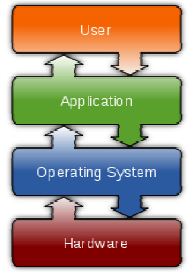
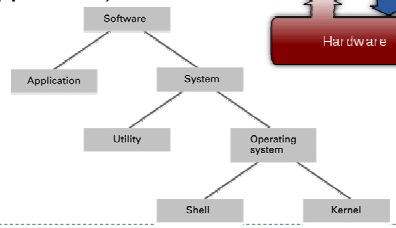
Operating System

Virtualization

The role of OS

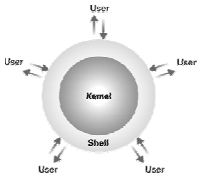
- Manage computer resources, such as disk, memory, CPU, …, for programs (application) to be executed on computers
- Provide an interface for users to use computers (and application).





Components of an operating system

- For **user**: shell, privilege control (security)
  - GUI (graphical user interface): **W**indows, **I**cons, **M**enus, **P**ointer
- For **data**: file manager
- For **hardware**:
  - Device manager,
  - Memory manager,
  - Boot loader
- For **software**:
  - Where to store: File manger, Registry
  - How to execute: scheduler, process manager



Outline

- History
- Virtualization
  - Multitasking
    - Resource sharing problems
      - Resource protection
      - Deadlock
  - Virtual memory

History of Operating System

- 1940s~1950s: programs are 'stored' on tapes and punched cards. ➔ Need operators to start programs
- The first operating system: use a program to setup and to stream programs ➔batch processing
- Two new needs
  - The need for users' interaction ➔ iterative processing
  - The deadline requirement ➔ real-time processing
- 1960s~1970s: multiuser environment
  - More than one iterative tasks and real-time tasks need be executed on a computer
  - Time sharing (multiprogramming), multitasking system

Virtualization

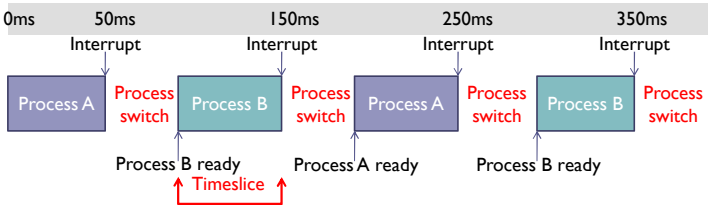
- Virtualization is a technique for hiding the physical characteristics of computing resources to simplify the way in which other systems, applications, or end users interact with those resources. *
- Two examples in OS
  - Multitasking
  - Virtual memory

* Cited from http://cplus.about.com/od/glossar1/g/virtualization.htm

Program vs. process vs. context

- **Program**: a set of instructions
- **Process**: the activity of executing a program
  - A program can be run multiple times; each instance/activity is called a process
- **Context**: a snapshot of the current status of a process
  - A process identifier, or PID
  - Register values, Program Counter value
  - The memory space, I/O, files for the process
  - **State** of the process.
    - Ready: ready for execution.
    - Waiting: waiting for some I/O.
    - Complete: finished process.

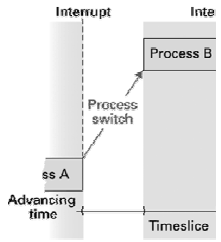
Multiprogramming



- Each process is executed a fixed period of time (timeslice)
- After a timeslice, the computer (CPU, memory…) is switched to execute another process
  - Called process switch or context switch
- The process switch is triggered by an interrupt, which is generated by a timer.

Context switch (process switch)

1. Receive an interrupt from timer
2. Go to the interrupt handler
   a. Save the context of process A
   b. Find a process ready to run (Assume that is process B)
   c. Load the context of process B
3. Start (continue) process B



Resources sharing problems

- What are resources?
  - CPU, memory, files, peripheral devices, …
- In a multitasking system, resources are shared by processes
- Some resources should not be occupied by more than one process at a time
  - E.g., Printer
- Protect non-sharable resources
  - **Critical Region:** A group of instructions that should be executed by only one process at a time
  - **Mutual exclusion:** Requirement for proper implementation of a critical region

First algorithm

- Use a flag (a global memory address)
  - flag=1: the critical region is occupied
  - flag=0: no process is in the critical region
- Problem:

Process A

```
if (flag == 0) {
   flag = 1;
   /*critical region*/
}
```

Context switch to A

Process B

```
if (flag == 0) {
   flag = 1;
   /*critical region*/
}
```

Context switch to B

  - Both processes get into the critical region

Solutions

Testing&setting the flag must be completed w/o interruption (atomic)

1. Use disable_Interrupt() to prevent context switch during the flag test and set process.
2. A machine instruction called "test-and-set" which cannot be interrupted
3. Semaphore: a properly implemented flag

```
Disable_Interrupt();
if (flag == 0) {
   flag = 1;
   Enable_Interrupt();
   / *critical region*/
}
Enable_Interrupt();
```

Another problem: deadlock

- Example:
  - A is in critical region 1, and waits to enter critical region 2
  - B is in critical region 2, and waits to enter critical region 1

Process A — Context switch to A

```
while (test_set(flag1)) {
   /*critical region 1*/
   while(!test_set(flag2));
   /*critical region 2*/
}
```

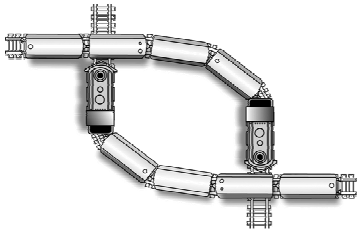
Process B — Context switch to B

```
while (test_set(flag2)) {
   /*critical region 2*/
   while (!test_set(flag1));
   /*critical region 1*/
}
```

Conditions for deadlock

1. Competition for non-sharable resources
2. Resources requested on a partial basis
3. Allocated resources cannot be forcibly retrieved
4. Circular wait

Remove any one of the conditions can resolve the deadlock.



Solutions

Which condition is removed?

1. Kill one of the process
2. Process need to request all the required resources at one time
3. Spooling
   - For example, stores the data to be printed and waits the printer available
4. Divide a file into pieces so that it can be altered by different processes

Exercises

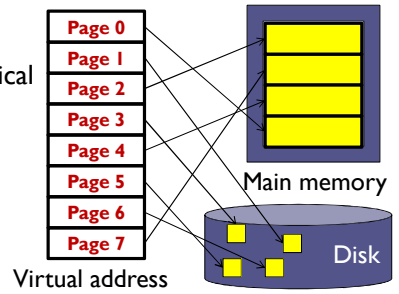
- There is a bridge that only allows one car to pass. When two cars meet in middle, it causes "deadlock". The following solutions remove which conditions
  1. Do not let a car onto the bridge until the bridge is empty.
  2. If cars meet, make one of them back up.
  3. Add a second lane to the bridge.
- What's the drawback of solution 1?

Virtual memory

- Virtual memory: employs the physical memory and disk space to create the *illusion* of a larger memory space
- Scenario 1: Suppose there is program A
  - Program A need memory space 1G
  - RAM is only of 512M
- Scenario 2: Suppose there are two programs: A and B
  - Program A need be placed in memory 0x0000-0x08000
  - Program B need be placed in memory 0x0000-0x0A000
  - Program A and B are executed concurrently (in the multiprogramming sense)

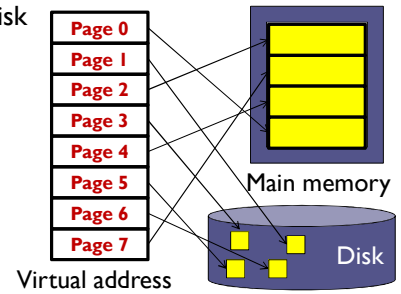
Paging system

- Memory space is divided into a set of equal-sized pieces; each piece is called a page.
- Programs use virtual address to access data and code
- There is a table (page table) mapping virtual address to physical memory address
- OS maintains the page table



Scenario 1

- Program A need memory space 1G; RAM is only 512M.
- There are 8 pages; each is of128M
- Program A asks OS for data in page 3
- OS finds page 3 is in disk
- OS does paging
  - Find a page in RAM (say page 0)
  - Save page 4 to disk
  - Load page 3 to RAM
- Program A gets data from page 3



Scenario 2

- Program A need be placed in memory 0x0000-0x08000
- Program B need be placed in memory 0x0000-0x0A000
- RAM is of size 64K; each page is 16K
- Program A and program B have different page tables
- When context switching, OS needs to swap page tables.