrupt system
- Interrupt Vectors: saved in low-end memory space

Multiple Interrupts

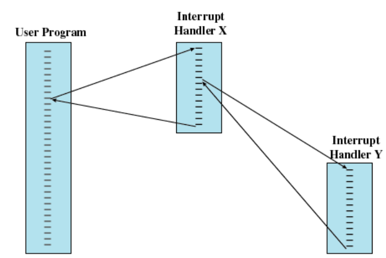
- Sequential interrupt processing: disable interrupts while an interrupt is being processed



(a) Sequential interrupt processing

Multiple Interrupts

- Nested interrupt processing: define priority for interrupts.
- A high-priority interrupt preempts a low-priority one.

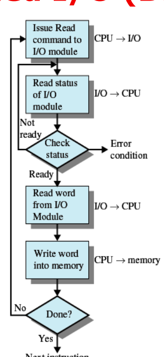


(b) Nested interrupt processing

I/O Communication Techniques

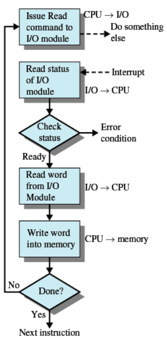
- Programmed I/O (busy-waiting)
- Interrupt-driven I/O
- Direct memory access (DMA)

Programmed I/O (Busy-waiting)



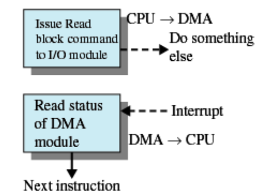
(a) Programmed I/O

Interrupt-driven I/O



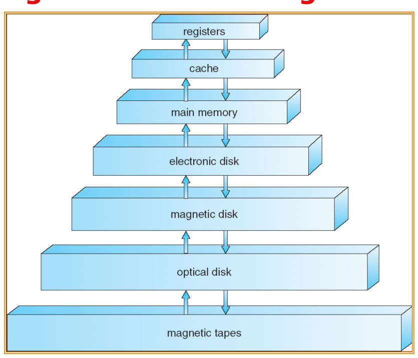
(b) Interrupt-driven I/O

DMA



(c) Direct memory access

Storage Structure: storage hierarchy



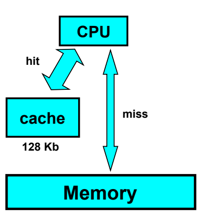
Storage Hierarchy

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Volatile vs. Persistent

Caching

- **Caching is an important principle in computer systems.**
- **Improve access speed with minimum cost.**
- **Caching: copy information to a faster storage system on a temporary basis.**

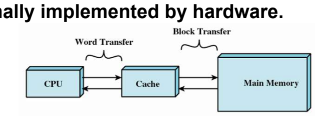


Example:

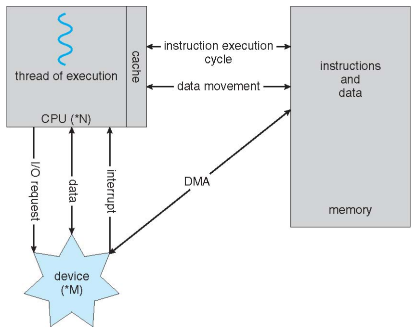
One memory access 100 nanoseconds
One cache access 20 nanoseconds
If hit rate is 99%, then
(1) 128M memory without cache: 100 nano
(2) 128M cache: 20 nano (too expensive)
(3) 128M memory + 128K cache:
0.99*20+0.01*120 = 21 nano

Caching

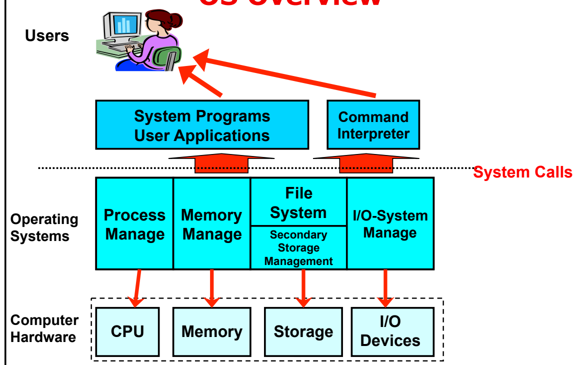
- **Why high hit rate?**
  - **Memory access is highly correlated**
  - **Locality of reference**
- **Cache Design:**
  - **Cache size**
  - **Replacement algorithm: Least-Recently-Used (LRU) algorithm**
  - **Write policy: write memory when updated or replaced.**
  - **Normally implemented by hardware.**



Computer System (overview)



OS Overview



Process Management

- A *process* is a program in execution.
- A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - Process suspension and resumption.
  - Provision of mechanisms for:
    - Process synchronization
    - Inter-process communication
    - Handling dead-lock among processes

Main-Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.
- For a program to be executed, it must be mapped to absolute addresses and loaded into memory.
- We keep several programs in memory to improve CPU utilization
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of memory usage.
  - Manage memory space of all processes.
  - Allocate and de-allocate memory space as needed.

Secondary-Storage Management

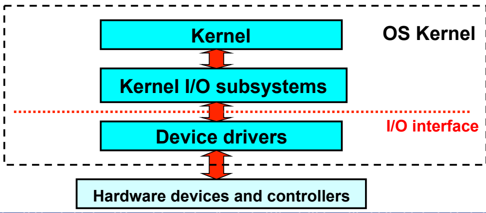
- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use hard disks as the principal on-line storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling

File Management

- File system: a uniform logical view of information storage
- A File:
  - logical storage unit
  - a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- Files are organized into directories to ease their use.
- The operating system is responsible for the following activities in connections with file management:
  - File Name-space management
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.

I/O System Management

- The I/O system consists of:
  - A memory-management component that includes *buffering*, *caching*, and *spooling*.
  - A general device-driver interface.
  - Drivers for specific hardware devices.



Protection System

- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.

Content in OS Course

- Managing CPU usage
  - Process and thread concepts
  - Multi-process programming and multithread programming
  - CPU scheduling
  - Process Synchronization
  - Deadlock
- Managing memory usage
  - Memory management and virtual memory
- Managing secondary storage
  - File system and its implementation
  - Mass-storage structure
- Managing I/O devices:
  - I/O systems
- Protection and Security
- Case study on Unix series (scattered in all individual topics)

Tentative schedule (subject to change)

Totally 12 weeks:

- Background (2.5 week)
- Process and Thread (2 weeks)
- CPU scheduling (1 week)
- Process Synchronization (2.5 weeks)
- Memory Management (2 weeks)
- Virtual Memory (1 week)
- Protection and Security (1 week)

Several must-know OS concepts

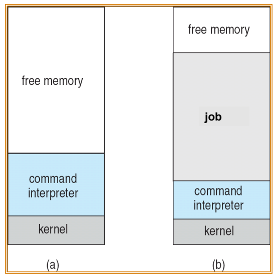
- System Booting
- Multiprogramming
- Hardware Protection
  - OS Kernel
- System Calls

OS Booting

- Firmware: bootstrap program in ROM
  - Diagnose, test, initialize system
- Boot block in disc
- Entire OS loading

Simple Batch Systems (Uniprograming)

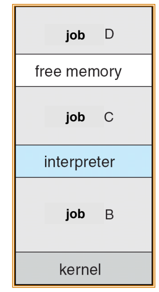
- OS Kernel:
  - initial control in OS
  - OS loads a job to memory
  - control transfers to job
  - when job completes control transfers back to monitor
- Automatic job sequencing – automatically transfers control to another job after the first is done.
- Batch system is simple to design, but CPU is often idle.



Memory Layout for a Simple Batch System

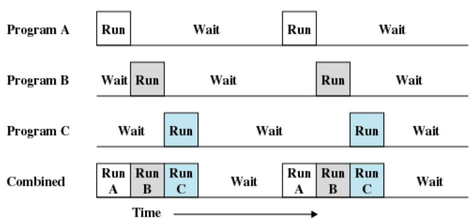
Multiprogramming System

- Several jobs are kept in main memory at the same time, and CPU is multiplexed among them.
- How to implement multiprogramming is the center of modern OS.
- OS Features Needed for multiprogramming:
  - Memory management – the system must allocate the memory to several jobs
  - Some scheduling mechanism – OS must choose among several jobs ready to run
  - Protection between jobs.
  - Allocation of devices to solve conflicts
  - I/O routine supplied by the OS



Memory Layout for Multiprogramming System

Multiprogramming



(c) Multiprogramming with three programs

Multiprogramming: example

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Time-Sharing Systems (Multitasking) –Interactive Computing

- Multitasking also allows time sharing among jobs: Job switch is so frequent that the user can interact with each program while it is running.
- Maintain a time slice.
- Allow many users share a single computer.
- Used to implement a popular scheduling algorithm towards fairness.

Hardware Protection

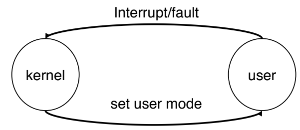
- Dual-mode Protection Strategy
  - OS Kernel
- Memory protection
- CPU protection
- I/O protection

Dual-Mode CPU

- Provide hardware support to differentiate between at least two modes of CPU execution.
  1. *Kernel mode* (also monitor *mode* or *system mode*) – execution on behalf of operating system.
  2. *User mode* – execution on behalf of user programs.
- A mode bit in CPU to indicate current mode.
- Machine instructions:
  - Normal instructions: can be run in either mode
  - Privileged instructions: can be run only in kernel modes
- Dual-model CPU for OS protection:
  - OS always in kernel mode; user program in user mode.

Dual-Mode CPU Operation (Cont.)

- When booted, CPU starts from kernel mode.
- When an interrupt occurs, hardware switches to kernel mode.

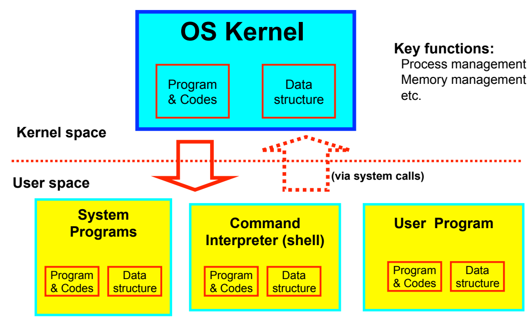


- OS always in kernel mode; user program in user mode. (Guaranteed? and how?)
  - OS always switches CPU to user mode before passing control of CPU to any user program.

Dual-Mode CPU Operation

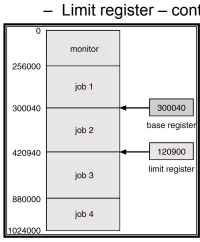
- Carefully define which instruction should be privileged:
  - Common arithmetic operations: ADD, SHF, MUL, …
  - Change from kernel to user mode
  - Change from user to kernel mode **(not allowed)**
  - Turn off interrupts
  - TRAP
  - Set value of timer
  - Set CPU special-purpose registers
  - I/O related instructions

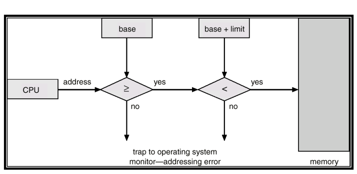
OS Kernel



Memory Protection

- Each running program has its own memory space
- Add two registers that determine the range of legal addresses:
  - base register – holds the smallest legal physical memory address.
  - Limit register – contains the size of the range





- Loading these registers are privileged instructions
- OS, running in kernel mode, can access all memory unrestrictedly

CPU Protection

- *Timer* – interrupts CPU after specified period to ensure operating system maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches the value 0, an interrupt occurs.
- OS must set timer before turning over control to the user.
- Load-timer is a privileged instruction.
- Timer commonly used to implement time sharing.
- Timer is also used to compute the current time.

I/O Protection

- To prevent users from performing illegal I/O, define all I/O instructions to be privileged instructions.
- User programs can not do any I/O operations directly.
- User program must require OS to do I/O on its behalf:
  - OS runs in kernel mode
  - OS first checks if the I/O is valid
  - If valid, OS does the requested operation; Otherwise, do nothing.
  - Then OS return to user program with status info.
- How a user program asks OS to do I/O
  - Through *SYSTEM CALL* (software interrupt)

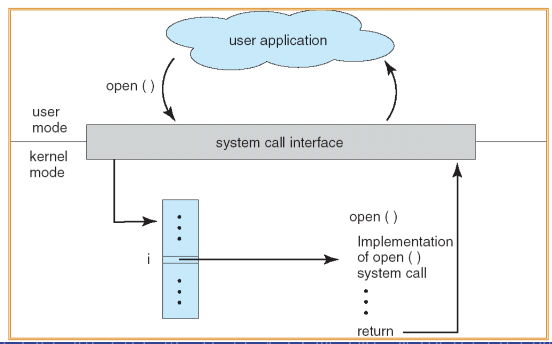
System Calls

- System calls provide the interface between a running user program and the operating system.
- Process and memory control:
  - Create, terminate, abort a process.
  - Load, execute a program.
  - Get/Set process attribute.
  - Wait for time (sleep), wait event, signal event.
  - Allocate and free memory.
  - Debugging facilities: trace, dump, time profiling.
- File management:
  - create, delete, read, write, reposition, open, close, etc.
- I/O device management: request, release, open, close, etc.
- Information maintain: time, date, etc.
- Communication and all other I/O services.

System Call Implementation (I)

- Typically, a unique number is associated with each system call:
  - System-call interface maintains a table indexed according to these numbers.
- Basically, every system call makes a software interrupt (TRAP).
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values.

System Call – OS Relationship

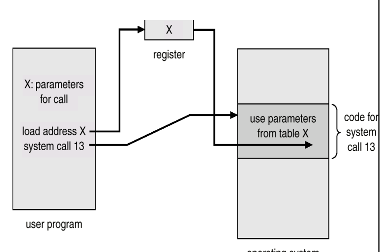


System Call Implementation (II)

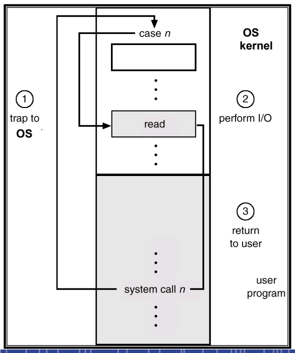
- Three general methods are used to pass parameters between a running program and the operating system.
  - Pass parameters in *registers*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
    (This approach is taken by Linux and Solaris.)
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

Parameters Passing Via Table

```
main()
{
   _strut_PARA  sp;

…

_set_para_(&sp) ;

_system_call_(13,&sp);

…

}
```



Use of A System Call to Perform I/O



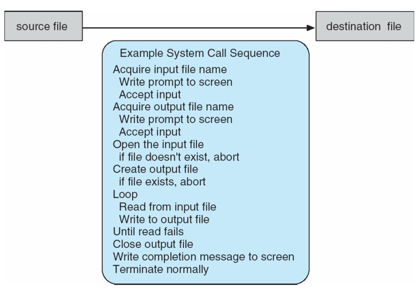
Some UNIX I/O system calls

- open(), read(), write(), close(), lseek():

```
#include <sys/stat.h>
#include <fcntl.h>
 int open(const char *path, int oflag)  ;

#include <unistd.h>
  ssize_t read(int fd, void *buf, size_t count);

#include <unistd.h>
 ssize_t write(int fd, const void *buf, size_t count);

#include <unistd.h>
 int close(int fd);

#include <unistd.h>
  off_t lseek(int fildes, off_t offset, int whence);
```

Example of System Calls

- System call sequence to copy the content of one file to another file

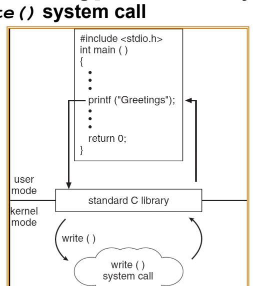


System Call vs. API

- System calls are generally available as assembly-language instructions:
  - Some languages support direct system calls, C/C++/Perl.
- Mostly accessed by programs via a higher-level Application Program Interface (API) rather than direct system call use.
- Why use APIs rather than system calls?
  - API's are easier to use than actual system calls since they hide lots of details
  - Improve portability

Standard C Library Example

- C program invoking ***printf()*** library call, which calls ***write()*** system call



System Calls: Unix vs. Windows

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()