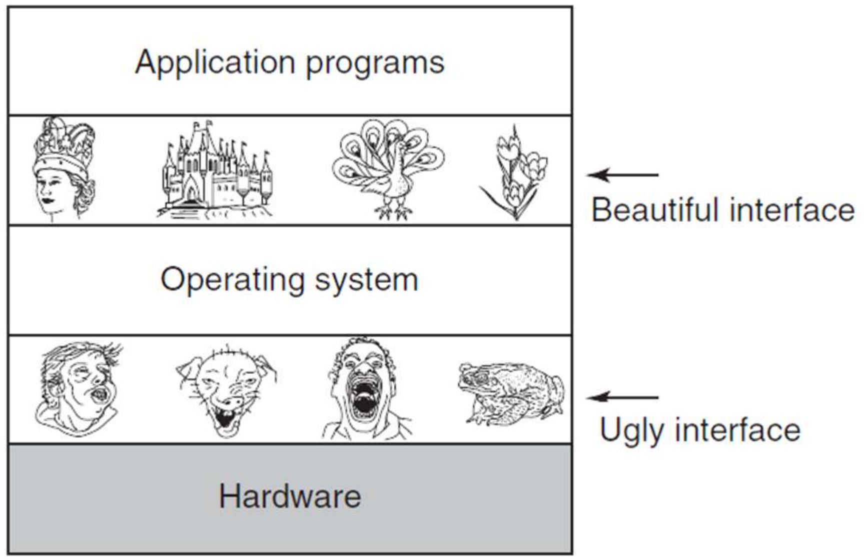
CSE 306 Operating Systems
Operating System Overview

YoungMin Kwon

OS Objectives and Functions

- Operating System
  - Controls the execution of application programs
  - Acts as an interface between application programs and computer hardware

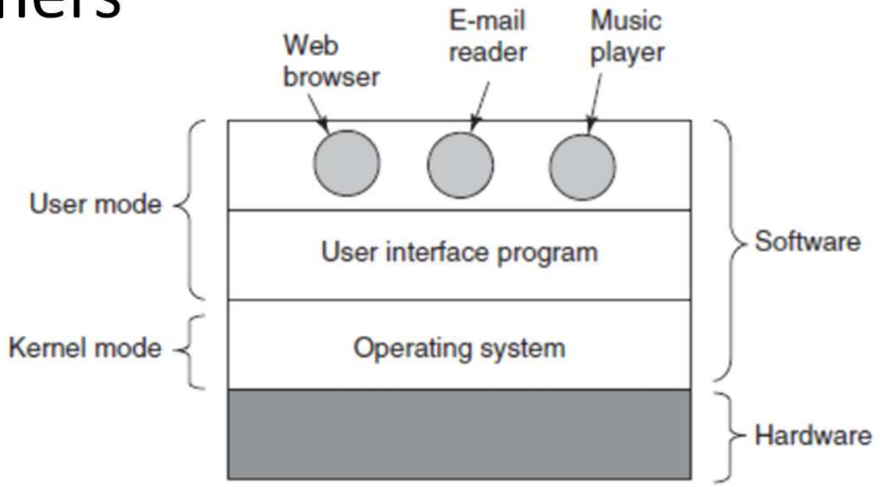


OS Objectives and Functions

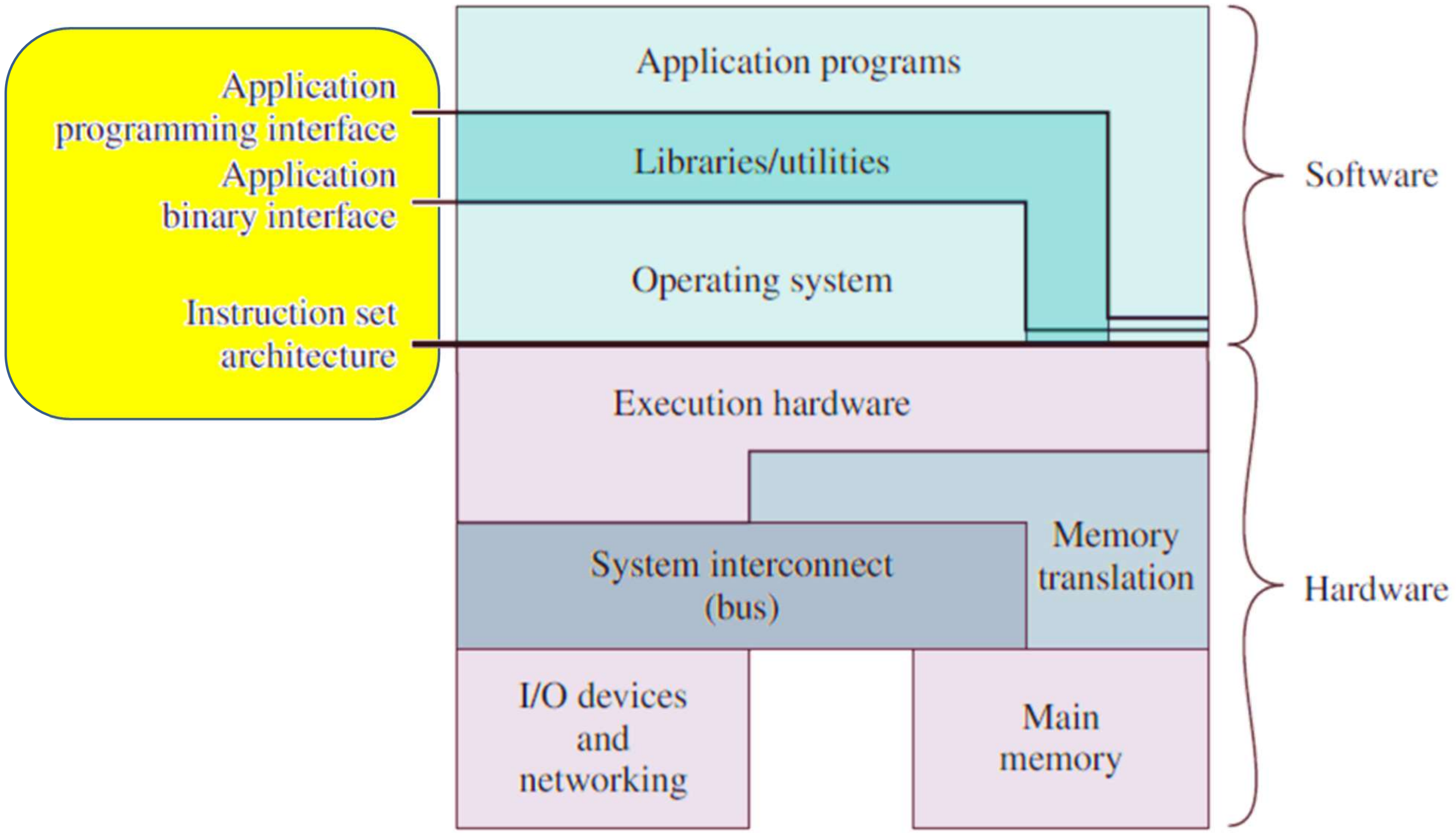
- Objectives
  - Convenience: makes computer easier to use
  - Efficiency: makes the use of computer resources more efficient
  - Ability to evolve: allows effective development, testing, and introduction of new system functions without interfering with service.

As a User/Computer Interface

- Operating Systems
  - The most important collection of system programs comprises the OS
    - Provided as utility or library programs
  - Hide the details of hardware from the programmers
  - Provide convenient interfaces to application programmers



A User/Computer Interface


Application programming interface
Application binary interface
Instruction set architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
System interconnect (bus)
Memory translation
I/O devices and networking
Main memory
Software
Hardware

- Computer Hardware and Software Structure

As a User/Computer Interface

Operating System Services

- Program development: provides a variety of facilities and services to programmers
  - E.g. editors, debuggers
- Program execution: steps to execute a program
  - Loading, linking, initializing I/O devices and files
- Access to I/O devices: OS hides device specific details from the programmers
- Controlled access to files: data structures stored in files, protection from other users

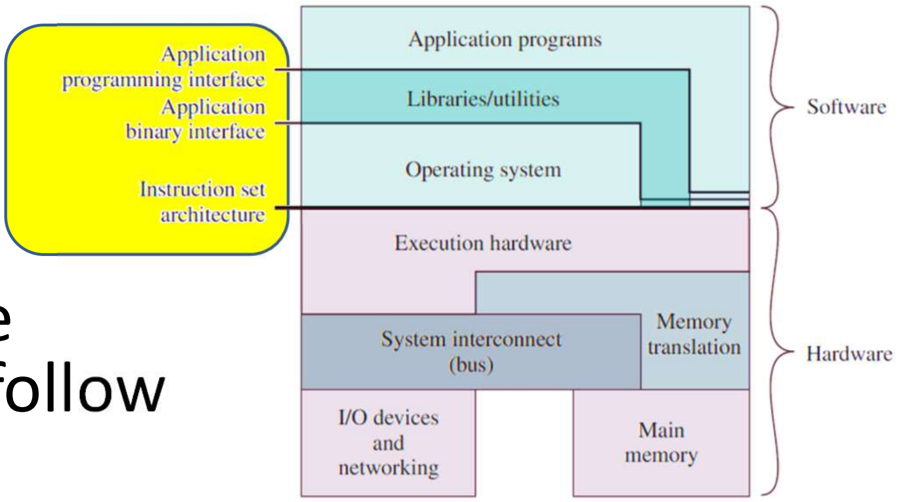
As a User/Computer Interface

Operating System Services

- System access: protection of resources and data from unauthorized users and resolve conflicts for resource contention
- Error detection and response: ability to detect errors and to make response to clear the error with the minimal impact
  - memory error, device failure, division by zero, illegal access to memory
- Accounting: collect usage statistics for various resources and monitor parameters

As a User/Computer Interface

3 Key Interfaces



Application programming interface
Application binary interface
Instruction set architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
System interconnect (bus)
Memory translation
I/O devices and networking
Main memory
Software
Hardware

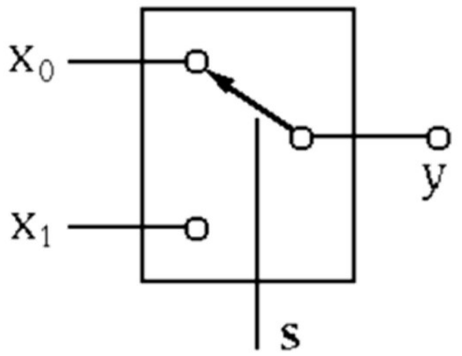
- Instruction Set Architecture (ISA): the set of machine language instructions that a computer can follow
  - User ISA, system ISA
- Application Binary Interface (ABI): standard for binary portability across programs
  - System call interfaces
- Application Programming Interface (API): through recompiling, enables application software to be ported easily to other systems
  - Library functions wrapping system calls

As a Resource Manager

- OS is responsible for controlling the computer's resources
  - I/O, main and secondary memory, CPU time
- Control mechanism: OS is **not** external to what is being controlled
  - OS functions in the same way as ordinary computer software
  - OS frequently relinquishes control and depends on CPU to regain control

As a Resource Manager

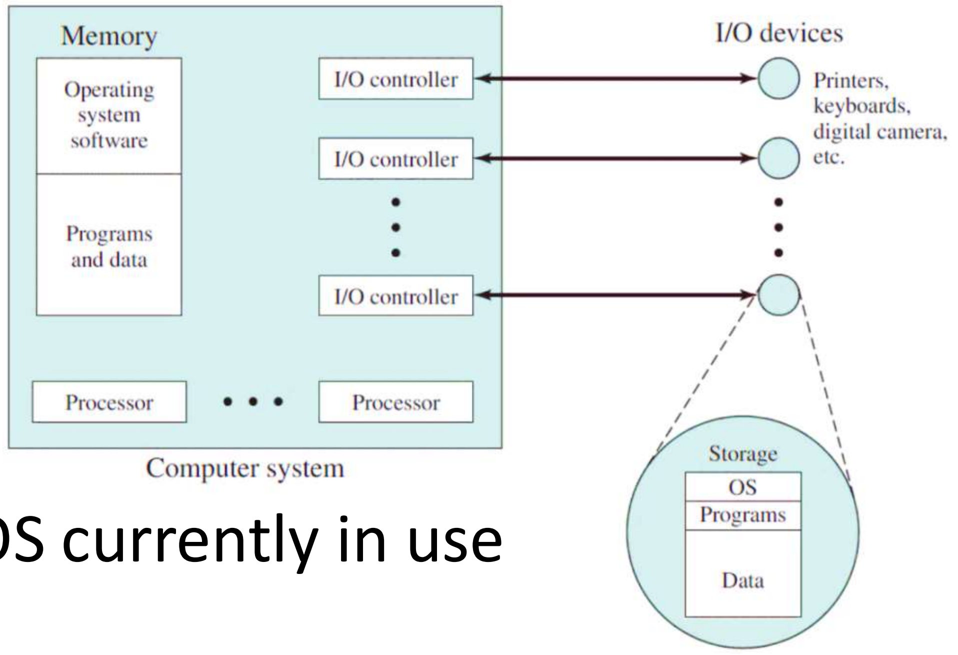
- Multiplexing resources



y: a resource; x_0, x_1: processes

  - By time: e.g. CPU
  - By space: e.g. memory

As a Resource Manager



- Main memory
  - Controlled by OS and MMU
  - User program and data
  - Kernel and other portions of OS currently in use
- I/O devices
  - OS controls when and which program can access a device and file
- Processor
  - OS controls how much processor time is used by a process

Evolution of an OS

- OS will evolve over time
  - Hardware upgrades and new types of hardware
    - Paging mechanism, Graphics terminal
  - New services
    - In response to user demands or system managers' needs
    - E.g. a new set of performance tools
  - Fixes
    - Bug fixes for OS

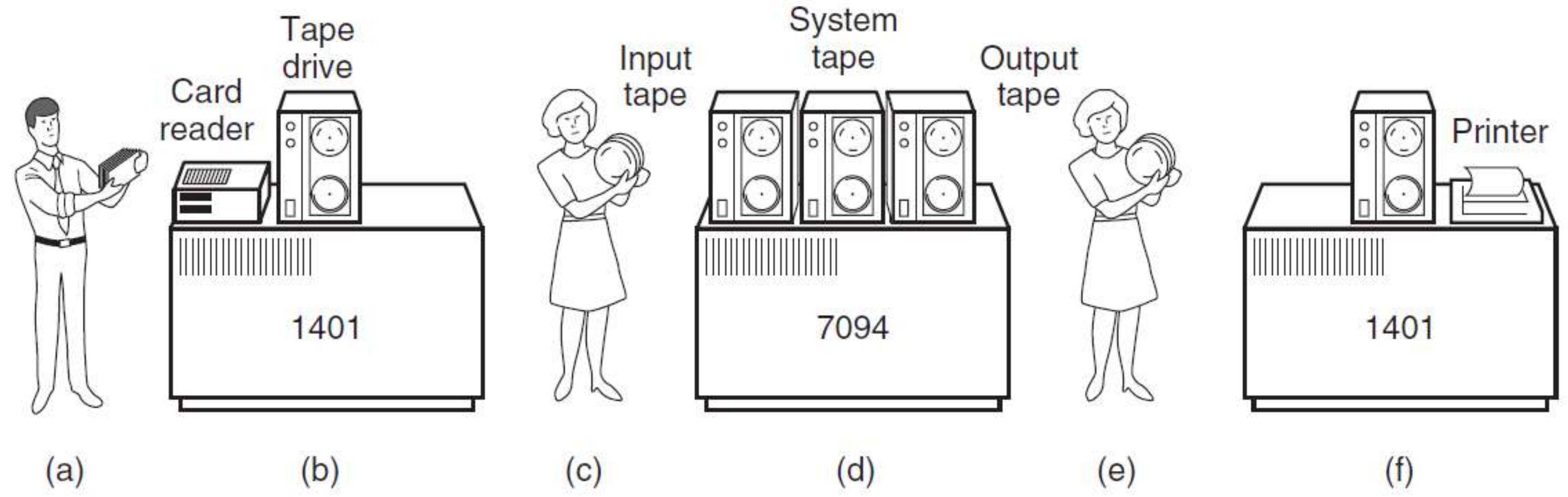
Serial Processing (Evolution of OS)

- Serial Processing
  - No OS support : programmers interact directly with the computer H/W
  - Programs are loaded via input devices (card reader)
  - Output is printed on a printer or a bulb is lit to indicate an error

Serial Processing (Evolution of OS)

- Two main problems
  - Scheduling: users signed up for a block of time (e.g. 45 min)
    - Time can be wasted if the computation is terminated early
    - Users can be expelled if the computation is not finished in time
  - Setup time: a single program (called job) involves
    - Loading the compiler and source program into memory
    - Save the compiled program (object program)
    - Loading and linking the object program and common functions

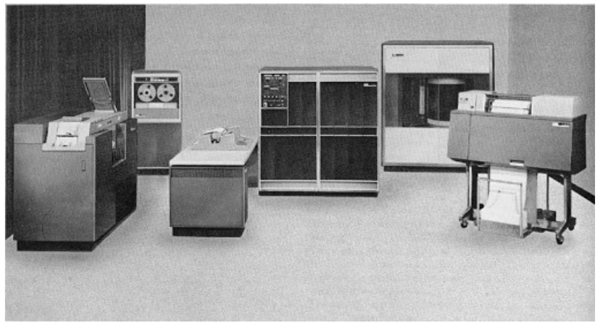
Simple Batch Systems (Evolution of OS)



An early batch system.

(a) Programmers bring cards to 1401.
(b) 1401 reads batch of jobs onto tape.
(c) Operator carries input tape to 7094.
(d) 7094 does computing.
(e) Operator carries output tape to 1401.
(f) 1401 prints output.

IBM 7094



IBM 1401

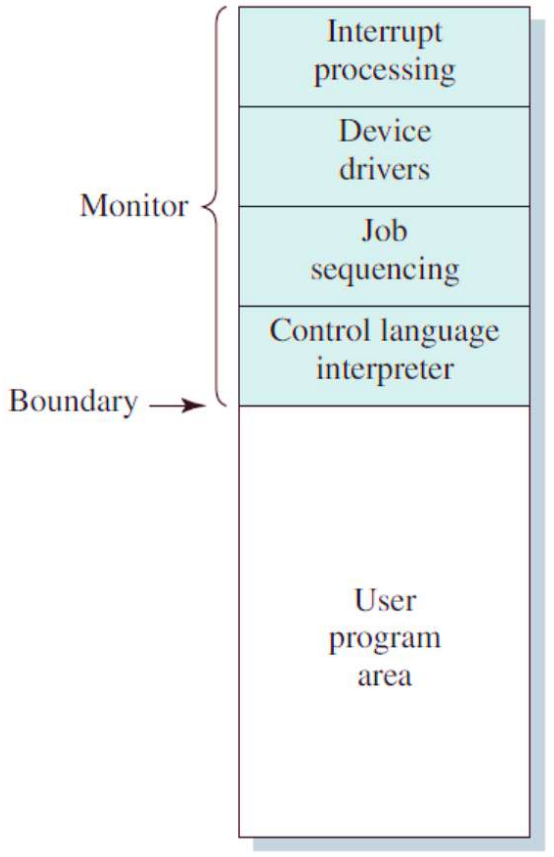
Simple Batch Systems (Evolution of OS)

- Batch OS
  - Improve processor utilization
    - Remove wasted time on scheduling and setup

- Use of software called monitor
  - Users submit jobs (on card, tape) to a computer operator
  - The computer operator batches the jobs together sequentially and places the batch on an input device
  - Each program returns back to the monitor when finished
  - Monitor will load the next program

Simple Batch Systems (Evolution of OS)

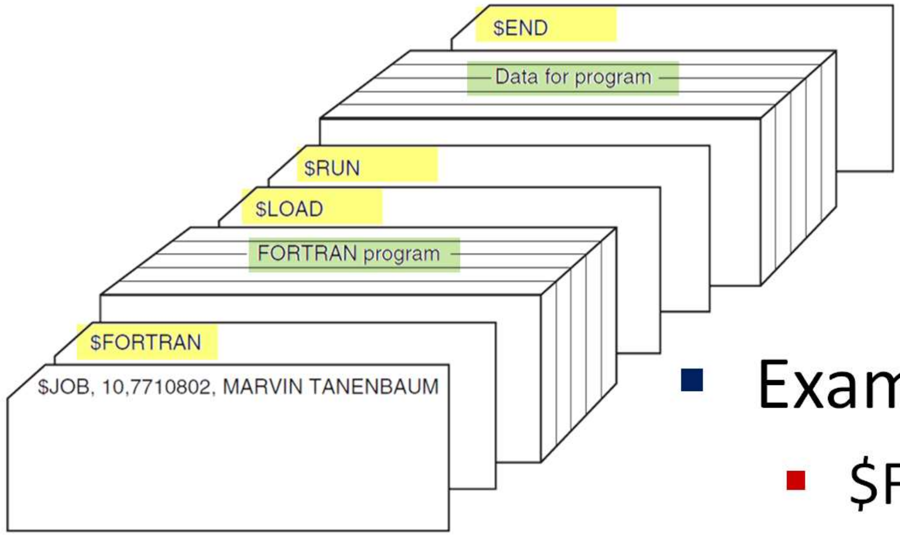
- Monitor
  - Most of the monitor always remain in main memory
  - The rest, utilities and common functions, is loaded at the beginning of the job when necessary
  - Monitor reads in jobs one at a time and replace the current job
  - When a job is completed it returns control to the monitor



Simple Batch Systems (Evolution of OS)

- Processor’s point of view
  - At a certain point of time, the CPU is executing an instruction from the monitor
  - Once the next job is read into memory, a branch instruction makes the CPU start executing the user program
  - The control goes back to the monitor when the user program is terminated either normally or erroneously

Simple Batch Systems (Evolution of OS)



- Example Job Control Language (JCL)
  - $FORTRAN: loads the compiler
  - The compiler translates the user's program into object code
  - $LOAD: loads the object program into memory
  - $RUN: transfers the control to the user program
  - $END: returns the control to the monitor

Simple Batch Systems (Evolution of OS)

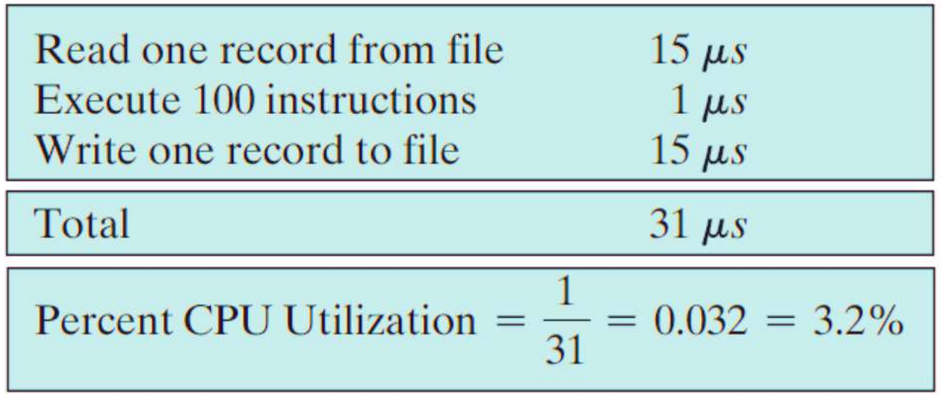
- Desirable features
  - Memory protection: monitor area should not be altered by user programs
  - Privileged instructions: e.g. I/O instructions to prevent reading next instructions from card reader
  - Timer: to prevent a single job from monopolizing the system
  - Interrupts: gives OS more flexibility in control transfer

- Solution: mode of operation
  - User mode: certain restrictions are enforced
  - Kernel mode: privileged instructions can be executed

Multiprogrammed Batch Systems (Evolution of OS)

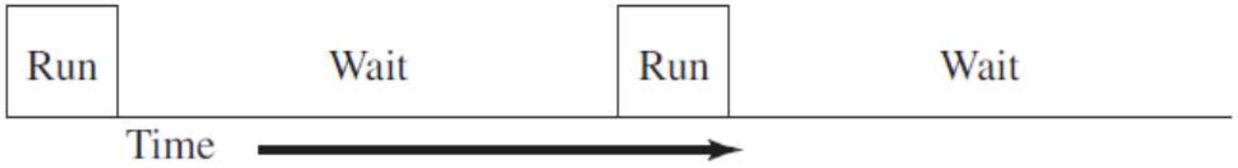
- Processor is still idle in a simple batch system
  - I/O devices are slower than processors

- System utilization example



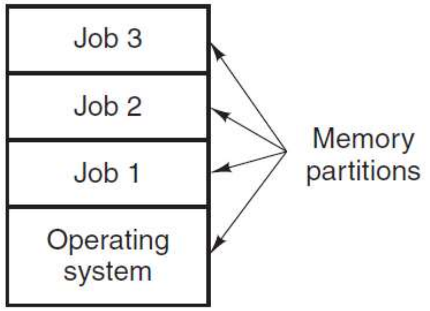
Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
Total	31 μs

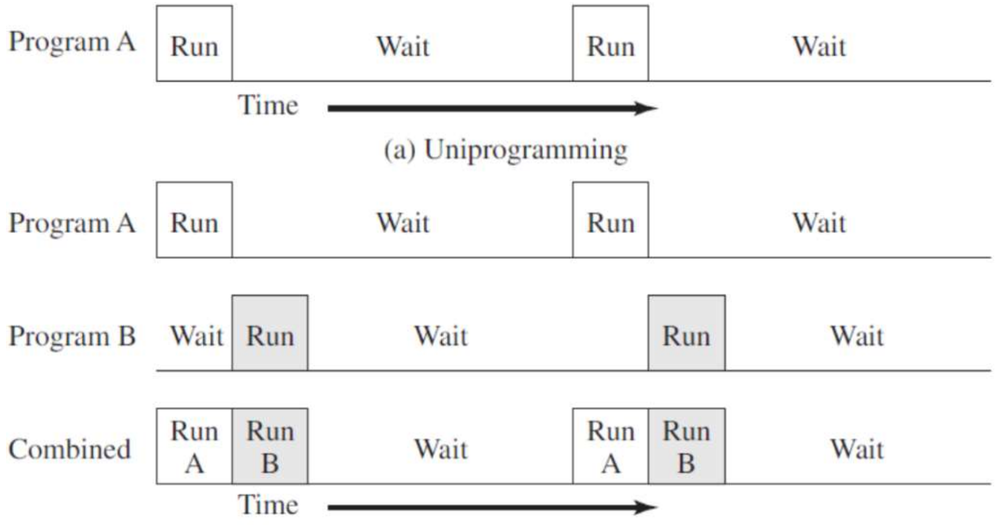
$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$



Multiprogrammed Batch Systems (Evolution of OS)

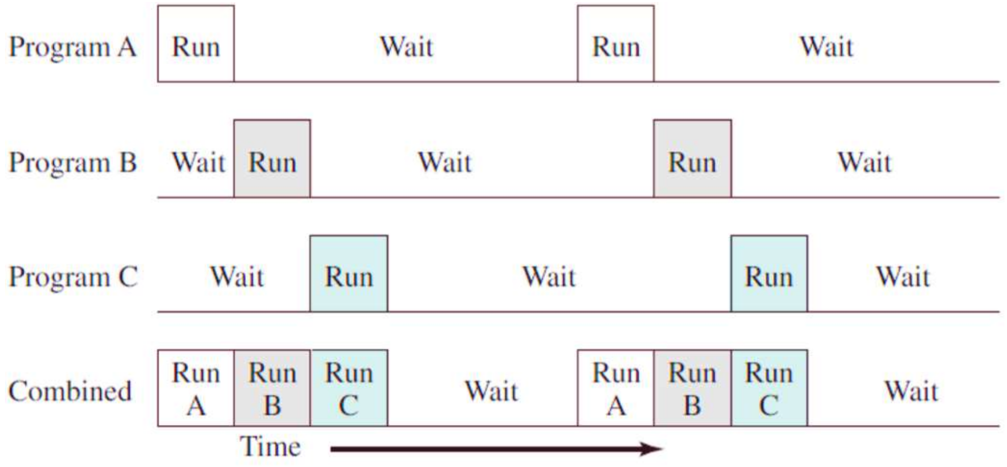
- Multiprogramming (multitasking)
  - When one job needs to wait for I/O, the processor can switch to another job





(a) Uniprogramming

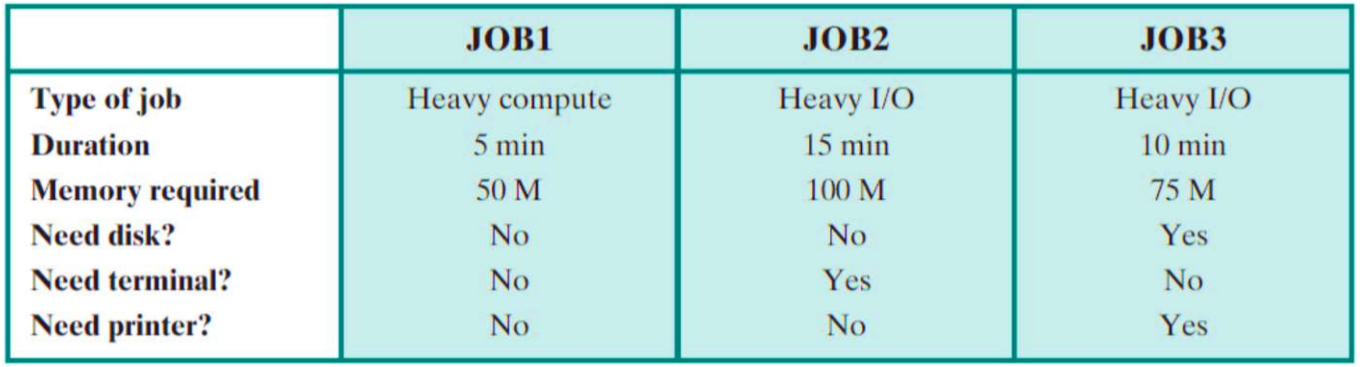
(b) Multiprogramming with two programs



(c) Multiprogramming with three programs

Multiprogrammed Batch Systems (Evolution of OS)

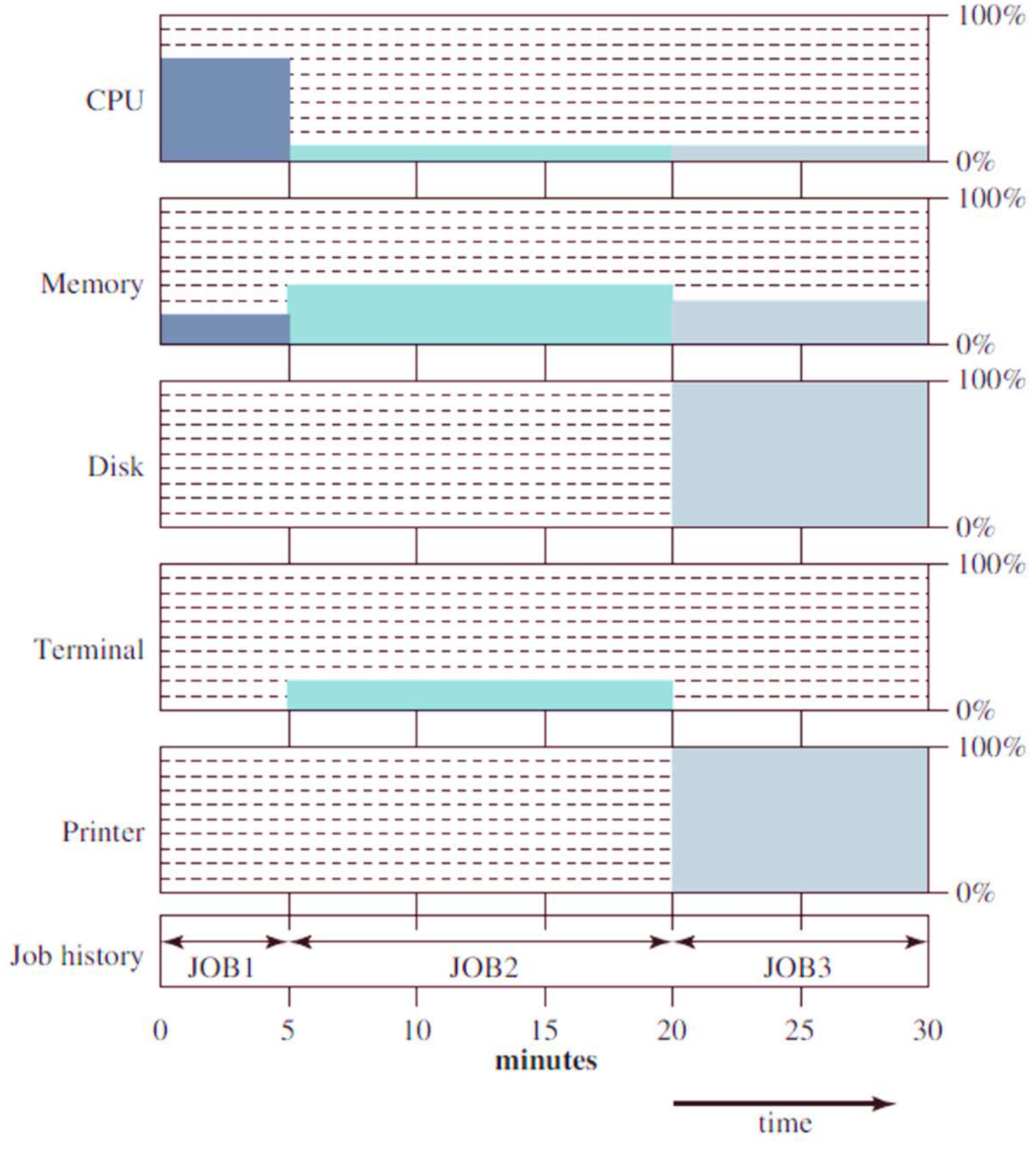
- Example:
  - Consider a computer with 250MB of available memory, a disk, a terminal, and a printer



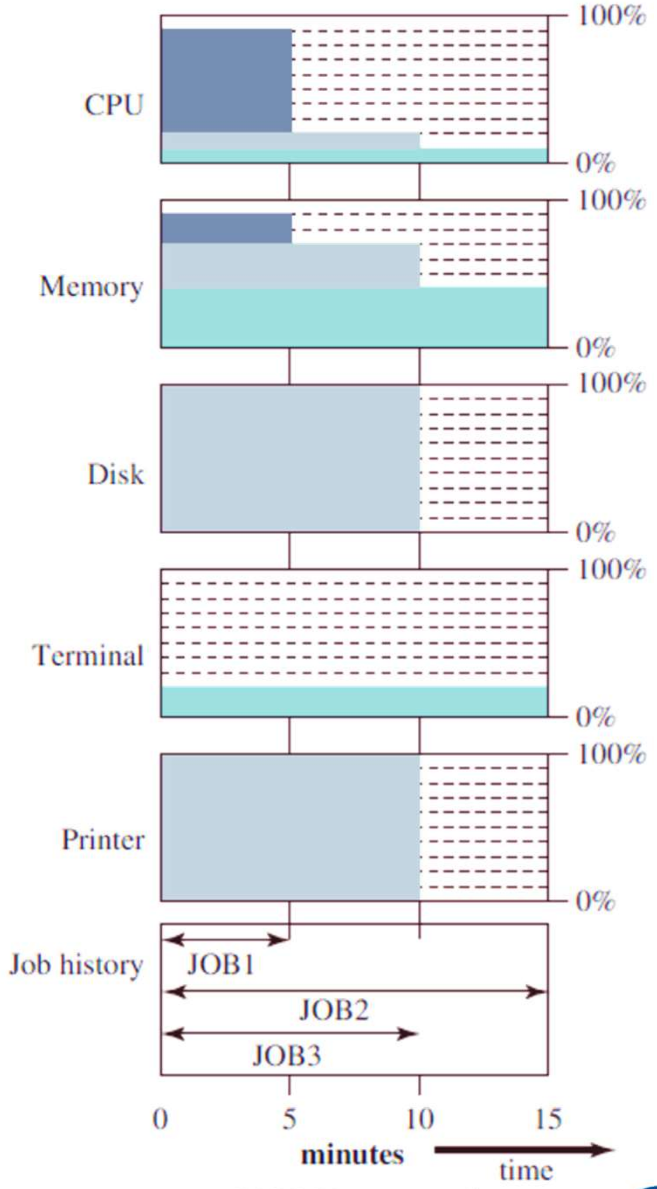
	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

3 Jobs

Multiprogrammed Batch Systems (Evolution of OS)



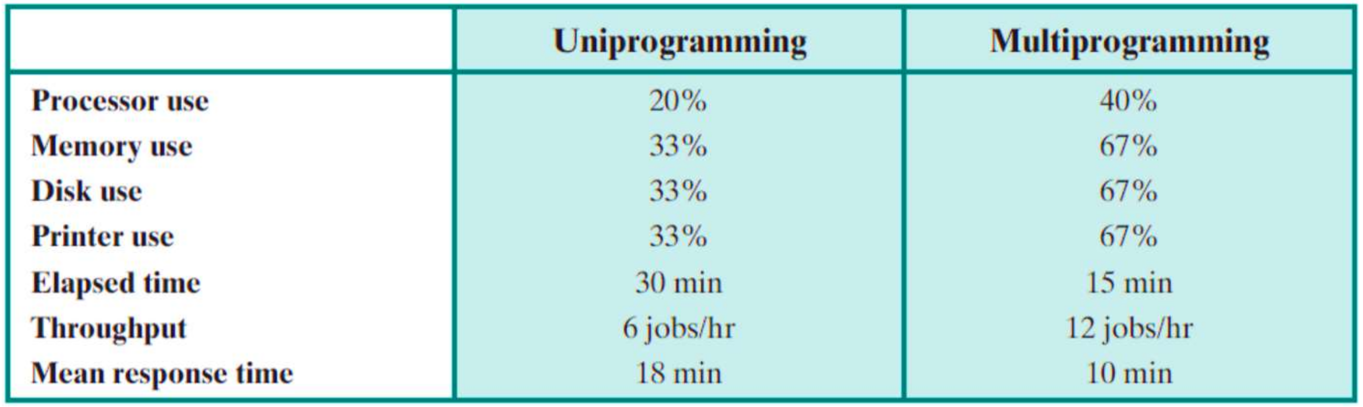
(a) Uniprogramming



(b) Multiprogramming

Multiprogrammed Batch Systems (Evolution of OS)

- Example:
  - Resource Utilization

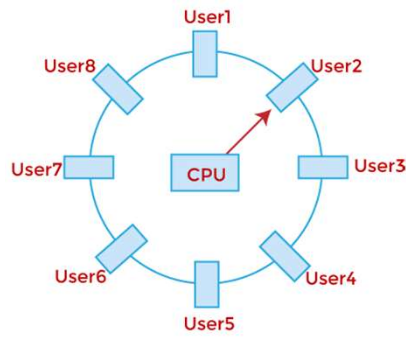


	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Resource Utilization

Time-Sharing Systems (Evolution of OS)

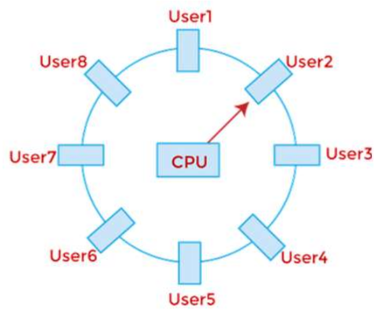
- Multiprogrammed batch system
  - Resource utilization is improved
  - User interaction still suffers



- Time-sharing
  - Multiprogramming: processor time is shared among multiple users
  - Multiple users simultaneously access the system through terminals
  - OS interleaves the execution of user programs in a short burst or quantum of computation

Time-Sharing Systems (Evolution of OS)

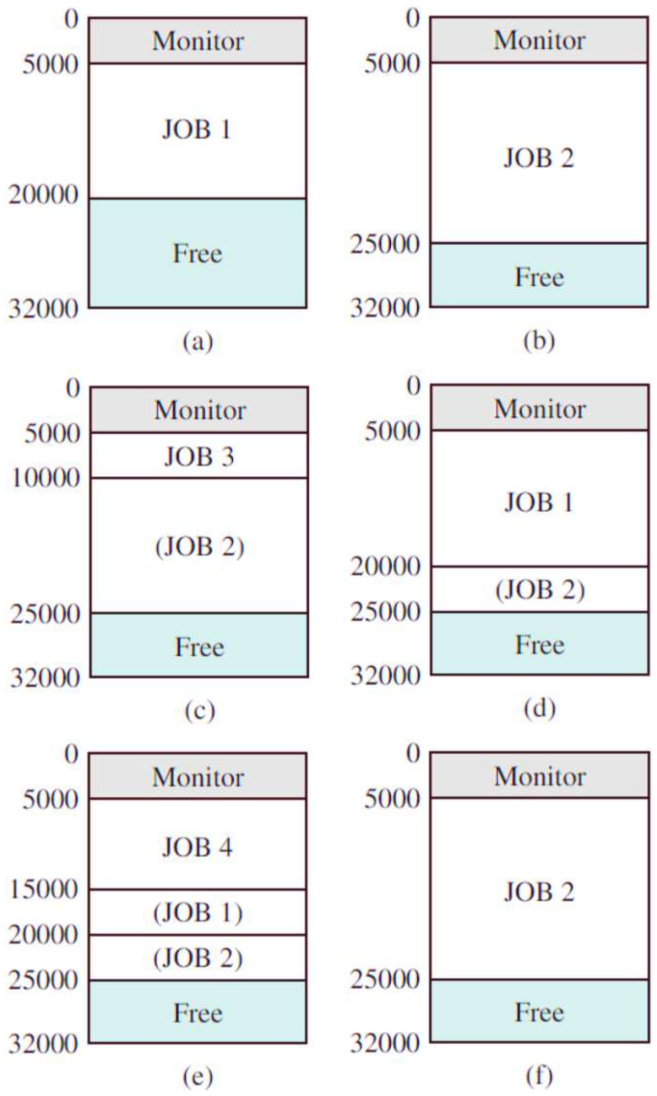
- Time-slicing
  - System clock generates an interrupt at a short interval
  - On each clock interrupt, OS regains control and assigns the processor to another user



- Batch Multiprogramming vs Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Time-Sharing Systems (Evolution of OS)



- Compatible Time-Sharing System (CTSS)
  - Clock interrupts at every 0.2 sec

a) JOB1 is loaded
b) JOB2 is loaded
c) JOB3 is loaded (part of JOB2 remains in mem)
d) JOB1 is loaded (part of JOB2 remains in mem)
e) JOB4 is loaded (parts of JOB1 and JOB2 are in mem)
f) JOB2 is loaded (only the first 20000 bytes are loaded)

Time-Sharing Systems (Evolution of OS)

- New issues raised by time-sharing and multiprogramming
  - Memory protection
    - Prevent other programs from modifying others' data
  - File system protection
    - Only authorized users have access to a particular file
  - Contention for resources
    - Printers, storage devices,…