Silberschatz and Galvin

Chapter 1

Introduction

Chapter Overview

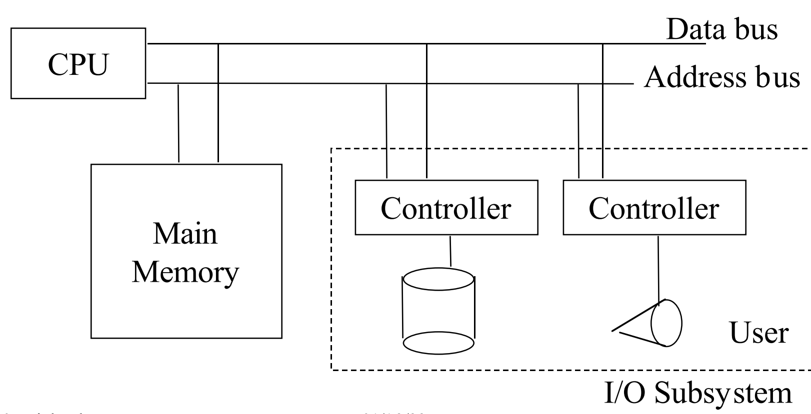
- What is an operating system?
- History of operating systems
  - structure
  - tradeoffs

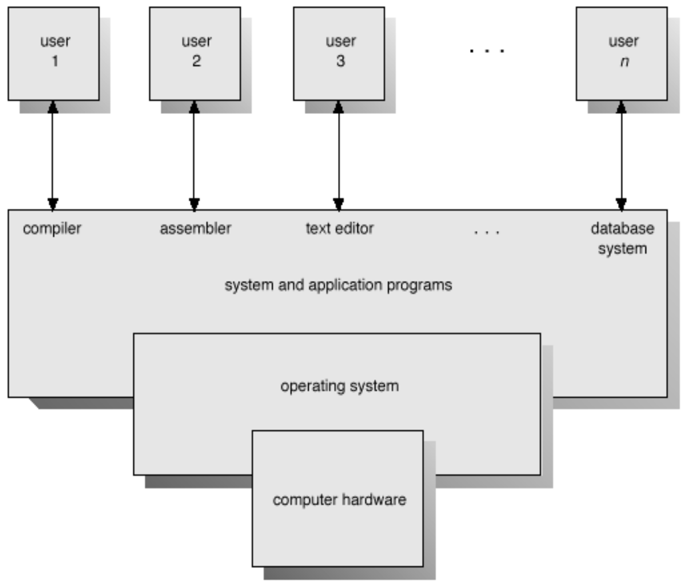
What is an Operating System?

- Computer system: hardware, operating system, application programs, users
- Computer hardware: von Neumann architecture: CPU, memory, input/output
- Applications programs: compilers, assemblers, text editors, utilities, etc....
- Operating system: *interface* between hardware and applications programs

Von Neumann Architecture

(From Bic and Shaw)





Operating System Definitions

- Resource allocator--manages and allocates resources
- Control program--controls the execution of user programs and operation of I/O devices
- Kernel--the one program running at all times (all else being application programs)

Operating System

- OS balances conflicting needs of users and programs. *Coordinator*. Permits multiple activities to coexist in efficient and fair manner. Implements "policy" based on assumptions
  – Is hardware cheap or expensive?
  – Interactive response time vs. wall clock time
  – Protect users or facilitate sharing?
- How encompassing is OS? Kernel concept. Is CLI in OS?

Historical Overview

- Early assumption
  – Hardware (very!) expensive and rare when compared to people time
  – Goal: make more efficient use of hardware even at expense of personal productivity
- Modern assumption
  – Hardware cheap. People are expensive.

1940's: No operating system

- Programmer writes in machine language, enters program directly (e.g., switches), operates computer
- Dedicated computer and peripherals; programmer=operator
- Different environments for different tasks.
- Manual scheduling. Organizational factors
- Perhaps have common subroutine library

1950's: Simple batch processing

- Programmer <> Operator
- Resident monitor (computer program): load and run, dump if exception
- "Batching" jobs ("automatic job sequencing")
- JCL (Job Control Language)
- One job at a time but maximize hardware use: off-line operation, buffering, interrupt handling, spooling, job scheduling (e.g., by time, subsystem, etc..)

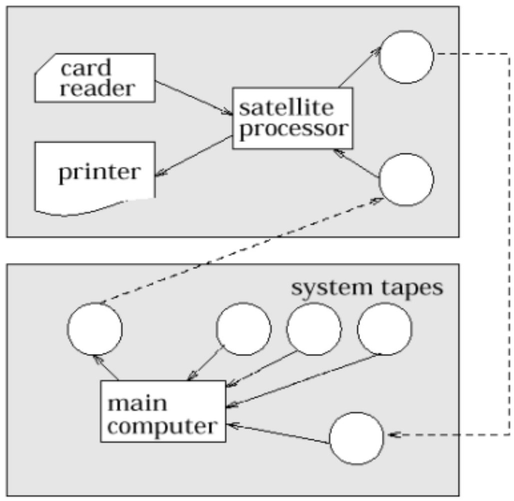
JCL (Job Control Language) OS/360

```
//QUESTNAR     JOB     (204121),MARCO.POLO,MSGLEVEL=1
//             EXEC    ASMFCG
//ASM.SYSIN    DD      *
     Program to be assembled
/*
//GO.OBJECT    DD      DSNAME=USERLIB,DISP=OLD
//             DD      *
     Object deck of subroutine
/*
//GO.SYSPRINT  DD      SYSOUT=A,DCB=(BLKSIZE=133)
//GO.INDATA    DD      DISP=OLD,UNIT=TAPE9,
//             DSNAME=QUEST214,VOLUME=SER=102139
//GO.SYSIN     DD      *
     Data cards, perhaps control cards for the program
/*
```


Off-line operation

- Load jobs into memory from tapes, not directly from cards
- Tape units are faster than card readers
- Application programs act as before
- Possible to use multiple reader-to-tape and tape-to-printer systems for one CPU

Off-line operation



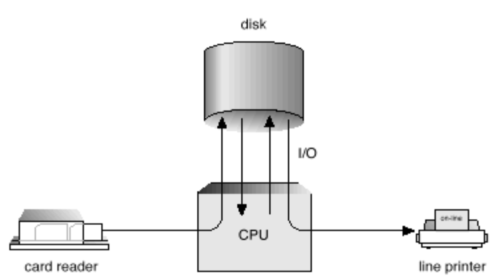
Early 1960's: Multiprogramming and multiprocessing

- **Multiprogramming**: several users share system at same time
  - batched: keep CPU busy by switching in other work when idle (e.g., waiting for I/O)
- Multitasking (timesharing): frequent switches to permit interactive use (extension of multiprogramming)
- **Multiprocessing**: several processors are used on a single system

Spooling

- Overlaps I/O of one job with computation of another job.
- While executing a job, the OS
  - Reads next job from card reader into storage area on disk (job queue)
  - Outputs printout of previous job from disk to printer
- Issue: what job to select to run next?

Spooling



Mid-1960's to mid-1970's: General purpose systems

- Large and expensive (e.g., OS/360)
  - 100k's of lines of code
  - hundreds to thousands of development man-years
  - complex, asynchronous, ideosyncratic to specific hardware
  - never completely debugged (1000's of release bugs)
  - hard to predict behavior, requires guesswork

Mid-1960's to mid-1970's

- OS begins to be treated as subject area
  - formerly collection of individual problems
  - basic concepts becoming standardized; theoretical underpinnings developed
  - research: concurrency, protection, scheduling (e.g., avoid thrashing), portability, maintainability (e.g., kernels)
  - research systems (e.g., Project MAC, THE)

Mid 1970's to present

- Cheap hardware, very expensive people
- OS in support of single user or small group of cooperating users
- Single process support evolves to multiple process support
- Device independent standards; commercial, defacto, and formal (MS-DOS, Unix, POSIX, etc.)
- Support for window packages, etc.

Two interesting special cases

- Distributed systems
  – tightly coupled (shared memory and clock) vs. loosely coupled (distributed)
  – issues of resource sharing, load sharing, reliability, communication
- Real-time systems
  – obligation to complete processing to meet defined constraints. Often conflicts with timesharing