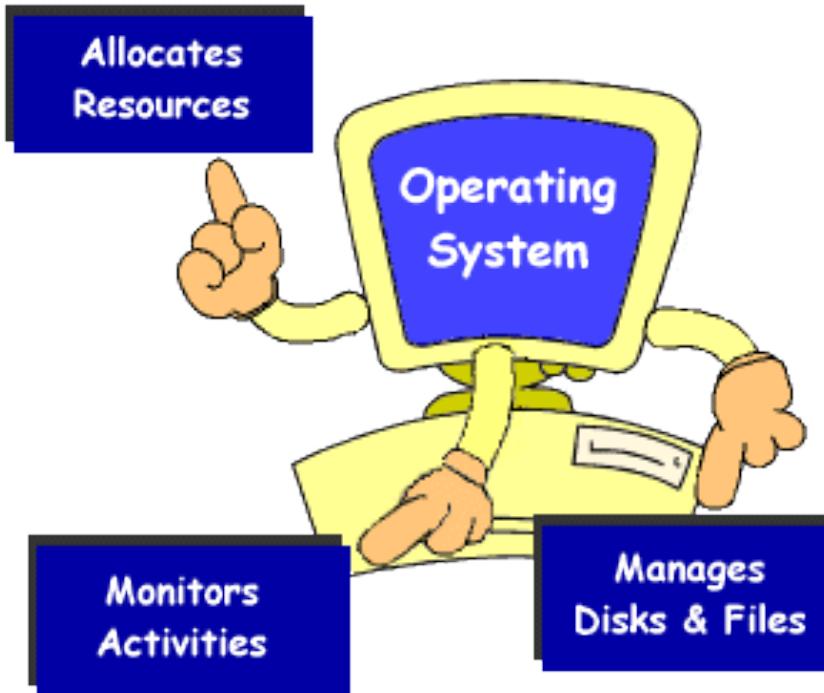




Hệ điều hành

Operating System



Google™



gOS





Thông tin cần biết

■ Tài liệu tham khảo

- [1] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, “Operating Systems: Three Easy Pieces”, 0.9v, 2014.
<http://pages.cs.wisc.edu/~remzi/OSTEP/>

[2] Silberschatz et al, “*Operating System Principles*”, 9th Ed., 2012.

[3] A. Tanenbaum, “*Modern Operating Systems*”, Prentice Hall, 4nd Ed., 2014.

■ Điểm môn học

- Thi cuối kỳ 50%
- Bài tập lớn 30%
- Thực hành 10%
- Bài tập 10%

■ Liên lạc

Phone: 8.647.256(5840)

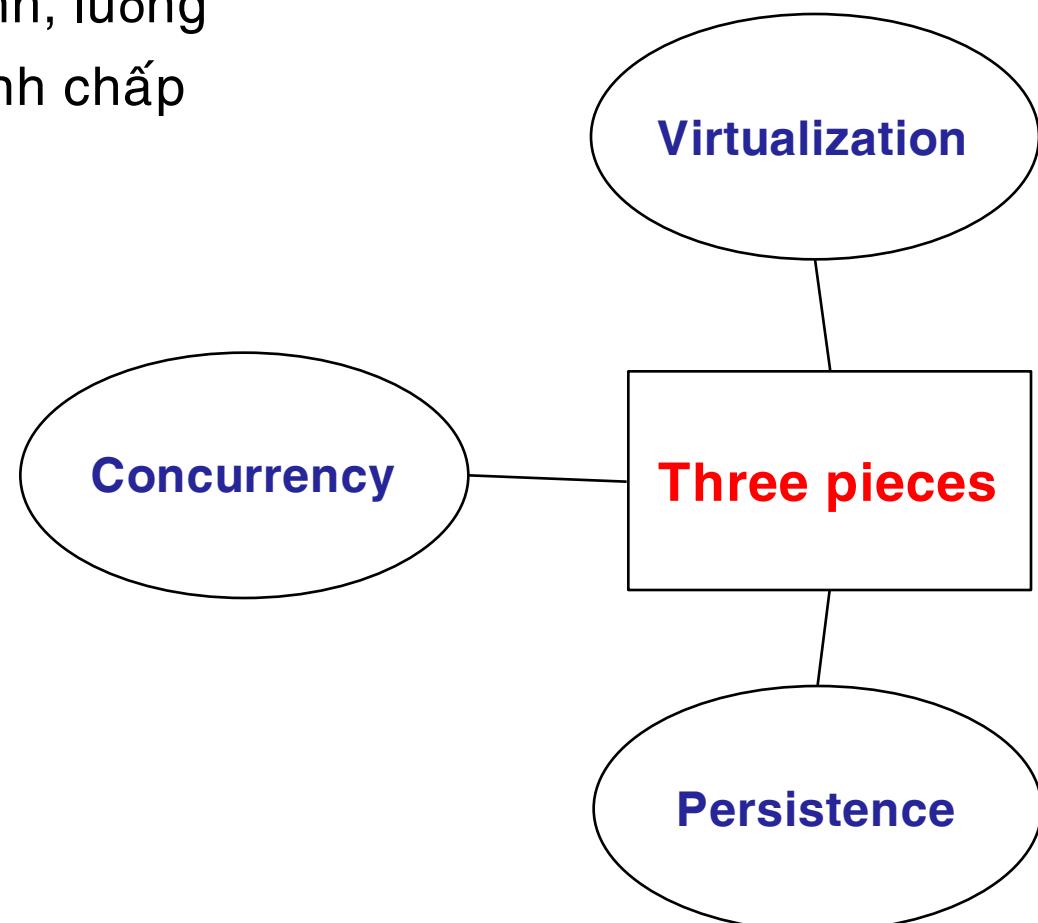
ltvan@hcmut.edu.vn

■ (Tập slide này có sử dụng slide từ các nguồn khác.)



Nội dung

- Tổng quan
- Khái niệm về quá trình
- Định thời biểu cho quá trình, luồng
- Đồng bộ và giải quyết tranh chấp
- Quản lý bộ nhớ
- Thay thế trang
- Quản lý nhập xuất
- Hệ thống file





Course outcome

CO - 1	define the functionality that a modern operating system must deliver to meet a particular need.
CO - 2	apply mechanisms that are useful to realize concurrent systems and describe the benefits of each.
CO - 3	compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems.
CO - 4	explain virtual memory and its realization in hardware and software.
CO - 5	compare and contrast different approaches to file organization, recognizing the strengths and weaknesses of each.



Chương 1: Tổng Quan

■ Giới thiệu

- Cấu trúc luận lý của hệ thống máy tính
- Định nghĩa hệ điều hành
- Các chức năng chính của hệ điều hành

■ Quá trình phát triển

- Máy tính lớn (mainframe system)
- Máy để bàn (desktop system)
- Đa xử lý (multiprocessor system)
- Phân bố (distributed system)
- Thời gian thực (real-time system)
- Cầm tay (handheld system/mobile system)



What is an operating system?





Định nghĩa

■ Hệ điều hành là gì?

- “*Phần mềm trung gian*” giữa phần cứng máy tính và người sử dụng, có chức năng *điều khiển phần cứng* và cung cấp các *dịch vụ cơ bản* cho các ứng dụng

■ Mục tiêu

- Giúp người dùng dễ dàng sử dụng hệ thống
- Quản lý và cấp phát tài nguyên hệ thống một cách hiệu quả [yếu tố kinh tế]

Người dùng



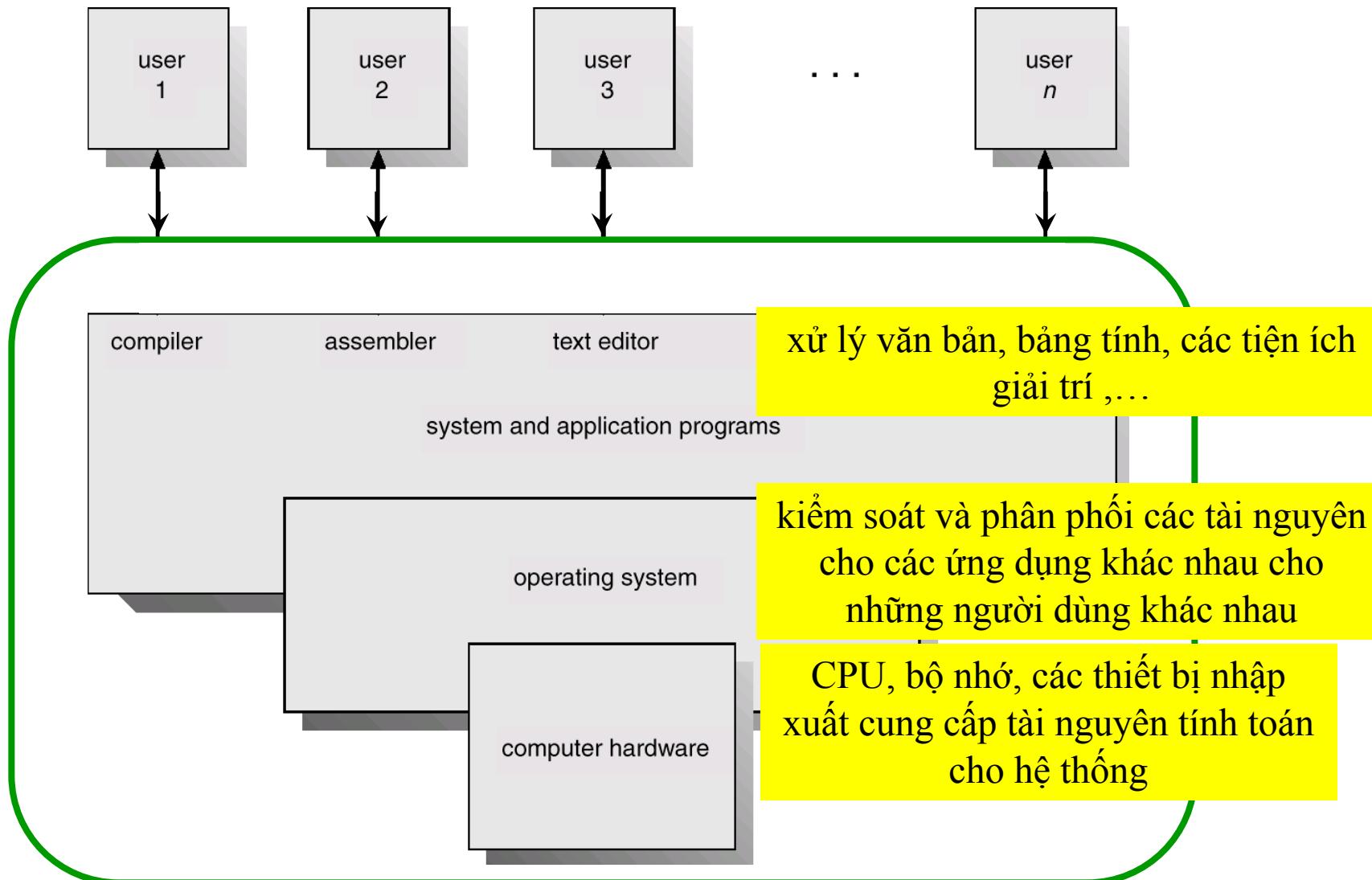
Các ứng dụng

Hệ Điều Hành

Phần cứng



Các thành phần của hệ thống máy tính



1.1 Fig 1.1



Các chức năng chính của OS

- Phân chia thời gian xử lý trên CPU (định thời)
- Phối hợp và đồng bộ hoạt động giữa các quá trình
- Quản lý tài nguyên hệ thống hiệu quả
- Kiểm soát quá trình truy cập, bảo vệ hệ thống
- Duy trì sự nhất quán của hệ thống, kiểm soát lỗi và phục hồi hệ thống khi có lỗi xảy ra
- Cung cấp giao diện làm việc thuận tiện cho người dùng



Lịch sử phát triển

■ *Máy tính lớn* (mainframe)

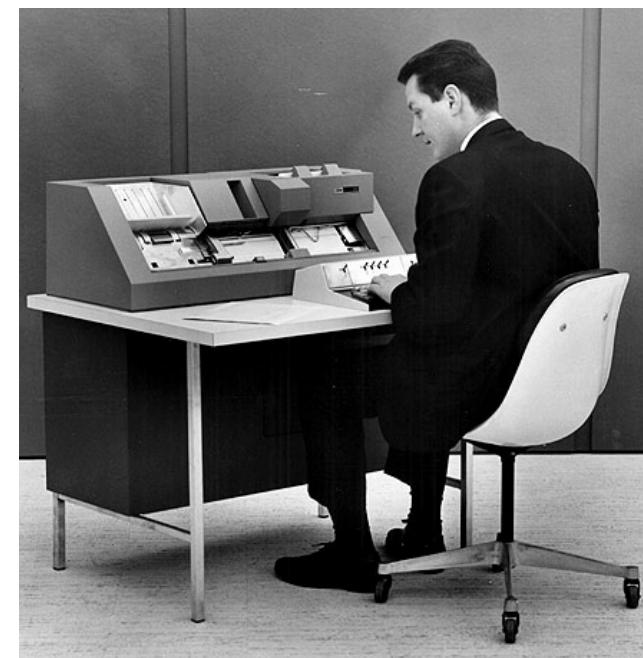
- Xử lý bó (batch, 1960)
- Đa chương (multiprogrammed, 1970)
- Đa nhiệm (time-sharing, multitasking; 1970)



Lịch sử phát triển

- (Mainframe) *Batch system*, 1960
 - I/O: card đục lỗ, băng từ (tape), line printer
 - Cần có người vận hành (operator)
 - Giảm setup time bằng cách ghép nhóm (batching) công việc (job)
 - ▶ Vd: ghép các công việc cùng sử dụng trình biên dịch Fortran
 - Tự động nạp lần lượt các công việc từ card reader

```
MSGBOX "HELLO WORLD" MODE(INFO); WRITE CONSOLE LVCURRENTDATE; LEAVESCRIPT;
```



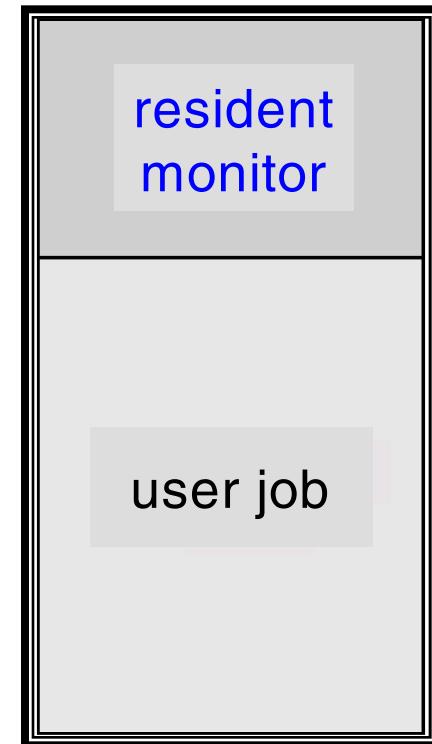


Lịch sử phát triển (tt)

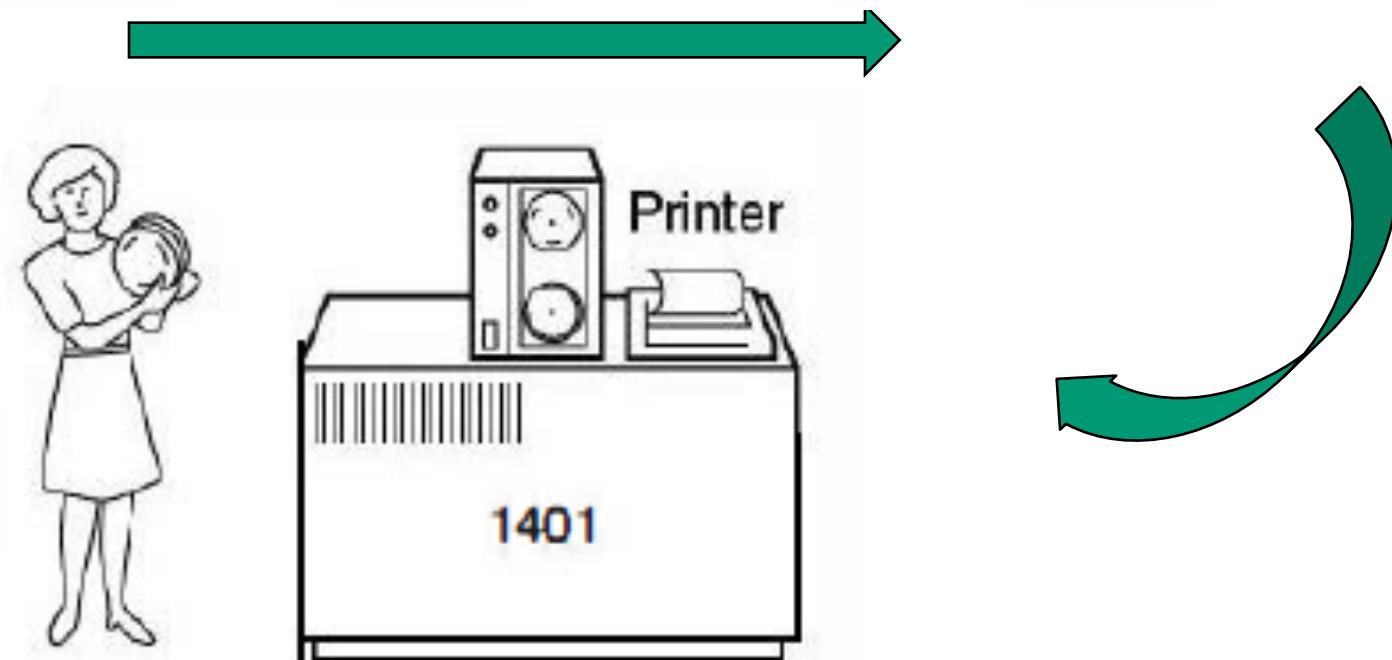
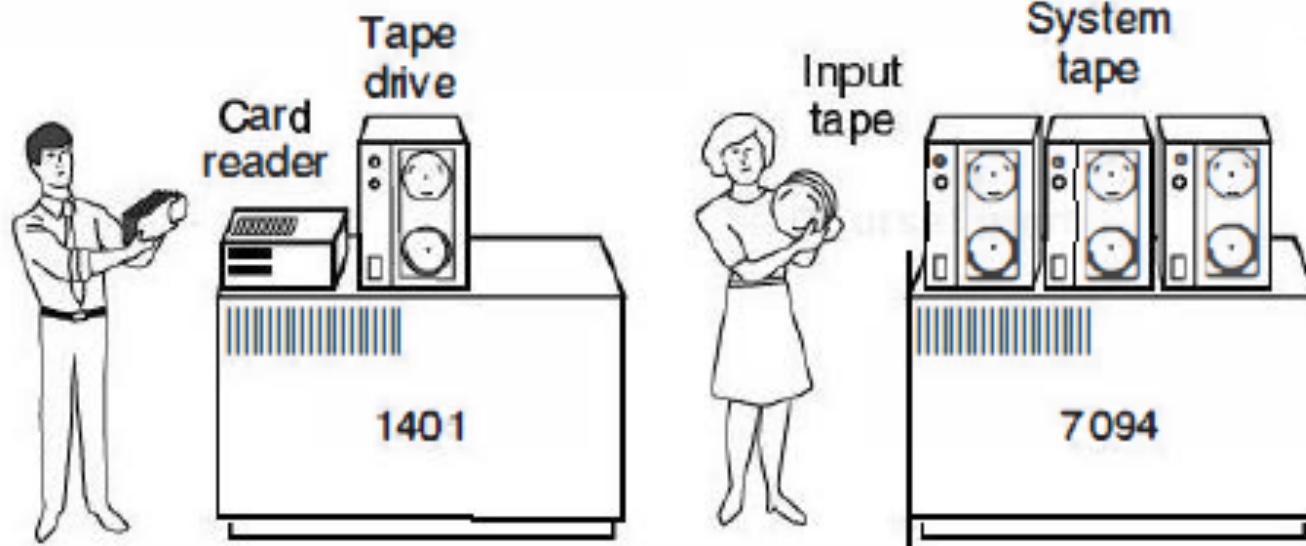
- (Mainframe) *Batch system* đơn giản

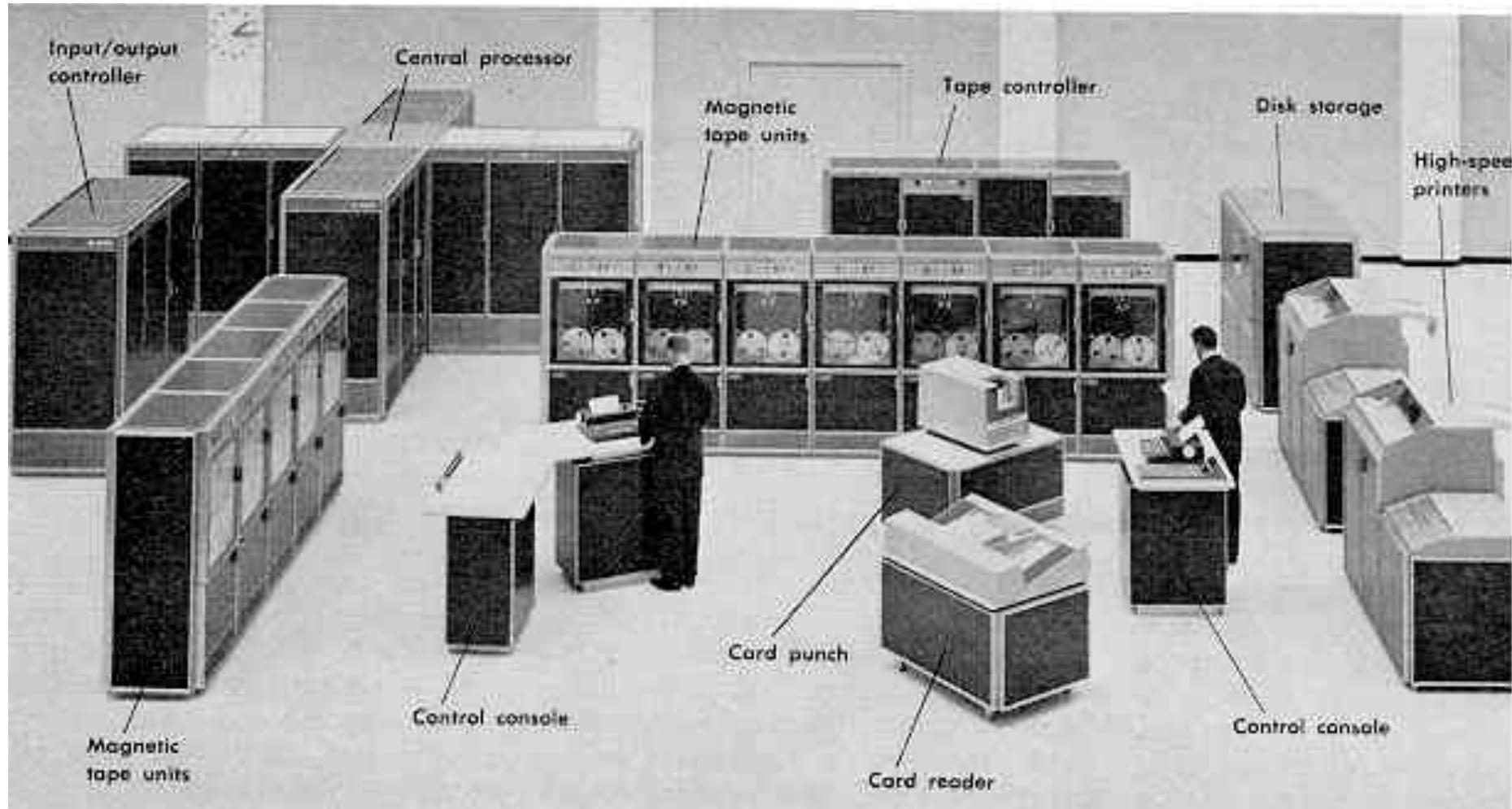
- *Uniprogramming*

- ▶ Khi một job thực thi xong, quyền điều khiển trở về hệ điều hành ('resident monitor')
 - ▶ Resident monitor đọc job kế tiếp từ card reader vào bộ nhớ
 - ▶ Thực thi job cho đến khi xong



Layout bộ nhớ





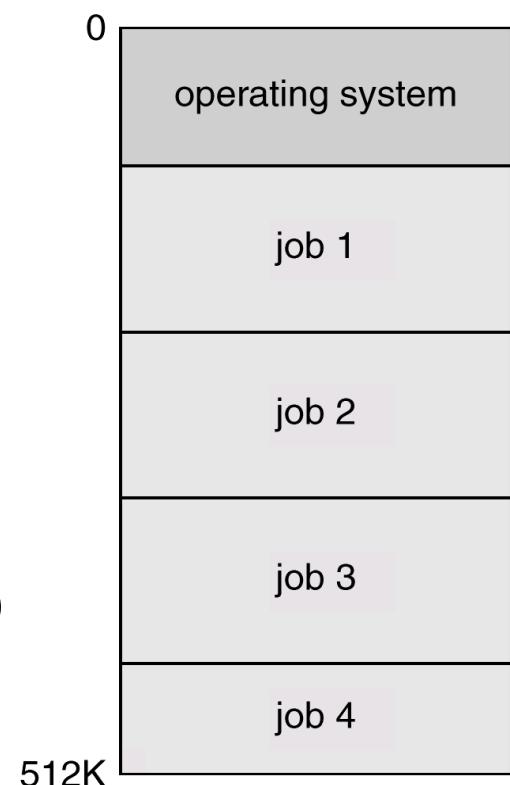
Mainframe computer in 1967



Lịch sử phát triển hệ điều hành (tt)

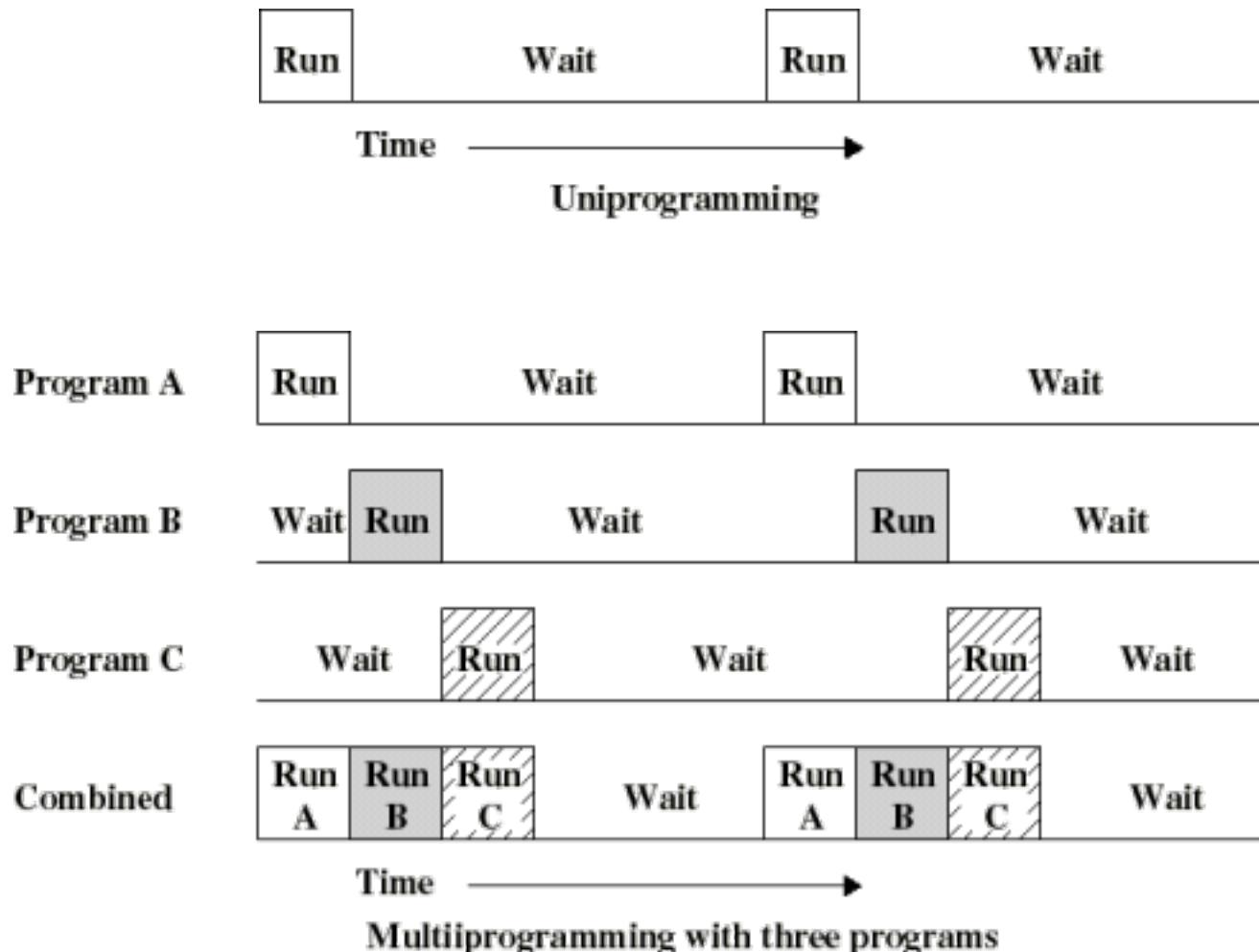
■ (Mainframe) *Multiprogrammed system*

- Nhiều job được giữ đồng thời trong bộ nhớ
- CPU thực thi luân phiên giữa các job trong bộ nhớ
- Tận dụng được thời gian rảnh, tăng *hiệu suất sử dụng* CPU
- Yêu cầu đối với hệ điều hành
 - Định thời job
 - Định thời CPU
 - Quản lý bộ nhớ (memory management)
 - Cấp phát tài nguyên (đĩa, máy in,...)
 - Bảo vệ





Lịch sử phát triển hệ điều hành (tt)





Lịch sử phát triển hệ điều hành (tt)

- (Mainframe) *Time-sharing system*
 - Multiprogrammed system không cung cấp khả năng tương tác hiệu quả với user
 - CPU thực thi luân phiên giữa các công việc
 - ▶ Mỗi công việc được chia một phần nhỏ thời gian CPU (*time slice, quantum time*)
 - ▶ Cung cấp tương tác giữa user và hệ thống với *thời gian đáp ứng* (response time) nhỏ (1 s)



Computer terminal, 1982



Lịch sử phát triển hệ điều hành (tt)

- Yêu cầu đối với OS trong hệ thống time-sharing
 - Quản lý bộ nhớ
 - ▶ Virtual memory
 - Quản lý các quá trình
 - ▶ Định thời CPU
 - ▶ Đồng bộ các quá trình (synchronization)
 - ▶ Giao tiếp giữa các quá trình (process communication)
 - ▶ Vấn đề deadlock
 - Quản lý hệ thống file, hệ thống lưu trữ (memory system)
 - Cấp phát hợp lý các tài nguyên
 - Bảo vệ



Lịch sử phát triển hệ điều hành (tt)

- *Máy để bàn* (desktop system, personal computer)
 - Nhiều thiết bị I/O: bàn phím, chuột, màn hình, máy in,...
 - Phục vụ người dùng đơn lẻ
 - Mục tiêu chính của OS
 - ▶ Thuận tiện cho user và khả năng tương tác cao
 - ▶ Không cần tối ưu hiệu suất sử dụng CPU và thiết bị ngoại vi
 - Nhiều hệ điều hành khác nhau – MS Windows, Mac OS, Unix, Linux,...



1981: IBM 5150



Lịch sử phát triển hệ điều hành (tt)

- *Hệ thống song song* (parallel, multiprocessor, hay tightly-coupled system)
 - Nhiều CPU
 - Chia sẻ computer bus, clock
 - Ưu điểm
 - ▶ *System throughput*: càng nhiều processor thì càng nhanh xong công việc (\rightarrow dự đoán thời tiết)
 - ▶ Multiprocessor system ít tốn kém hơn multiple single-processor system: vì có thể dùng chung tài nguyên (đĩa,...)
 - ▶ *Độ tin cậy*: khi một processor hỏng thì công việc của nó được chia sẻ giữa các processor còn lại



Lịch sử phát triển hệ điều hành (tt)

■ Phân loại hệ thống song song

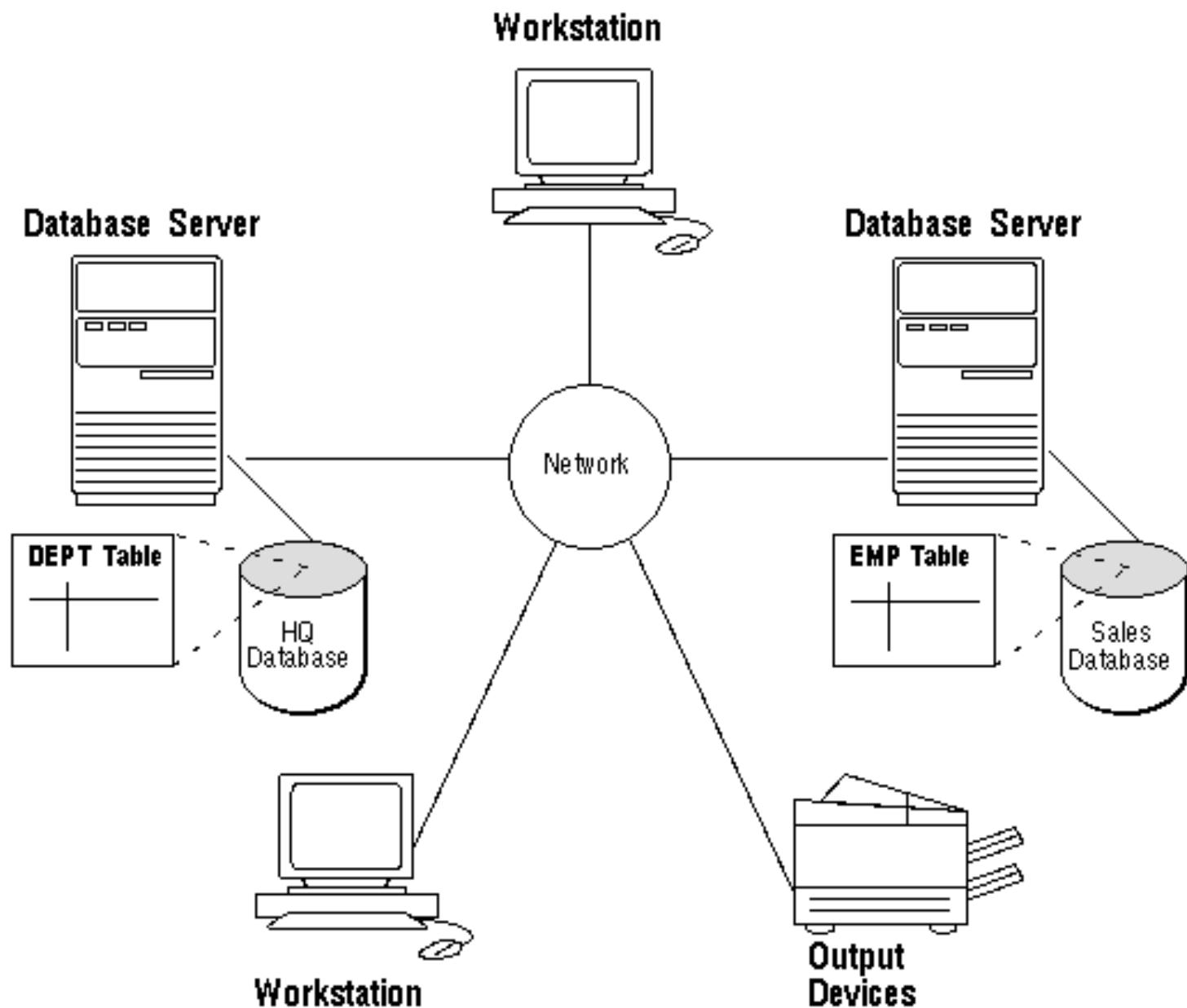
- *Đa xử lý đối xứng* (symmetric multiprocessor – SMP)
 - ▶ Mỗi processor vận hành một identical copy của hệ điều hành
 - ▶ Các copy giao tiếp với nhau khi cần
- *Đa xử lý bất đối xứng* (asymmetric multiprocessor)
 - ▶ Mỗi processor thực thi một công việc khác nhau
 - ▶ **Master** processor định thời và phân công việc cho các **slave** processor



Lịch sử phát triển hệ điều hành (tt)

■ *Hệ thống phân bố* (distributed system, loosely-coupled system)

- Mỗi processor có bộ nhớ riêng, các processor giao tiếp qua các kênh nối như mạng, bus tốc độ cao, leased line
- Người dùng chỉ thấy một hệ thống đơn nhất
- Ưu điểm
 - ▶ Chia sẻ tài nguyên (resource sharing)
 - ▶ Chia sẻ sức mạnh tính toán (computational sharing)
 - ▶ Độ tin cậy cao (high reliability)
 - ▶ *Độ sẵn sàng* cao (high availability): các dịch vụ của hệ thống được cung cấp liên tục cho dù một thành phần hardware trở nên hỏng





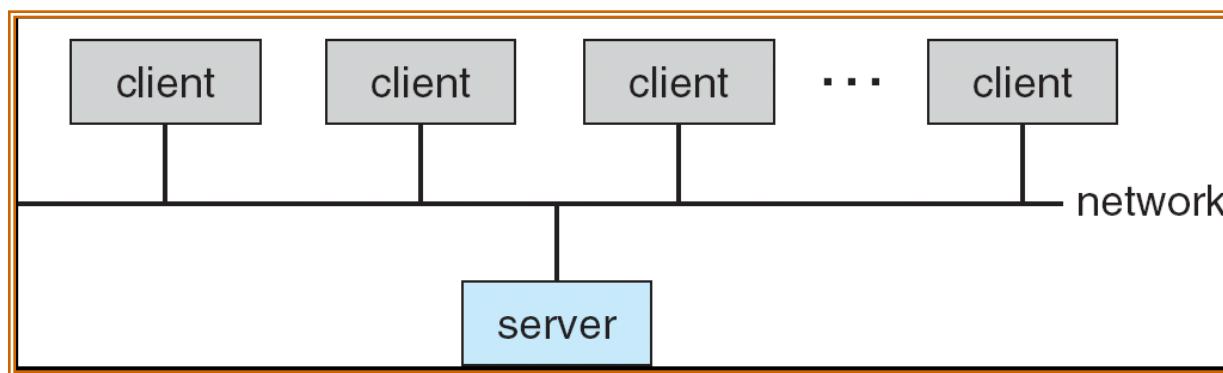
Lịch sử phát triển hệ điều hành (tt)

■ Hệ thống phân bố (tt)

Các mô hình hệ thống phân bố

- *Client-server*

- ▶ Server: cung cấp dịch vụ
- ▶ Client: có thể sử dụng dịch vụ của server





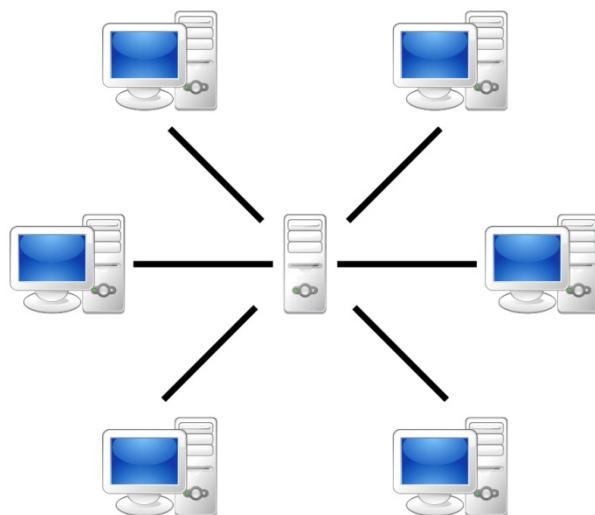
Lịch sử phát triển hệ điều hành (tt)

■ Hệ thống phân bố

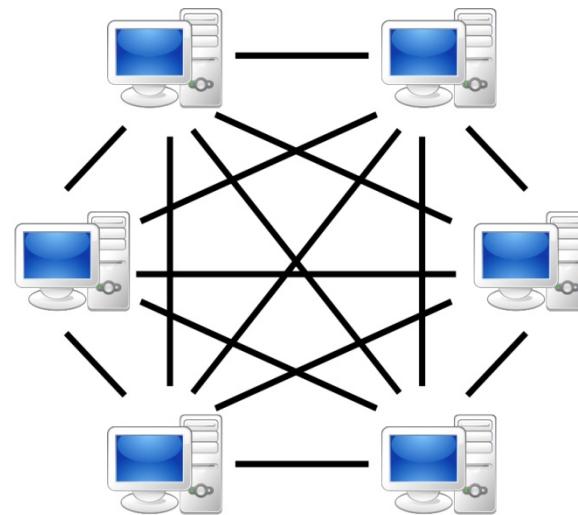
Các mô hình hệ thống phân bố (tt)

- *Peer-to-peer* (P2P, 1990s)

- ▶ Các *peer* (máy tính trong hệ thống) đều ngang hàng nhau
- ▶ Không dựa trên cơ sở dữ liệu tập trung
- ▶ Các peer là tự trị
- ▶ Vd: mạng gnutella, một mạng trong Internet để chia sẻ file



Server-based



P2P-network



Lịch sử phát triển hệ điều hành (tt)

■ *Hệ thống thời gian thực* (real-time system)

- Điều khiển trong xe hơi, dây chuyền công nghiệp,...
- Ràng buộc về thời gian: đáp ứng của hệ thống phải thỏa thời hạn (deadline) để xử lý biến cố (event)

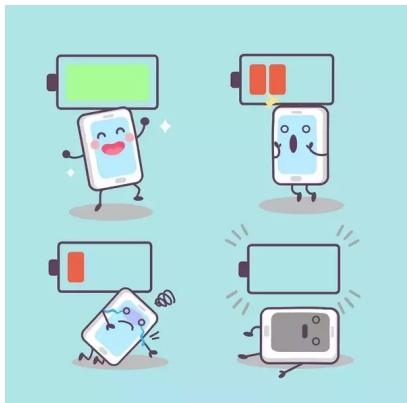
Phân loại

- *Hard real-time*
 - Điều khiển máy nổ, airbag trong xe hơi, robotics,...
 - Do hạn chế (hoặc không có) bộ nhớ thứ cấp, tất cả dữ liệu nằm trong bộ nhớ chính (RAM hoặc ROM)
 - Yêu cầu về thời gian đáp ứng/xử lý rất nghiêm ngặt
- *Soft real-time*
 - Multimedia, virtual reality
 - Yêu cầu mềm dẻo hơn về thời gian đáp ứng

Lịch sử phát triển hệ điều hành (tt)

■ Các hệ điều hành dành cho thiết bị di động

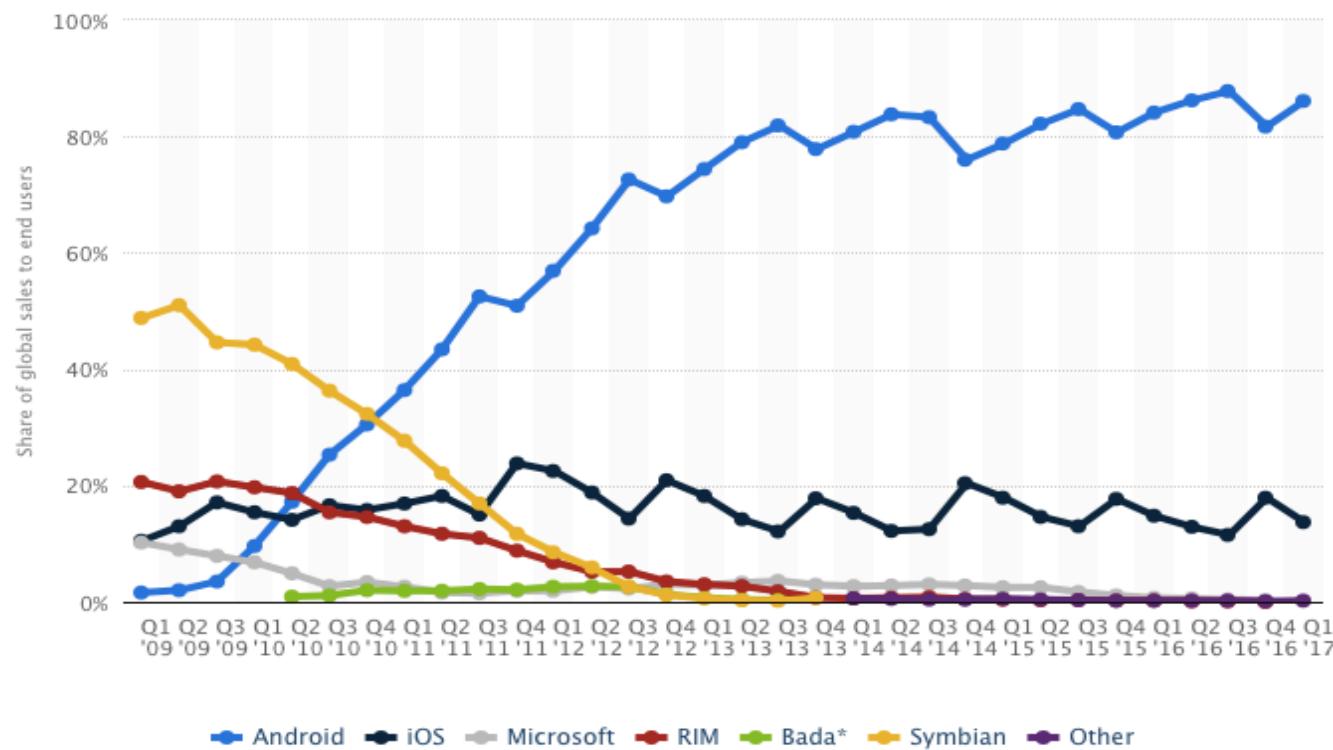
- ❑ iOS
- ❑ Android
- ❑ Blackberry
- ❑ Firefox OS
- ❑ Sailfish OS
- ❑ Tizen
- ❑ Ubuntu Touch
- ❑ Windows



Các hệ điều hành không còn được tiếp tục phát triển

❑ Symbian

— Palm OS



Global mobile OS market share 2009-2017, by quarter

© Statista 2017



History of Windows

- History of Windows

<https://www.youtube.com/watch?v=aqkkByP8RDM>

- History of Android

<https://www.youtube.com/watch?v=DNAQtgH1ErI>

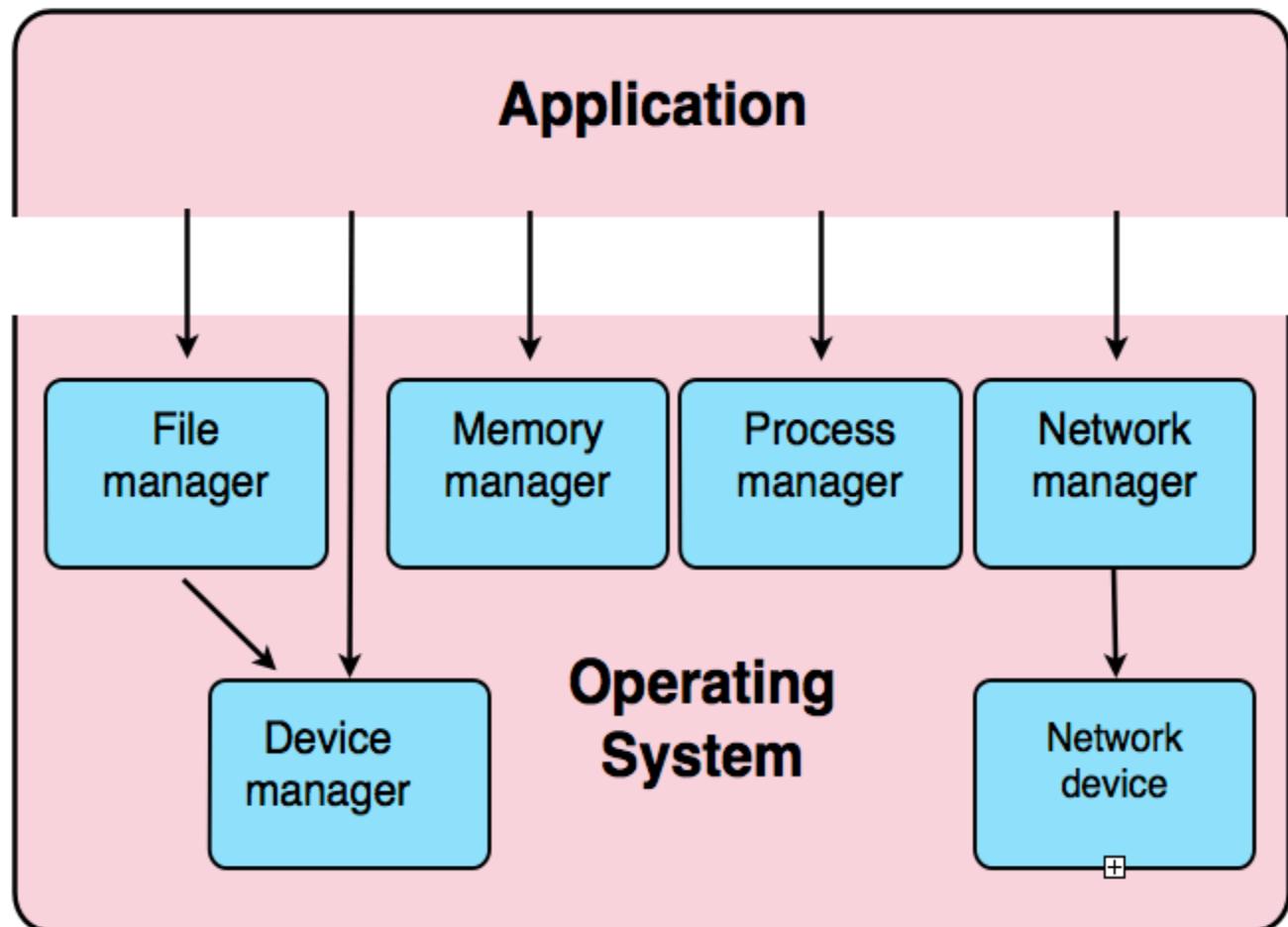
- History of Linux

<http://www.youtube.com/watch?v=0kteK4-RSJ8>

Cơ chế vận hành của hệ thống

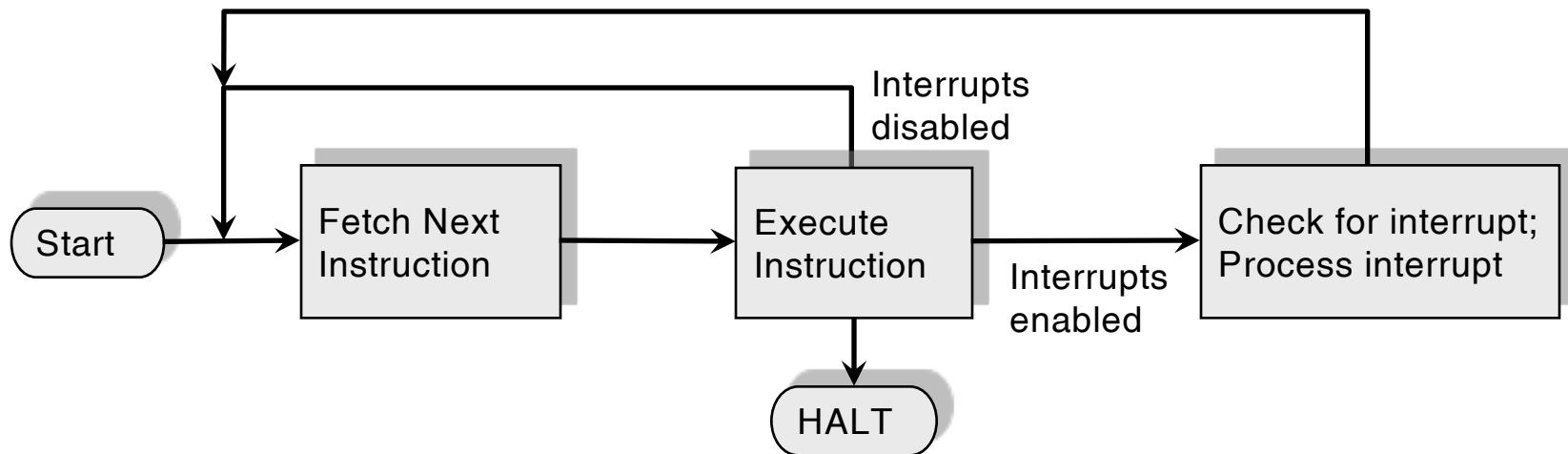


■ Cấu trúc hệ thống máy tính





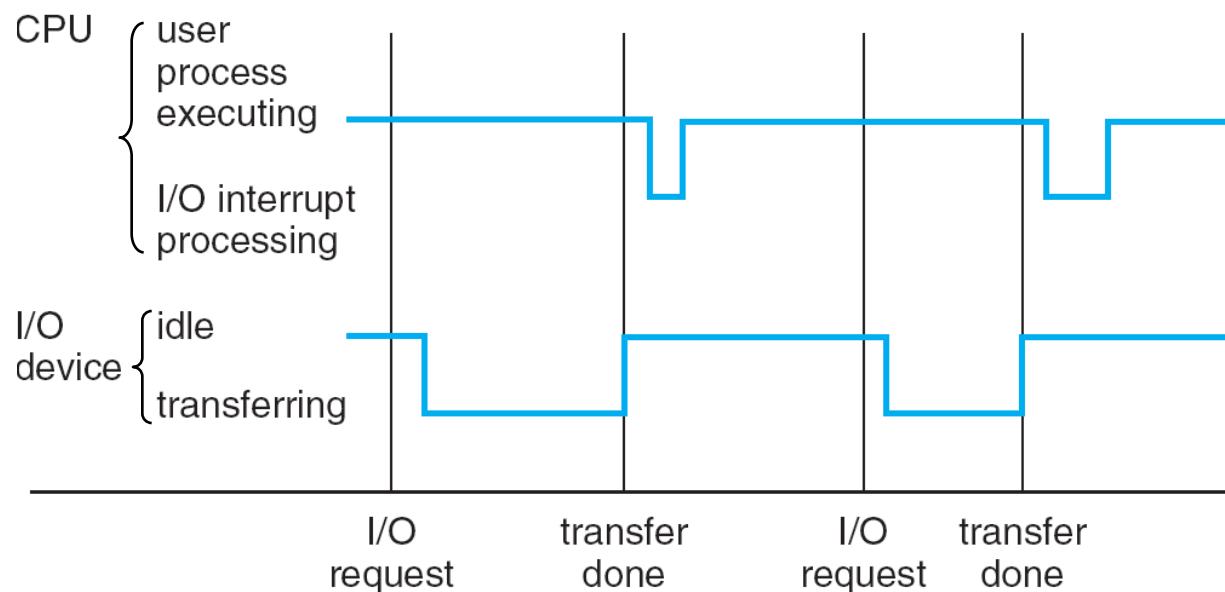
Chu trình hoạt động của CPU



Ngắt quãng

■ Phân loại: ngắt quãng do

- *Program*: tràn số học, chia cho 0, truy cập bộ nhớ bất hợp pháp
- *Timer*: cho phép CPU thực thi một tác vụ nào đó theo định kỳ
- *I/O*: kết thúc tác vụ I/O, xảy ra lỗi trong I/O
- *Hardware failure*: Hư hỏng nguồn, lỗi memory parity,...
- *Trap (software interrupt)*: yêu cầu dịch vụ hệ thống (gọi system call),...

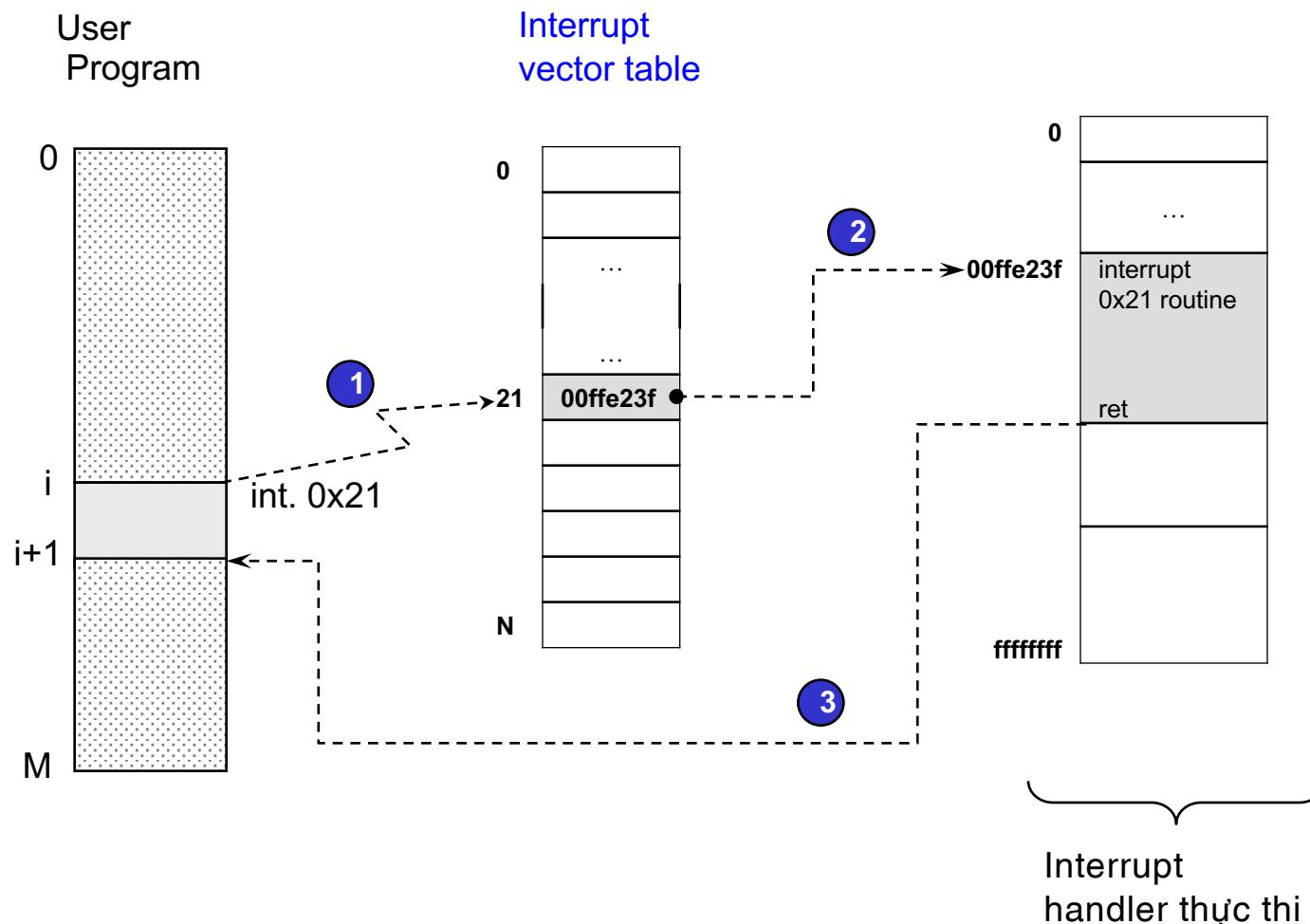


Lược đồ thời gian khi process có yêu cầu các tác vụ I/O



Quá trình xử lý ngắt quang

Xử lý trap tương tự xử lý HW interrupt



Cấu trúc hệ thống I/O



Điều khiển thiết bị I/O

Trên device controller có

- Thanh ghi lệnh (command / control register) để nhận yêu cầu I/O từ OS
- Thanh ghi trạng thái (status register) để báo OS tình trạng sẵn sàng / đang bận / lỗi của thiết bị
- Thanh ghi dữ liệu (data register) – OS đọc dữ liệu từ thiết bị hay ghi dữ liệu ra thiết bị qua thanh ghi này



Các kỹ thuật thực hiện I/O (1/2)

■ Kỹ thuật I/O dùng polling

- I/O code chờ thiết bị sẵn sàng bằng cách liên tục (như bằng cách dùng vòng lặp) kiểm tra tình trạng của thiết bị
- Khi thiết bị sẵn sàng, I/O code gửi lệnh, và đọc/ghi dữ liệu thông qua các thanh ghi của thiết bị



Các kỹ thuật thực hiện I/O (2/2)

■ Kỹ thuật I/O dùng **ngắt quãng** (interrupt-driven I/O)

- I/O code gửi lệnh, và đọc/ghi dữ liệu thông qua các thanh ghi của thiết bị
- Khi thiết bị hoàn tất lệnh I/O thì sẽ gây ngắt đến CPU



Truyền dữ liệu giữa bộ nhớ và thiết bị

■ Programmed I/O

- OS dùng CPU cycle để di chuyển dữ liệu giữa bộ nhớ và thiết bị
- Nhận xét: programmed I/O không hữu hiệu (cần nhiều CPU cycle) khi thực hiện di chuyển khối lượng lớn dữ liệu

■ Kỹ thuật Direct Memory Access (DMA)

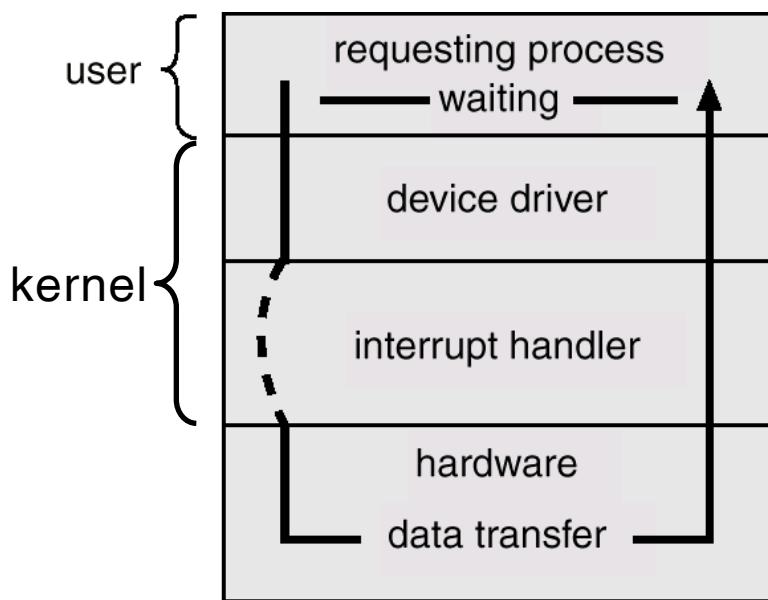
- DMA cần có phần cứng hỗ trợ đặc biệt, đó là DMA controller
- Kỹ thuật DMA thực hiện truyền dữ liệu trực tiếp giữa thiết bị I/O và bộ nhớ mà không cần sự can thiệp của CPU



Kỹ thuật thực hiện I/O ở mức ứng dụng

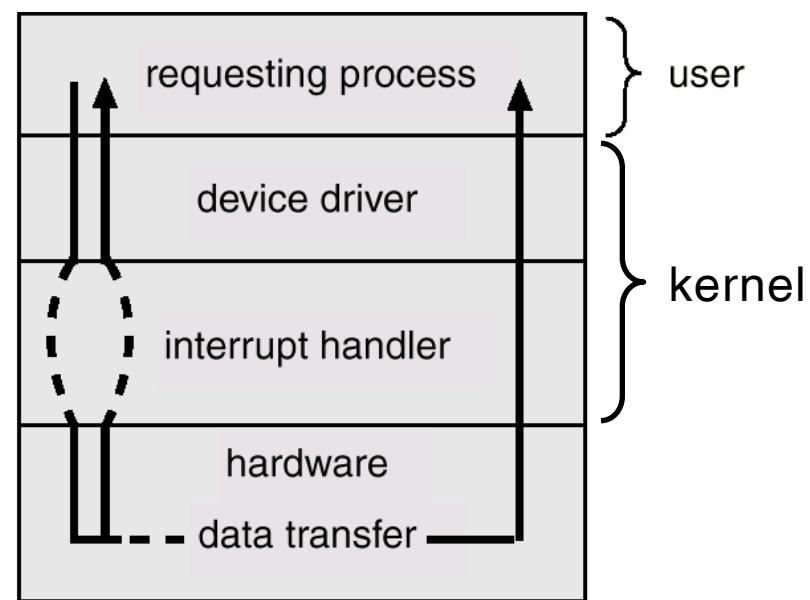
■ Blocking / nonblocking I/O

Blocking



Nonblocking, asynchronous

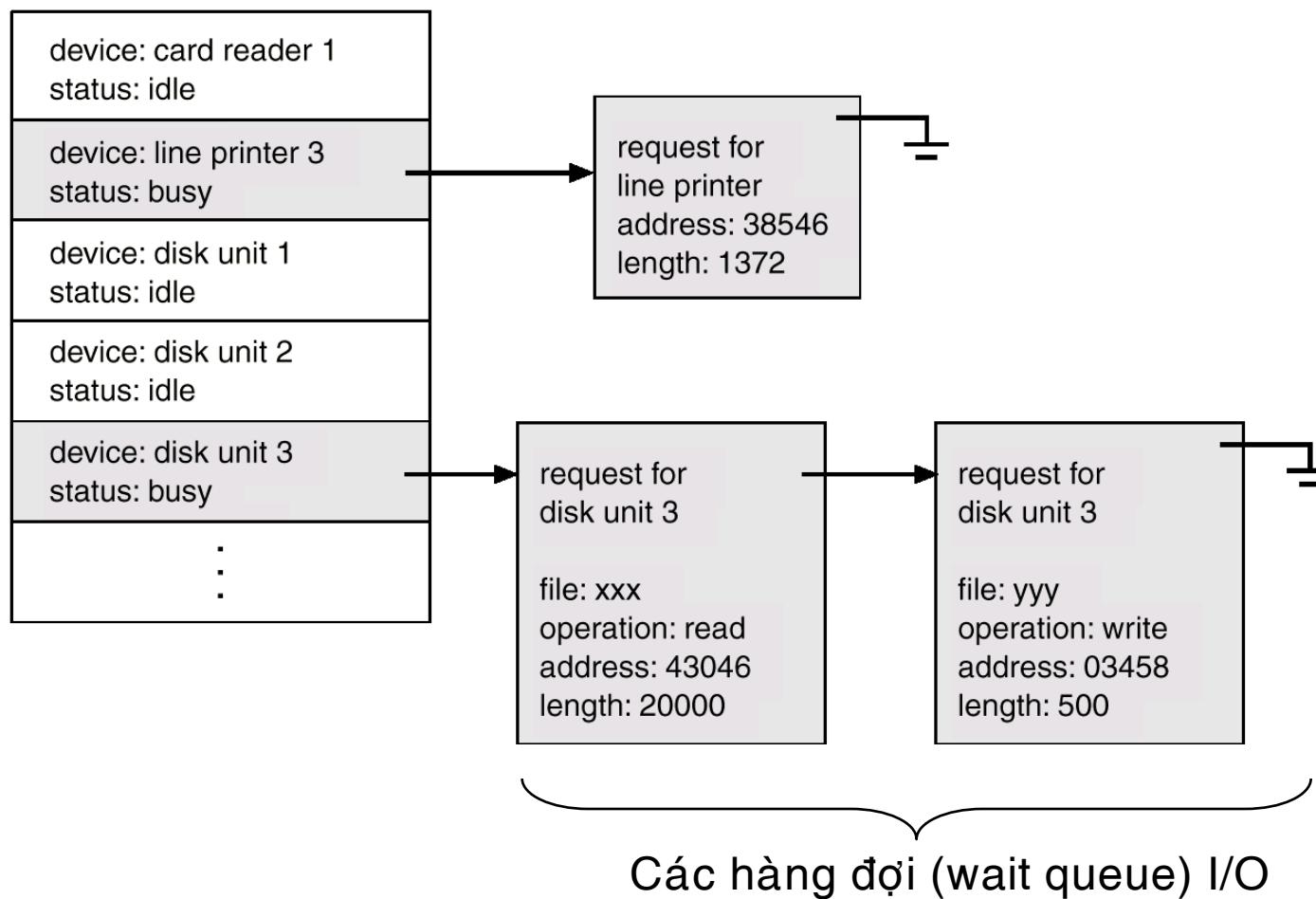
-- Thông báo I/O hoàn tất bằng signal hay callback



--- : bỏ qua đoạn mã

Quản lý các truy cập thiết bị I/O

■ Hàng đợi các yêu cầu I/O, vd



Cấu trúc & phân cấp hệ thống lưu trữ



Hệ thống lưu trữ

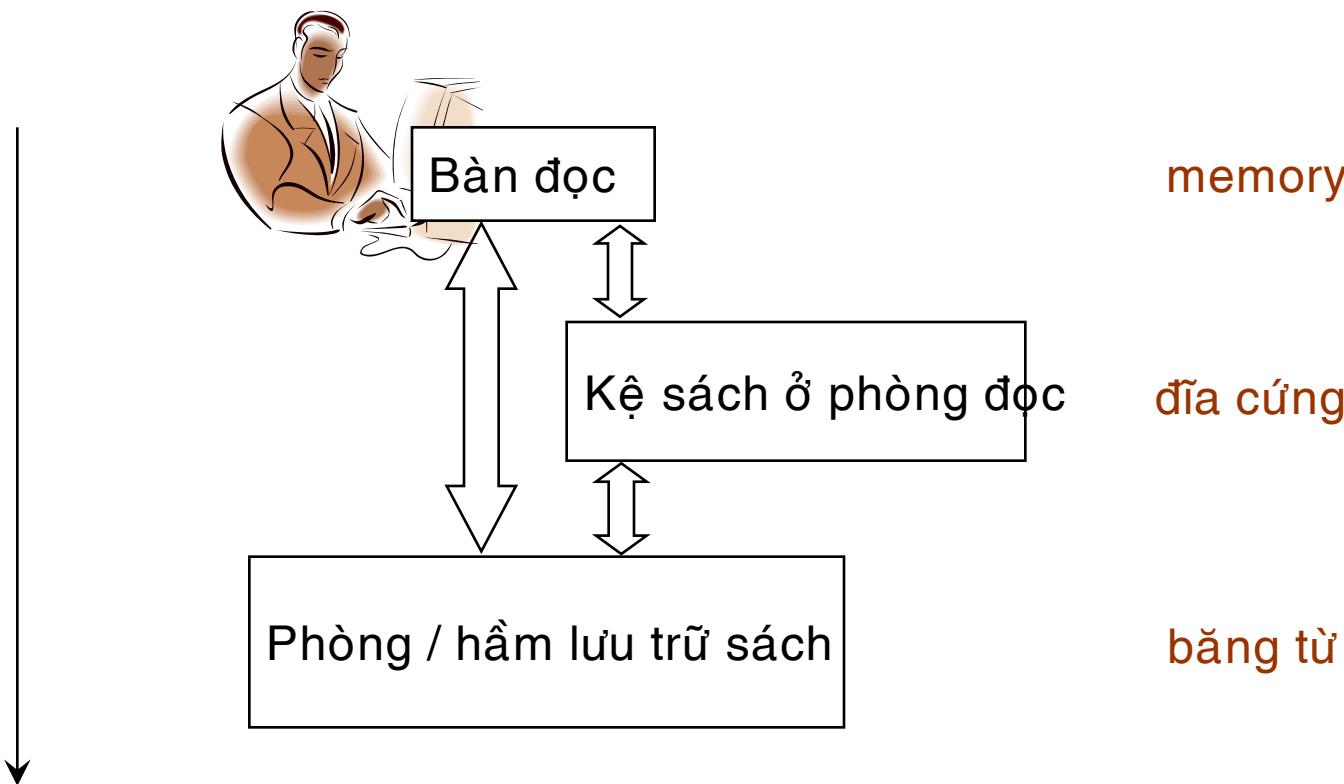
- Lưu trữ (memory, storage) là một trong những dạng thức I/O quan trọng
 - *Bộ nhớ chính* (main memory, primary memory)
 - ▶ Trực tiếp thì CPU chỉ có thể truy cập được các thanh ghi (register) và bộ nhớ ROM, RAM
 - *Bộ nhớ phụ* (secondary storage): Hệ thống lưu trữ thông tin *bền vững* (nonvolatile storage)
 - ▶ Đĩa từ (magnetic disk): đĩa mềm, đĩa cứng, băng từ
 - ▶ Đĩa quang (optical disk): CD-ROM, DVD-ROM
 - ▶ Flash ROM: USB disk



Metaphor (ẩn dụ) cho ‘phân cấp’ hệ thống lưu trữ

■ Đọc sách ở thư viện

Dung
lượng
tăng

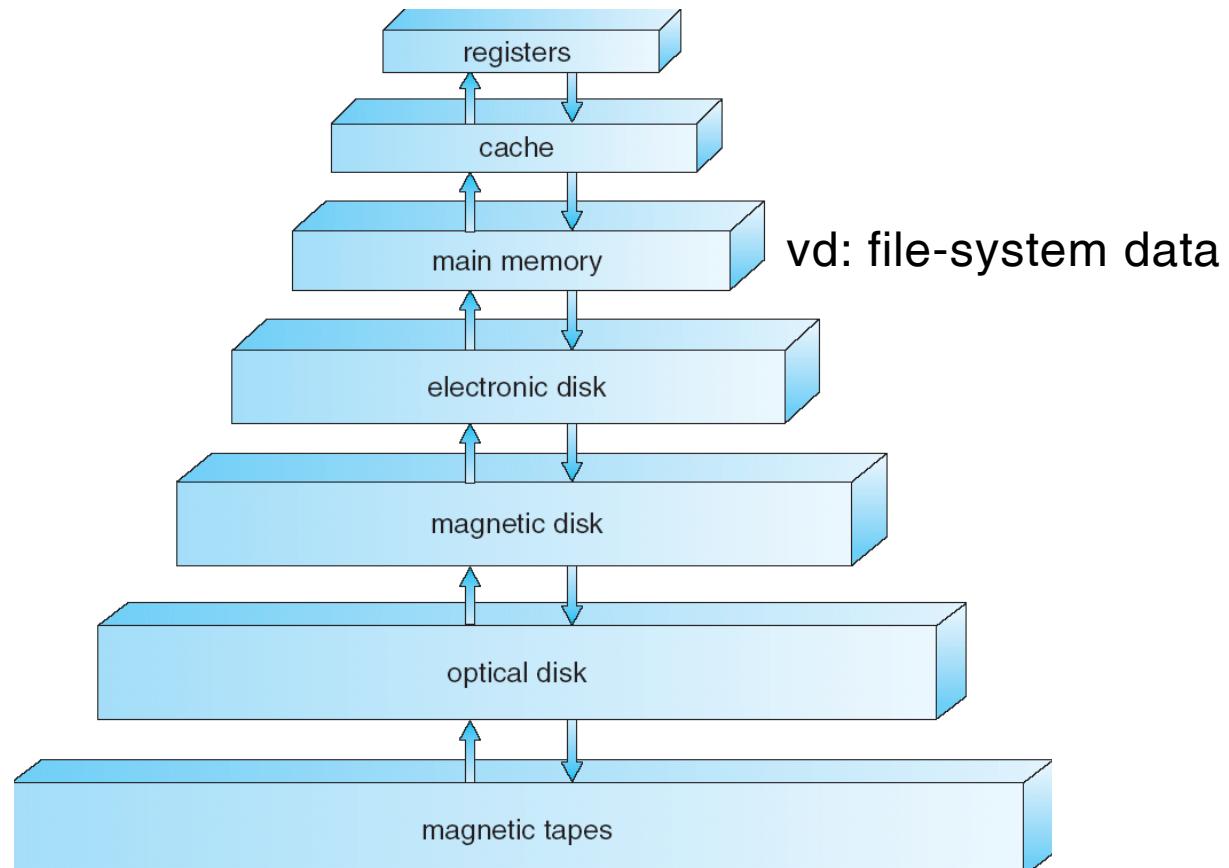


Phân cấp hệ thống lưu trữ

Tốc độ nhanh



Giá thành mỗi byte rẻ
Dung lượng lớn

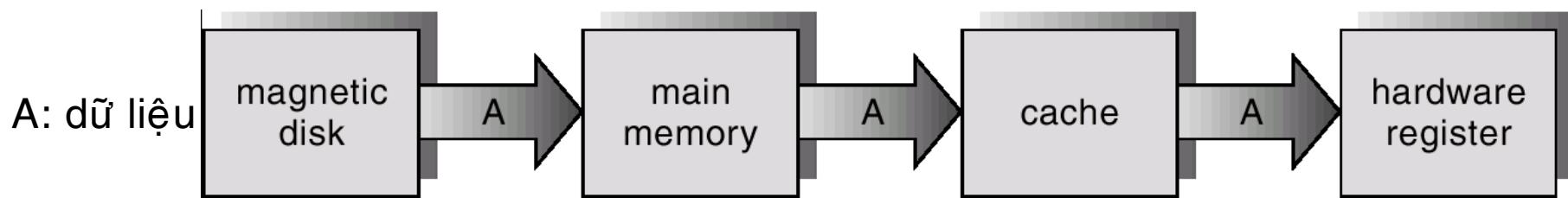


2.4 Fig 2.6



Kỹ thuật caching

- **Caching:** nạp trước dữ liệu vào thiết bị lưu trữ có tốc độ truy cập cao hơn



- Tại sao dùng cache?
 - Cải thiện tốc độ truy cập dữ liệu



Kỹ thuật caching

■ Vì sao caching “works”?

- Nguyên lý cục bộ (locality principle)

■ Vấn đề:

- Dữ liệu lớn, còn kích thước cache nhỏ → phải quản lý cache: thay nội dung nào của cache khi nó đầy?
- Một dữ liệu có thể được lưu trữ nhiều nơi → cần bảo đảm tính nhất quán dữ liệu: *cache coherency problem*

Bảo vệ tài nguyên chia sẻ



Dual mode (1/3)

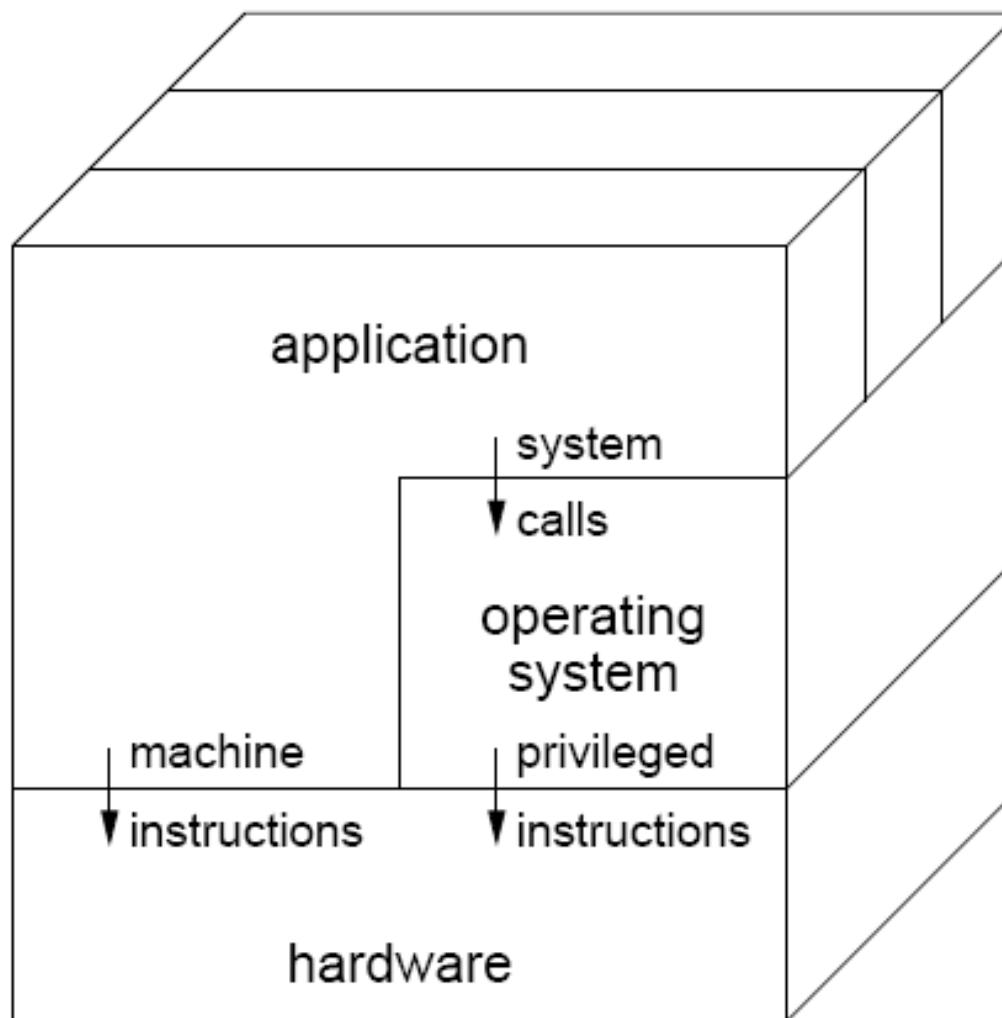
- Mục tiêu: bảo vệ hệ điều hành và chương trình ứng dụng
- Giải pháp

Kỹ thuật *dual mode*: CPU thực thi dưới một trong hai chế độ (mode)

- *User mode* – chỉ thực thi được các lệnh nonprivileged
 - ▶ Application chạy trong user mode
- *Kernel mode* (còn gọi là *supervisor mode*, *system mode*, *monitor mode*) – thực thi được tất cả các lệnh (privileged và nonprivileged) của CPU
 - ▶ Kernel chạy trong kernel mode



Dual mode (2/3)

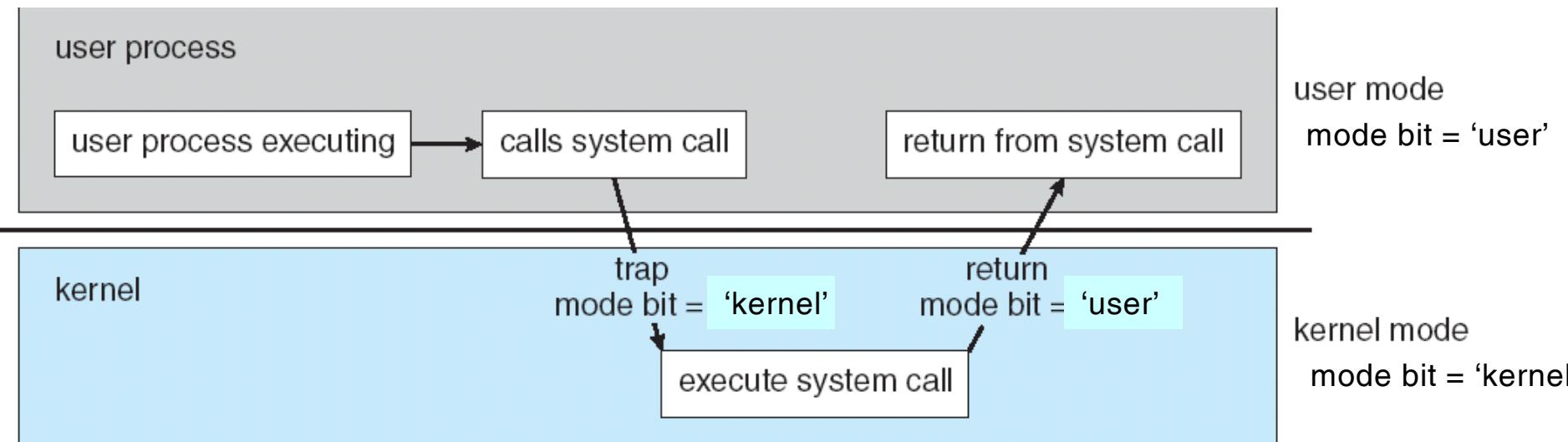


Hình của Dror G. Feitelson



Dual mode (3/3)

- Phần cứng có thêm *mode bit* để kiểm soát mode hiện hành:
 - mode bit = ‘kernel’ (= 0) -- kernel mode
 - mode bit = ‘user’ (= 1) -- user mode
 - Khi CPU bị ngắt (do thiết bị ngoại vi, trap,...), CPU sẽ chuyển sang kernel mode và thực thi interrupt service routine tương ứng.



Dòng thực thi và thay đổi chế độ CPU khi gọi system call



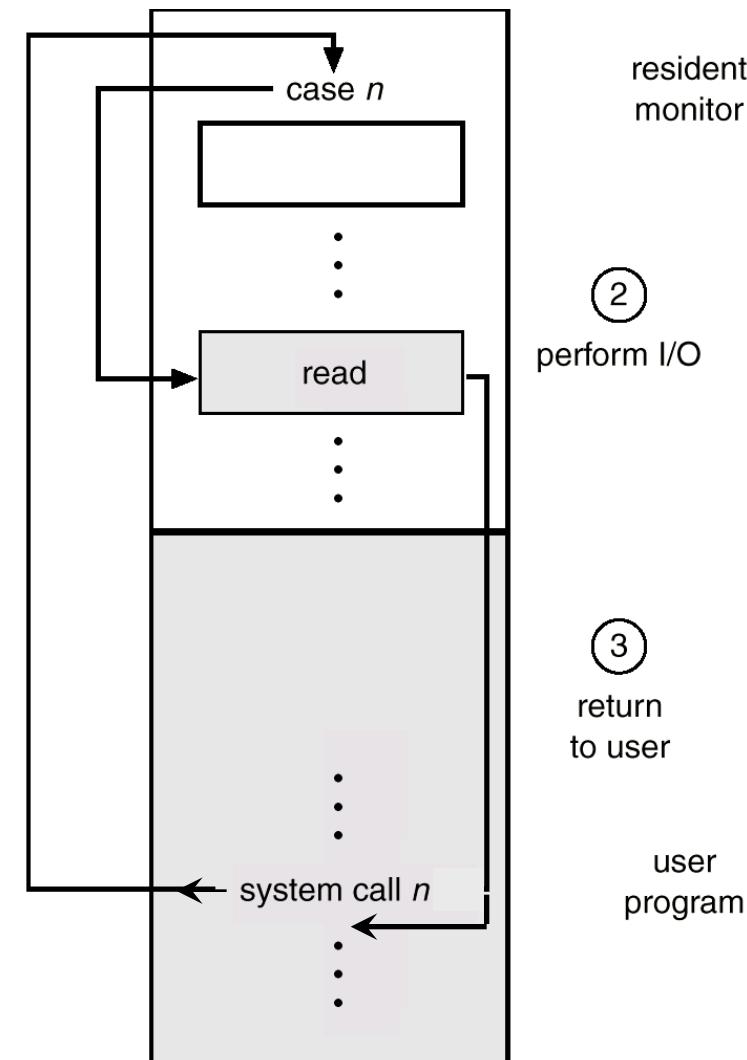
Bảo vệ I/O

- Ngăn user sử dụng I/O không hợp lệ
- Giải pháp: lệnh I/O đều là *privileged instruction*

- User mode program không thực thi được lệnh I/O (\rightarrow trap), phải thông qua lời gọi *system call*

System call

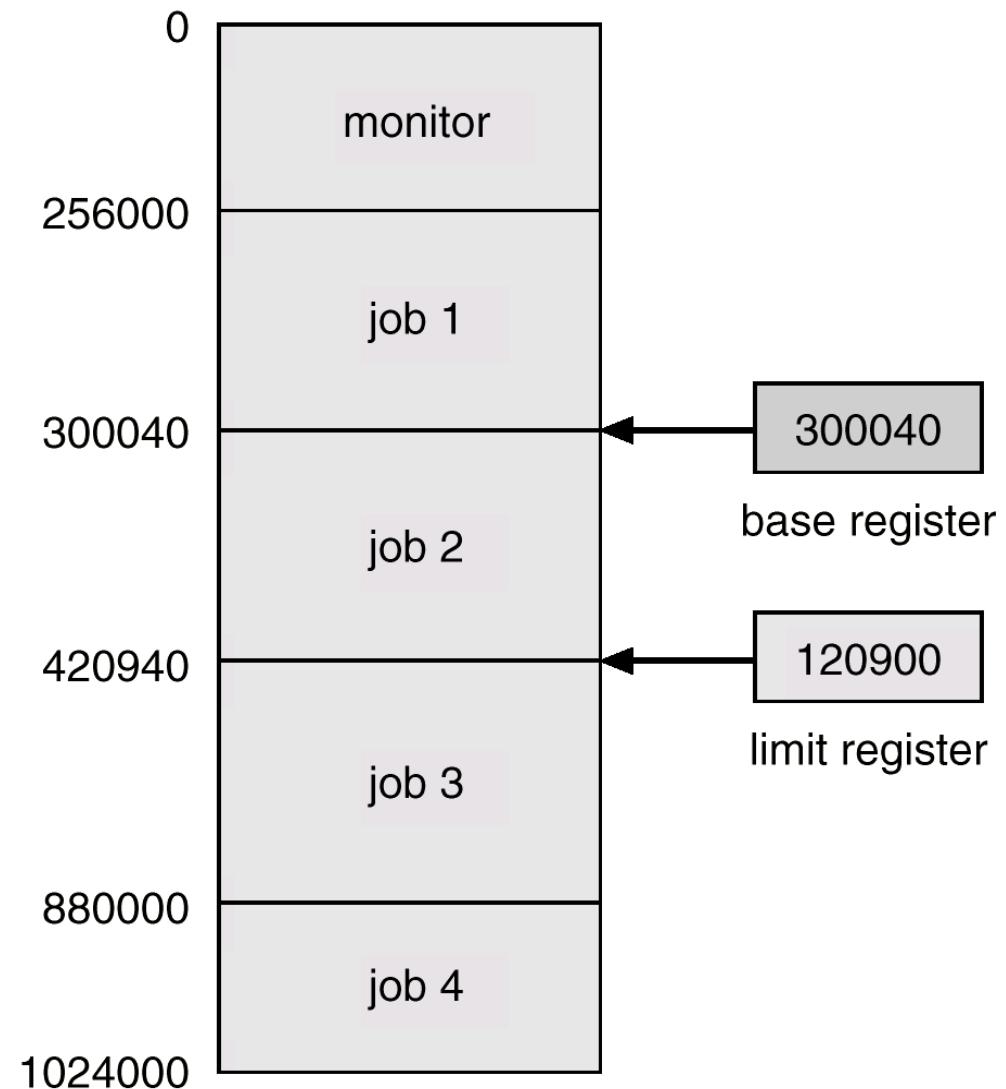
- Là phương thức duy nhất để process yêu cầu các dịch vụ của hệ điều hành
- System call sẽ gây ra ngắt mềm (*trap*), quyền điều khiển được chuyển đến trình phục vụ ngắt tương ứng, đồng thời thiết lập *mode = 'kernel'*
- Hệ điều hành kiểm tra tính hợp lệ, đúng đắn của các đối số, thực hiện yêu cầu rồi trả quyền điều khiển về lệnh kế tiếp ngay sau lời gọi system call, *mode = 'user'*



2.5.2 Fig 2.8

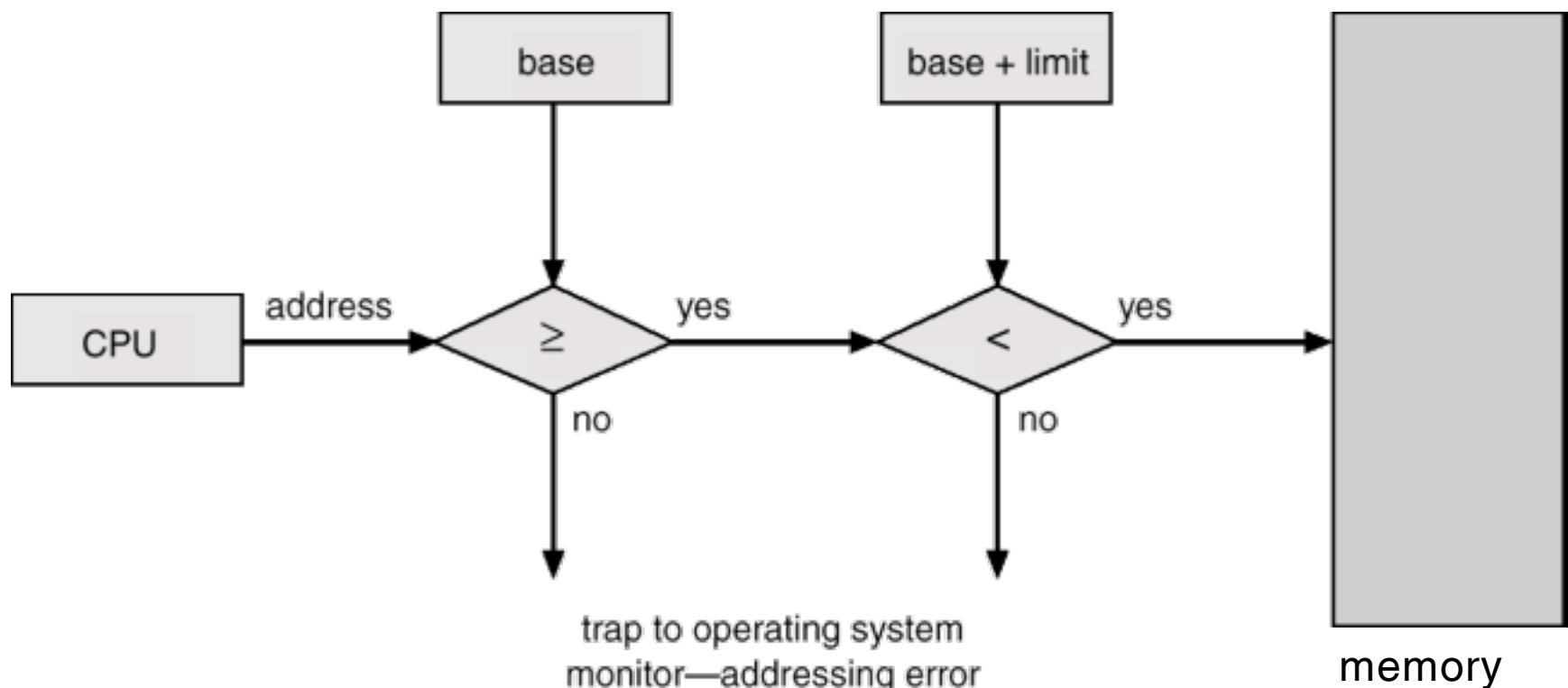
Bảo vệ bộ nhớ (1/2)

- Ví dụ: bảo vệ vùng nhớ cấp phát cho các process, hỗ trợ bởi phần cứng
 - Base register
 - Limit register



Bảo vệ bộ nhớ (2/2)

- Truy cập bộ nhớ ngoài vùng xác định bởi thanh ghi base và thanh ghi limit sẽ sinh ra **trap**
- Lệnh nạp giá trị cho các thanh ghi base và thanh ghi limit đều là privileged instruction





Bảo vệ CPU

■ Bảo vệ CPU

- Bảo đảm OS duy trì được quyền điều khiển CPU
 - ▶ Làm gì trường hợp CPU thực thi trong vòng lặp vô hạn?

Cơ chế thực hiện là dùng timer để kích khởi các ngắt quãng định kỳ

- Bộ đếm timer sẽ giảm dần sau mỗi xung clock
- Khi bộ đếm timer bằng 0 thì ngắt timer được kích hoạt → hệ điều hành sẽ nắm lại quyền điều khiển
- Lệnh nạp giá trị bộ đếm timer là privileged instruction



Timer

- Có thể sử dụng timer để thực hiện time-sharing
 - Thiết lập timer gây ngắt định kỳ N ms (N : *time slice, quantum time*) và định thời CPU sau mỗi lần ngắt
- Có thể dùng timer để tính thời gian trôi qua (elapse time)



Các thành phần của hệ điều hành (1/7)

■ *Quản lý quá trình*

- Quá trình vs chương trình
- Một quá trình cần các tài nguyên của hệ thống như CPU, bộ nhớ, file, thiết bị I/O,... để hoàn thành công việc
- Các nhiệm vụ
 - ▶ Tạo và hủy quá trình
 - ▶ Tạm ngưng / tiếp tục thực thi (suspend / resume) quá trình
 - ▶ Cung cấp các cơ chế
 - đồng bộ hoạt động các quá trình
 - giao tiếp giữa các quá trình
 - xử lý deadlock



Các thành phần của hệ điều hành (2/7)

■ Quản lý bộ nhớ chính

- Cần thiết vì nhiều quá trình chạy đồng thời trong hệ thống phải chia sẻ bộ nhớ
- Tùy thuộc kiến trúc máy tính
- Để có hiệu suất cao, hệ điều hành cần dùng giải thuật quản lý bộ nhớ thích hợp
- Các nhiệm vụ
 - ▶ Theo dõi, quản lý các vùng nhớ trống và đã cấp phát
 - ▶ Quyết định sẽ nạp chương trình nào khi có vùng nhớ trống
 - ▶ Cấp phát và thu hồi các vùng nhớ



Các thành phần của hệ điều hành (3/7)

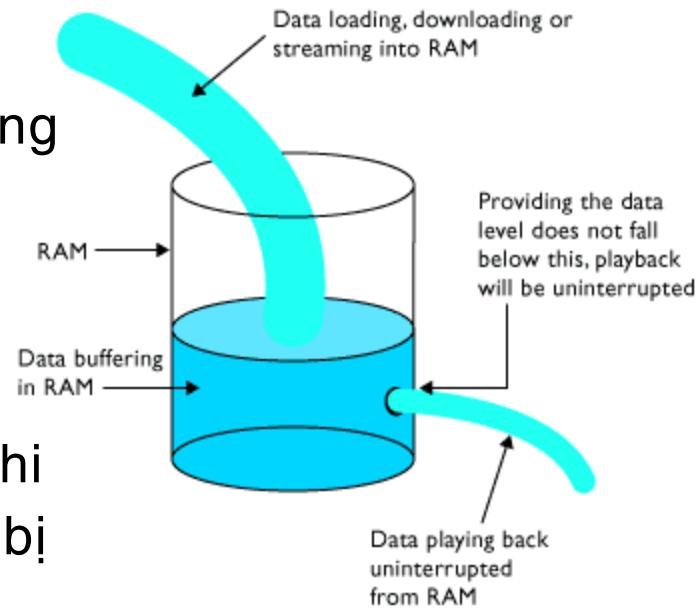
- Cần thiết cho việc lưu trữ dữ liệu bền vững (persistent):
Tiện ích *file* và *quản lý file*
 - Hệ thống file (file system)
 - ▶ *File*
 - ▶ *Thư mục*
 - Các dịch vụ mà thành phần cung cấp
 - ▶ Tạo và xoá file/thư mục
 - ▶ Các tác vụ xử lý file/thư mục (rename, copy, move, new,...)
 - ▶ “Ánh xạ” file/thư mục vào thiết bị lưu trữ thứ cấp tương ứng
 - ▶ Sao lưu và phục hồi dữ liệu



Các thành phần của hệ điều hành (4/7)

■ Quản lý hệ thống I/O

- Che dấu các đặc trưng riêng biệt của từng thiết bị I/O đối với user
- Có chức năng
 - ▶ Buffering, caching, spooling
 - Buffer: vùng nhớ để lưu dữ liệu khi chúng được truyền giữa hai thiết bị hay giữa thiết bị và ứng dụng
 - ▶ Cung cấp giao diện chung đến các trình điều khiển thiết bị (device-driver interface)
 - ▶ *Trình điều khiển thiết bị* cho mỗi chủng loại thiết bị phần cứng khác nhau





Các thành phần của hệ điều hành (5/7)

■ Quản lý hệ thống lưu trữ thứ cấp

- Bộ nhớ chính: kích thước nhỏ, là môi trường chứa tin không bền vững → cần hệ thống lưu trữ thứ cấp để lưu trữ bền vững các dữ liệu, chương trình
- Phương tiện lưu trữ thông dụng là đĩa từ, đĩa quang
- Nhiệm vụ
 - ▶ Quản lý vùng trống
 - ▶ Cấp phát không gian lưu trữ (storage allocation)
 - ▶ Định thời đĩa (disk scheduling)



Các thành phần của hệ điều hành (6/7)

- *Hệ thống bảo vệ* (protection system) – cần thiết khi hệ thống cho phép nhiều user hay nhiều quá trình
 - Kiểm soát quá trình người dùng đăng nhập/xuất (login, logout) và sử dụng hệ thống
 - Kiểm soát việc truy cập các tài nguyên trong hệ thống
 - ▶ Bảo đảm chỉ những người dùng/quá trình *đủ quyền hạn* mới được phép sử dụng các tài nguyên tương ứng
 - Các nhiệm vụ
 - ▶ Cung cấp cơ chế kiểm soát đăng nhập/xuất
 - ▶ Phân định được sự truy cập tài nguyên *hợp lệ* và *bất hợp lệ* (authorized / unauthorized)
 - ▶ Phương tiện thi hành các chính sách (enforcement of policies)
Chính sách: cần bảo vệ dữ liệu của ai đối với ai



Các thành phần của hệ điều hành (7/7)

- *Trình thông dịch lệnh (command line interpreter)*
 - Là giao diện chủ yếu giữa người dùng và OS
 - ▶ Ví dụ: shell, mouse-based window-and-menu
 - Khi user login
 - ▶ Hệ thống khởi tạo *command line interpreter* (shell) cho user, và nó chờ nhận lệnh từ người dùng, thực thi lệnh và trả kết quả về
 - Liên hệ chặt chẽ với các thành phần khác của hệ điều hành để thực thi các yêu cầu của người dùng



Các thành phần của hệ điều hành (7/7)

■ *Trình thông dịch lệnh* (tt)

- Các nhóm lệnh trình thông dịch lệnh để
 - ▶ Tạo, hủy, xem thông tin quá trình, hệ thống
 - ▶ Điều khiển truy cập I/O
 - ▶ Quản lý, truy cập hệ thống lưu trữ thứ cấp
 - ▶ Quản lý, sử dụng bộ nhớ
 - ▶ Truy cập hệ thống file
 - ▶ ...



Các dịch vụ hệ điều hành cung cấp (1/2)

- *Một số dịch vụ chủ yếu* mà người dùng hay chương trình cần
 - Thực thi chương trình
 - Thực hiện các tác vụ I/O do yêu cầu của chương trình
 - Các tác vụ lên file
 - ▶ Đọc/ghi hay tạo/xóa file
 - Giao tiếp, trao đổi thông tin giữa các quá trình
 - ▶ Shared memory
 - ▶ Message passing
 - Phát hiện lỗi
 - ▶ Trên thiết bị I/O: dữ liệu hư, hết giấy,...
 - ▶ Chương trình ứng dụng: chia cho 0, truy cập đến địa chỉ bộ nhớ không được phép



Các dịch vụ hệ điều hành cung cấp (2/2)

■ *Các dịch vụ khác*

- Cấp phát tài nguyên
 - ▶ Tài nguyên: tape drives,...
 - ▶ OS có các routine tương ứng
- Kế toán (accounting)
 - ▶ Ví dụ để tính phí



Các chương trình hệ thống

- *Chương trình hệ thống* (phân biệt với application program) gồm
 - Quản lý file: như create, delete, rename, list
 - Thông tin trạng thái hệ thống: như time, dung lượng bộ nhớ trống
 - Soạn thảo văn bản: như **vi/vim** trên Unix/Linux
 - Hỗ trợ ngôn ngữ lập trình: như compiler, assembler, interpreter
 - Nạp, thực thi, giúp tìm lỗi chương trình: như loader, debugger
 - Giao tiếp: như email, talk, **web browser**
 - ...
- Người dùng cuối chủ yếu làm việc thông qua các system program (không sử dụng “trực tiếp” các system call).



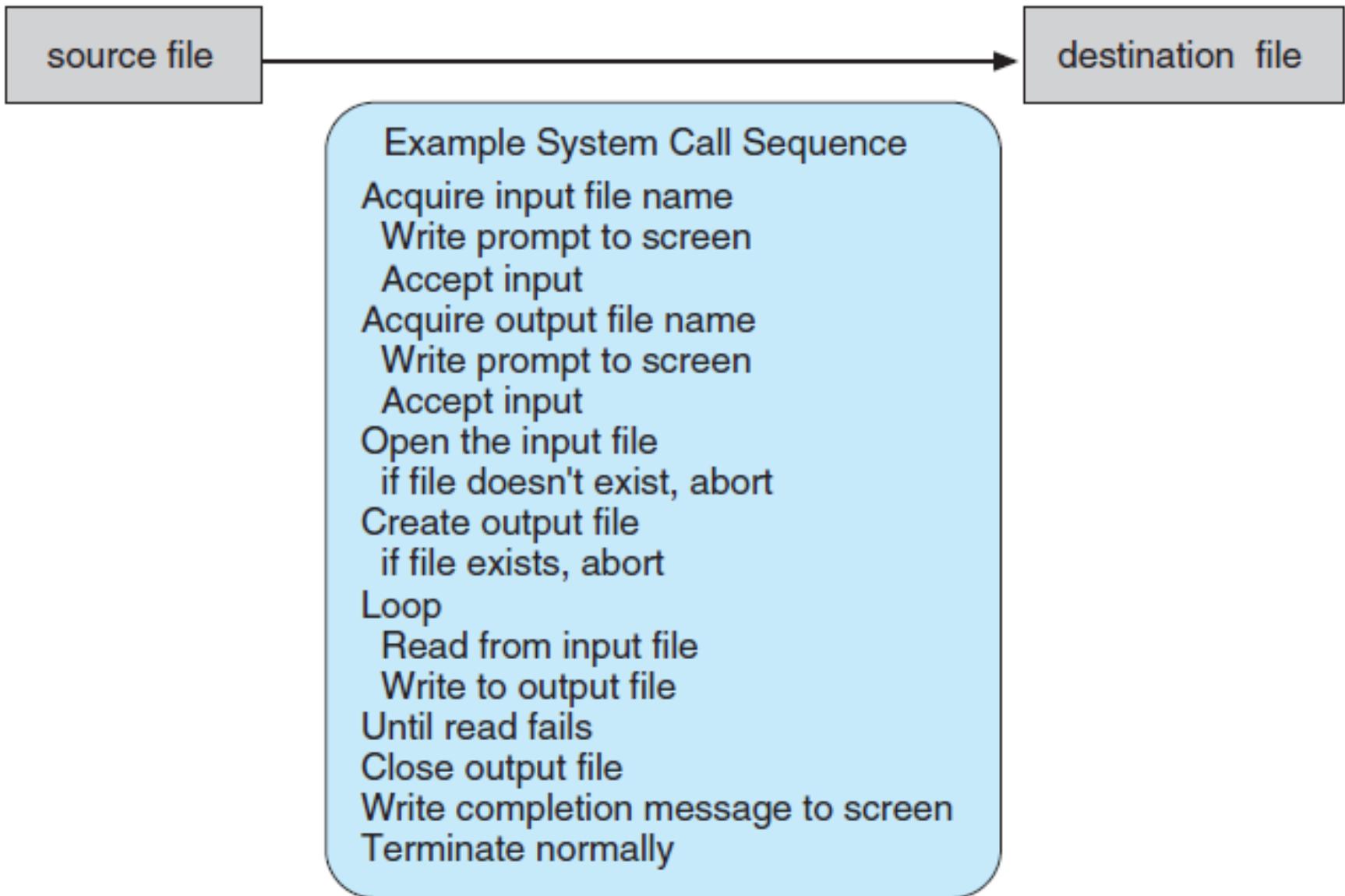
Giao diện giữa quá trình và hệ điều hành

■ *System call*

- Phương pháp hiện thực giao diện giữa quá trình và hệ điều hành
 - ▶ Vd: open, read, write file
- Thông thường ở dạng thư viện nhị phân (binary library)
- Trong các ngôn ngữ lập trình cấp cao, một số thư viện lập trình được xây dựng dựa trên các thư viện hệ thống (ví dụ Windows API, thư viện GNU C/C++ như glibc, glibc++,...)
- Ba cách *truyền tham số* trong system call
 - ▶ Truyền tham số qua **thanh ghi**
 - ▶ Truyền tham số thông qua một **vùng nhớ**, địa chỉ của vùng nhớ được gửi đến hệ điều hành qua thanh ghi
 - ▶ Truyền tham số qua **stack**



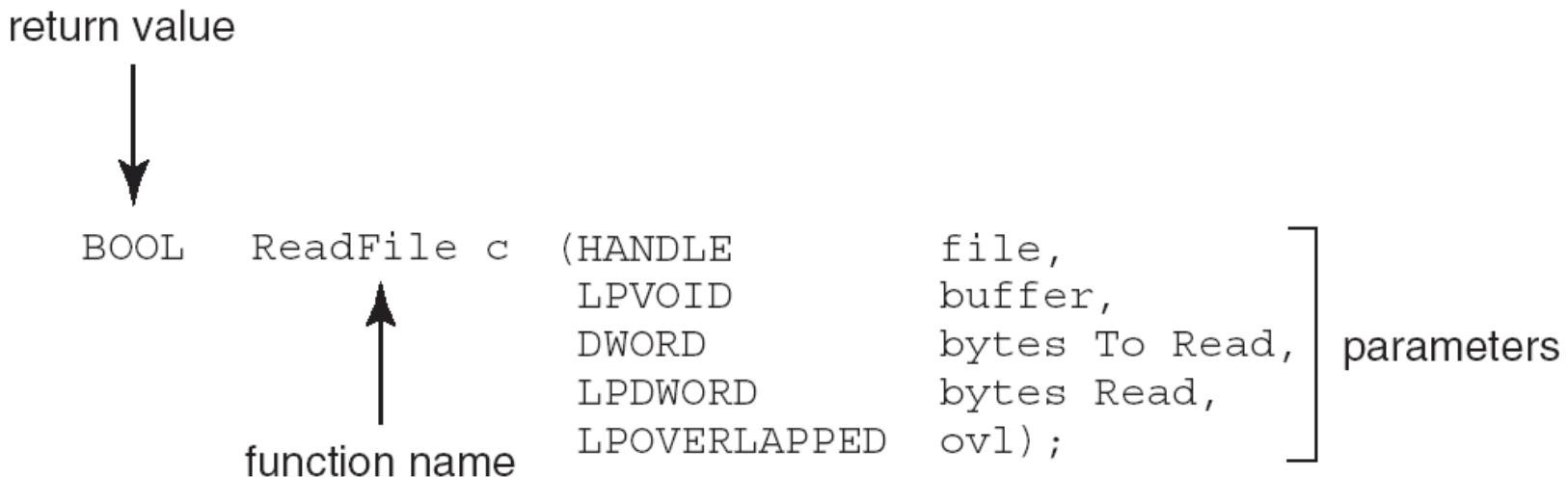
Ví dụ về system call





Standard API

■ Ví dụ ReadFile()



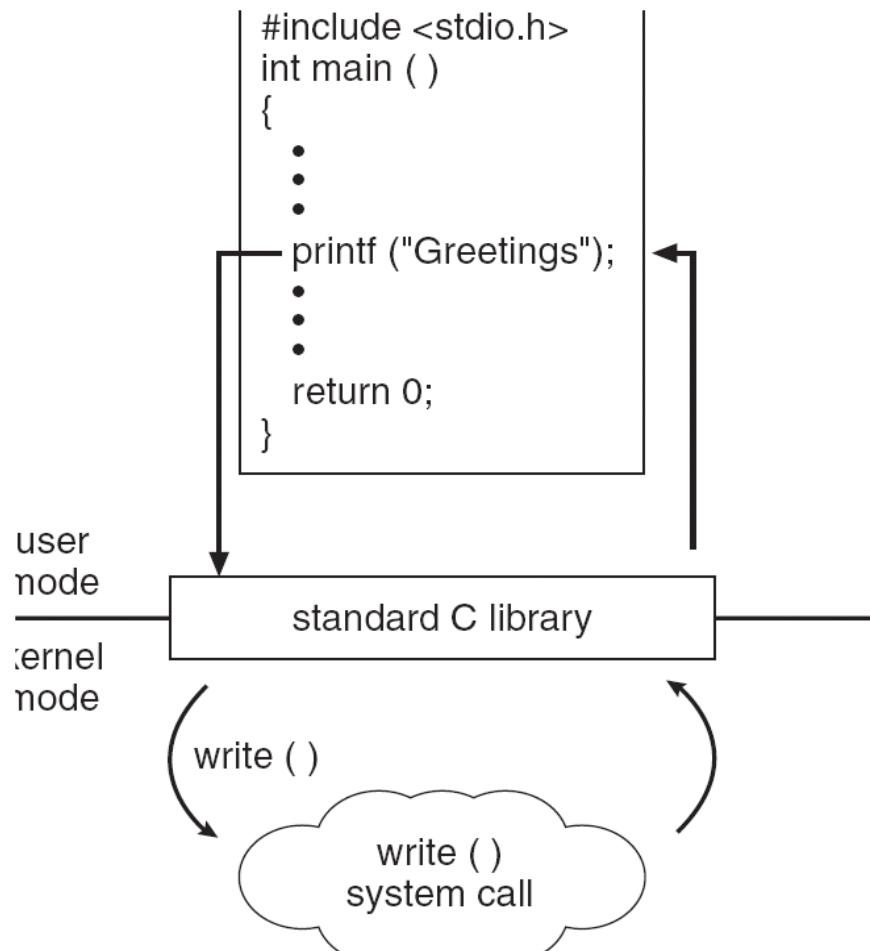
■ Mô tả các tham số truyền đến ReadFile()

- `HANDLE file`—the file to be read
- `LPVOID buffer`—a buffer where the data will be read into and written from
- `DWORD bytesToRead`—the number of bytes to be read into the buffer
- `LPDWORD bytesRead`—the number of bytes read during the last read
- `LPOVERLAPPED ovl`—indicates if overlapped I/O is being used



Standard C Library Example

- Chương trình C gọi printf(), và printf() sẽ gọi đến lời gọi hệ thống write()





Một số lời gọi của hệ Windows and Unix

	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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()



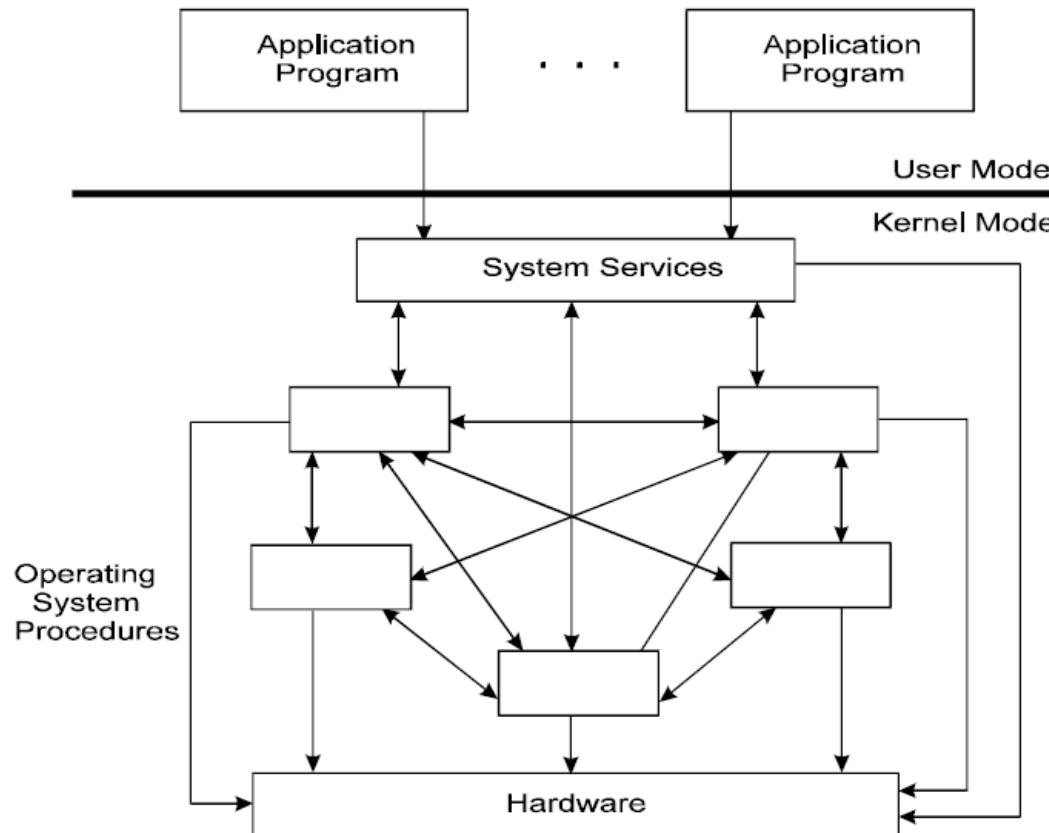
Các chương trình hệ thống

- *Chương trình hệ thống* (phân biệt với application program) gồm
 - Quản lý file: như create, delete, rename, list
 - Thông tin trạng thái hệ thống: như time, dung lượng bộ nhớ trống
 - Soạn thảo văn bản: như **vi/vim** trên Unix/Linux
 - Hỗ trợ ngôn ngữ lập trình: như compiler, assembler, interpreter
 - Nạp, thực thi, giúp tìm lỗi chương trình: như loader, debugger
 - Giao tiếp: như email, talk, **web browser**
 - ...
- Người dùng cuối chủ yếu làm việc thông qua các system program (không sử dụng “trực tiếp” các system call).

Cấu trúc hệ điều hành (1/7)

■ Hệ thống **đơn khối** (**monolithic**)

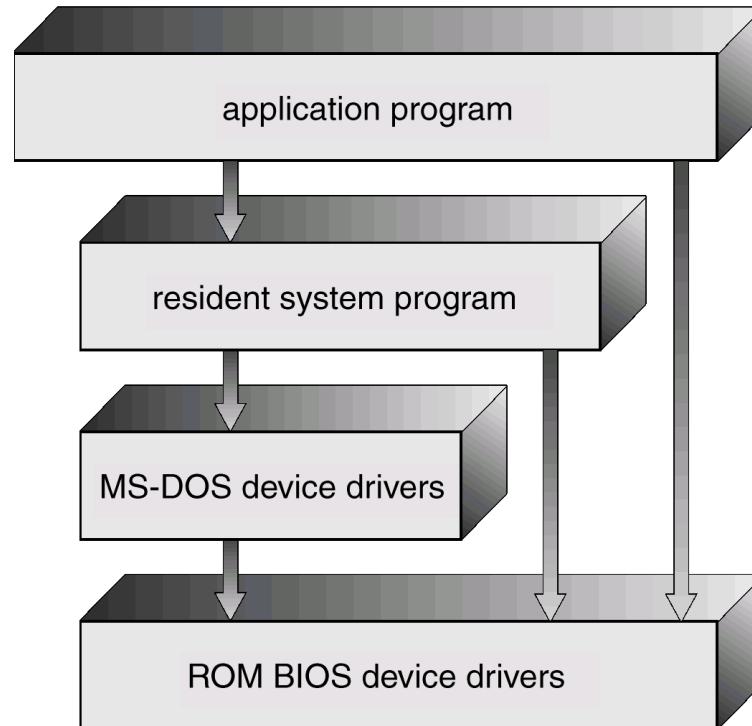
- Các thành phần được tổ chức tùy tiện, các module có thể tùy tiện gọi module khác



Cấu trúc hệ điều hành (2/7)

■ Hệ thống đơn khối – ví dụ

- MS-DOS: được thiết kế dưới điều kiện giới hạn về dung lượng bộ nhớ – Intel 8088, 1 MB bộ nhớ, không dual mode.
 - ▶ Nhìn lại, có thể *phân lớp*:





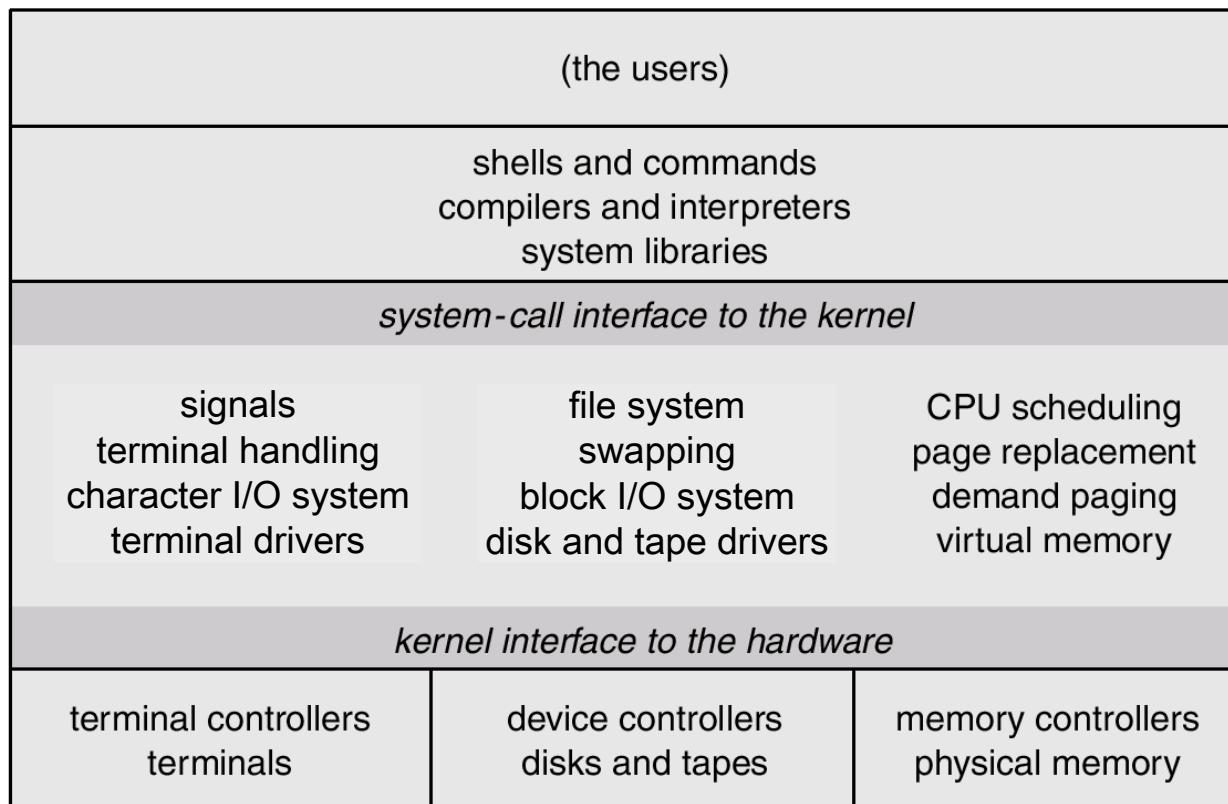
Cấu trúc hệ điều hành (3/7)

■ Hệ thống đơn khối – ví dụ

- UNIX: gồm hai phần

- ▶ các system program và kernel (file system, CPU scheduling, memory management, và một số chức năng khác)

Nhìn lại, có thể
phân lớp:



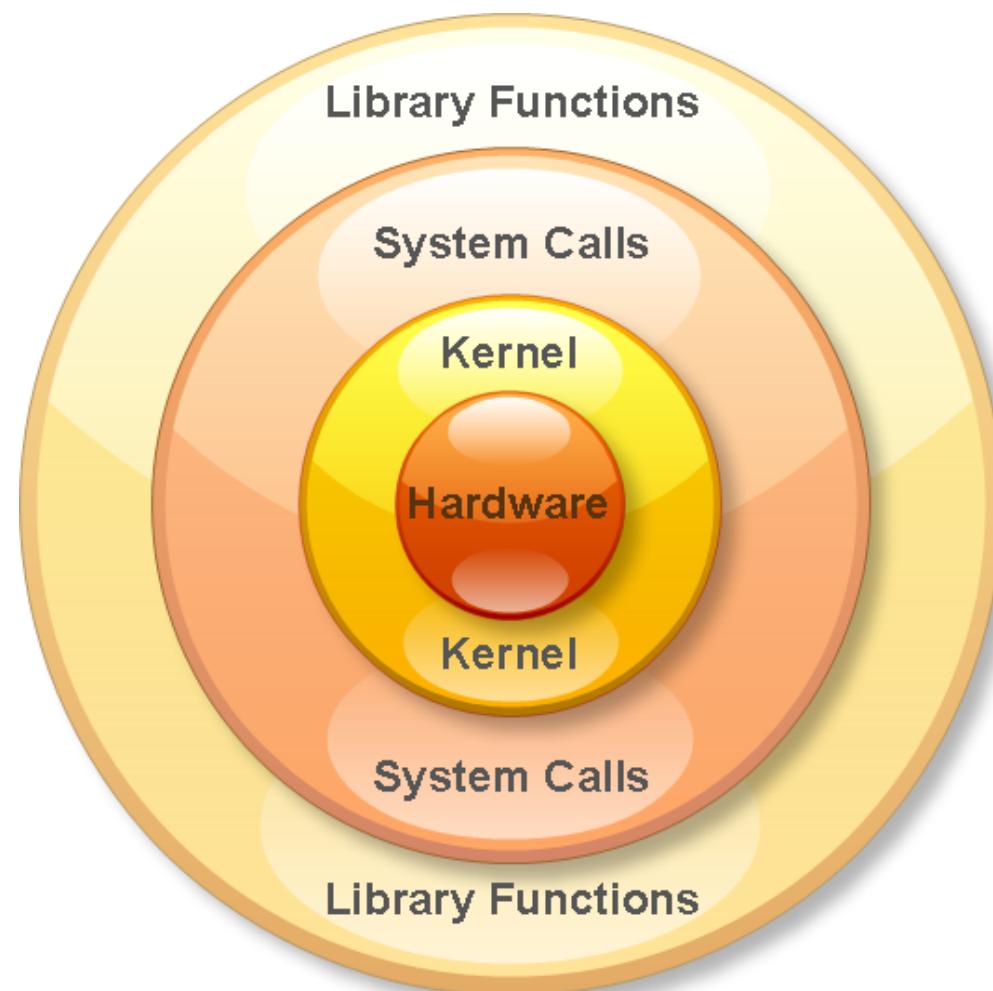


■ Trong hệ điều hành được phân lớp

(**layered** operating system) mỗi lớp gồm có cấu trúc dữ liệu và thủ tục chỉ được gọi bởi các lớp ở mức cao hơn.

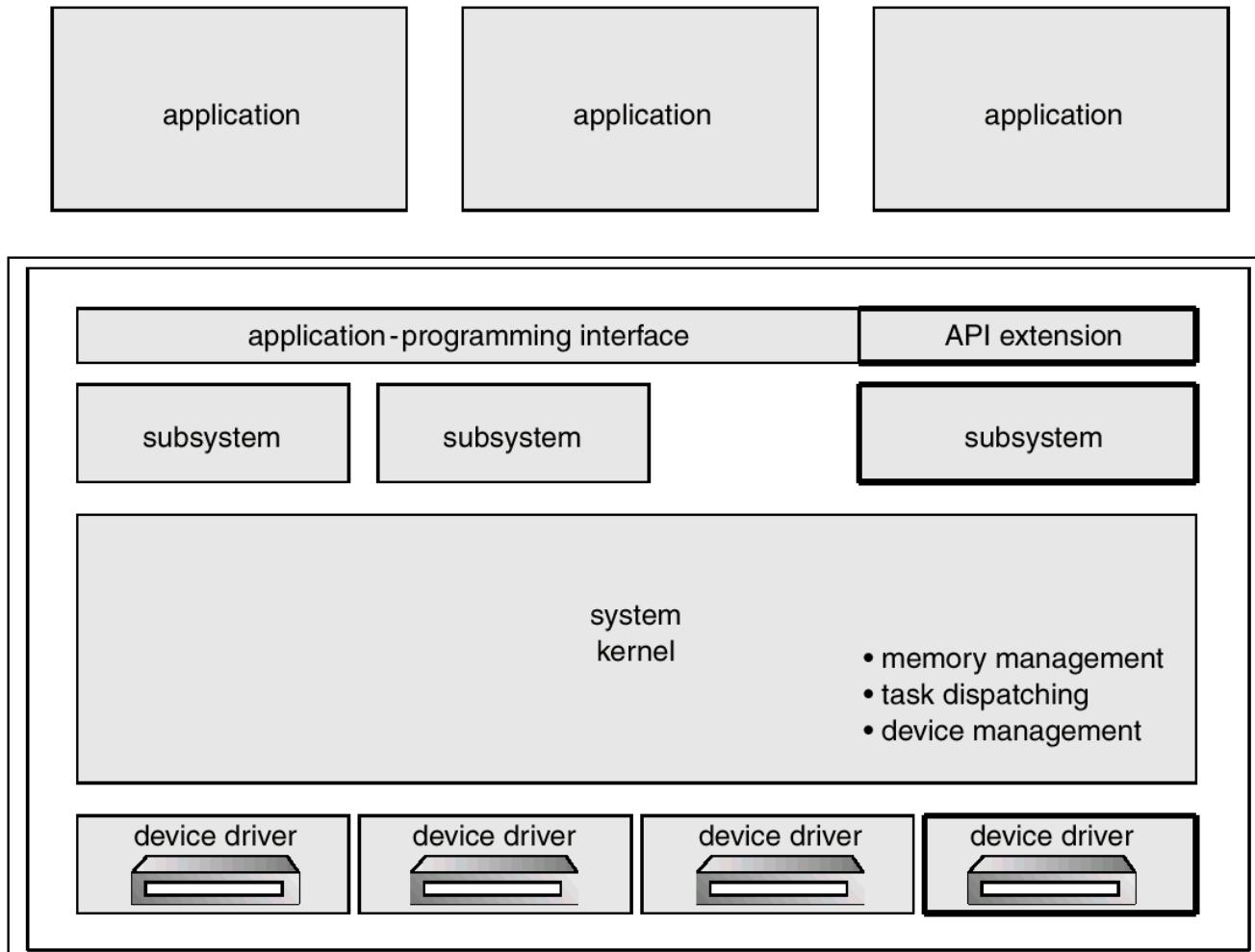
- Lớp thấp nhất là phần cứng
- Lớp cao nhất là giao diện người dùng
- Lớp dưới che giấu cấu trúc dữ liệu và cách hiện thực của thủ tục đối với lớp cao hơn

Cấu trúc hệ điều hành (4/7)



Cấu trúc hệ điều hành (5/7)

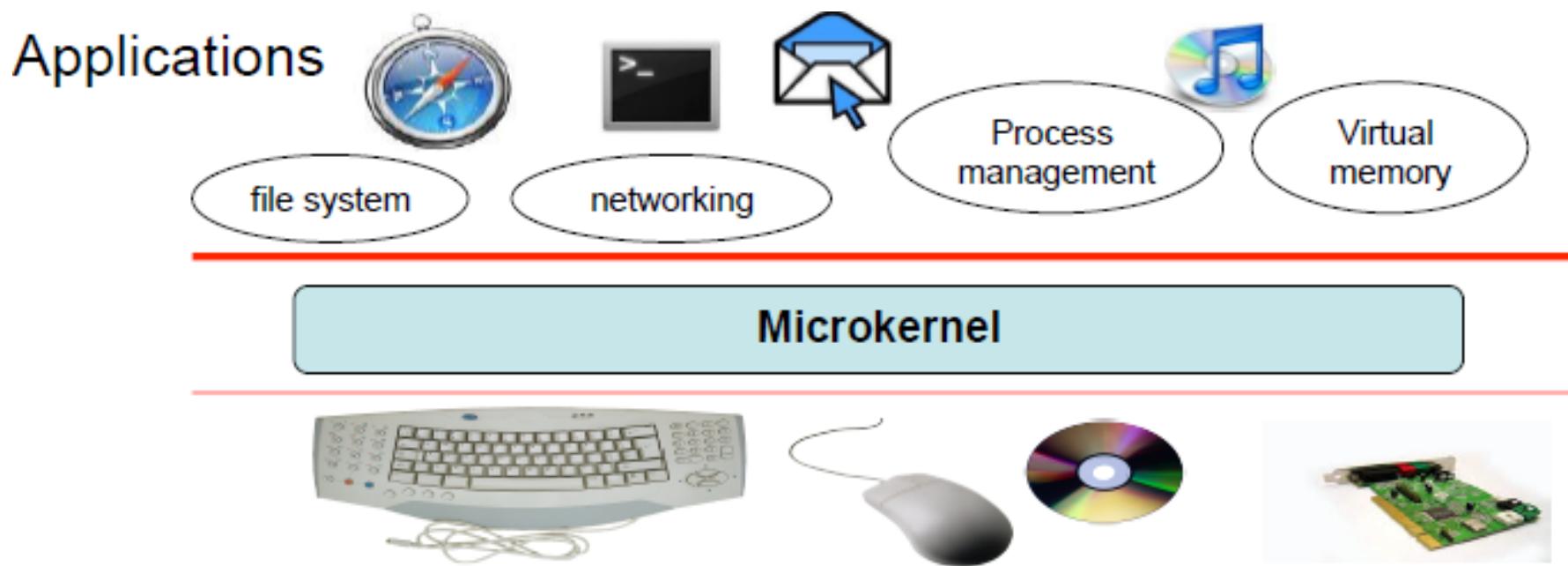
- **Layered approach** -- thiết kế một OS bằng cách phân chia module thành nhiều *lớp*. Vd: hệ điều hành OS/2 (IBM)



Cấu trúc hệ điều hành (6/7)

■ **Microkernel approach** (CMU Mach OS, 1980)

- Dời một số chức năng của OS từ kernel space sang user space (vd: file server)





Cấu trúc hệ điều hành (7/7)

■ **Microkernel approach** (tt)

- Thu gọn kernel → microkernel, chỉ gồm các chức năng cần thiết nhất như quản lý quá trình, bộ nhớ và cơ chế giao tiếp giữa các quá trình
 - ▶ Khối lượng code chạy trong kernel mode nhỏ hơn → kernel chạy ổn định hơn.
Vd: nếu networking service sụp đổ vì buffer overflow thì chỉ vùng nhớ của nó (thuộc user space) bị hư hại, còn kernel vẫn tiếp tục được
- Giao tiếp giữa các module dùng kỹ thuật *truyền thông điệp*



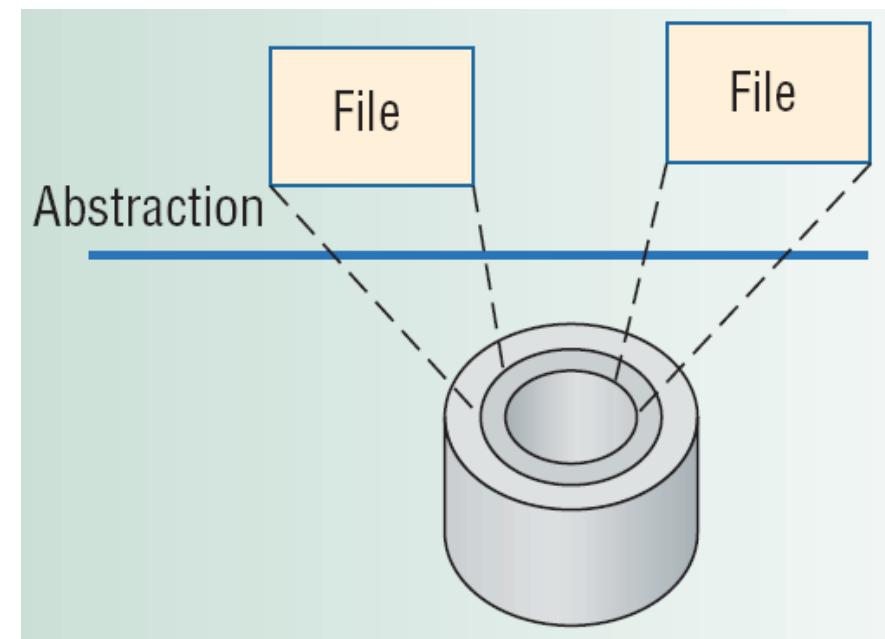
Hệ thống phân cấp

- Hệ thống được thiết kế theo layered approach là **hệ thống phân cấp** (hierarchical system)
 - Mỗi lớp của hệ thống phân cấp tạo nên một mức trừu tượng (level of abstraction)



Trùu tượng hóa

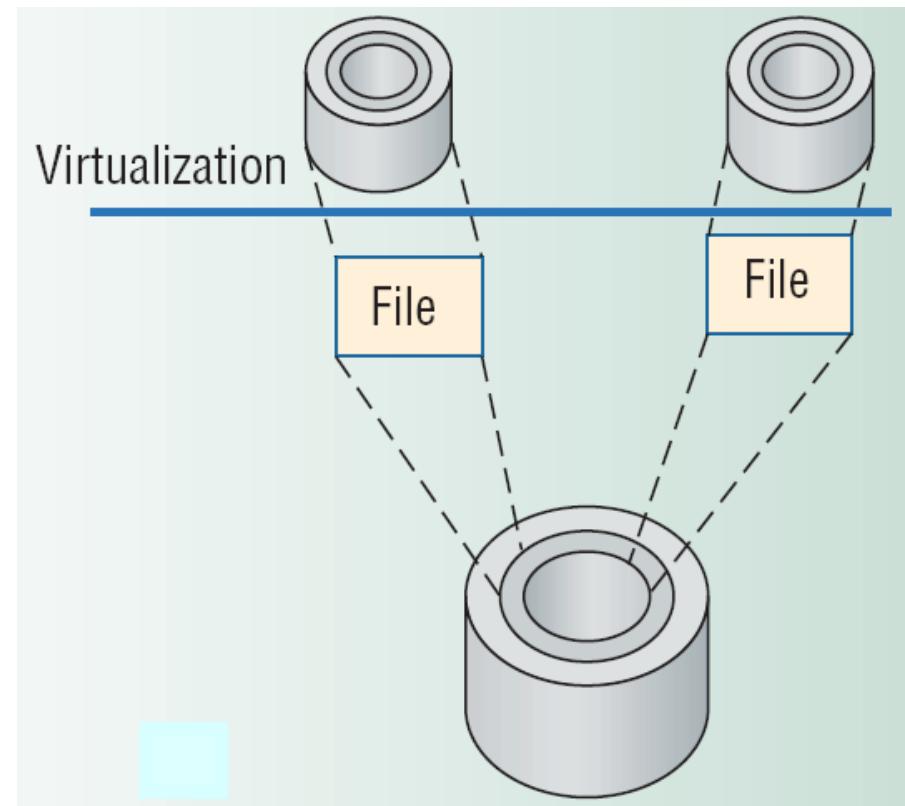
- Mục tiêu
 - Cung cấp một giao diện được đơn giản hóa đến tài nguyên bên dưới
 - Che giấu chi tiết hiện thực
- Ví dụ
 - Hệ điều hành trùu tượng hóa các chi tiết của đĩa cứng để đĩa cứng hiện ra đối với ứng dụng như là một tập các file





Ảo hóa

- **Ảo hóa** một hệ thống hay thành phần (processor, bộ nhớ, hay một thiết bị I/O) ở một mức trừu tượng
 - ánh xạ (hay chuyển đổi) giao diện và tài nguyên thấy được lên giao diện và tài nguyên của một hệ thống / tài nguyên bên dưới
- Ví dụ
 - Ảo hóa biến đổi một đĩa cứng lớn thành hai đĩa cứng nhỏ
 - ▶ Ghi vào một đĩa ảo được chuyển thành ghi vào file (và vì vậy thành ghi vào đĩa thực)

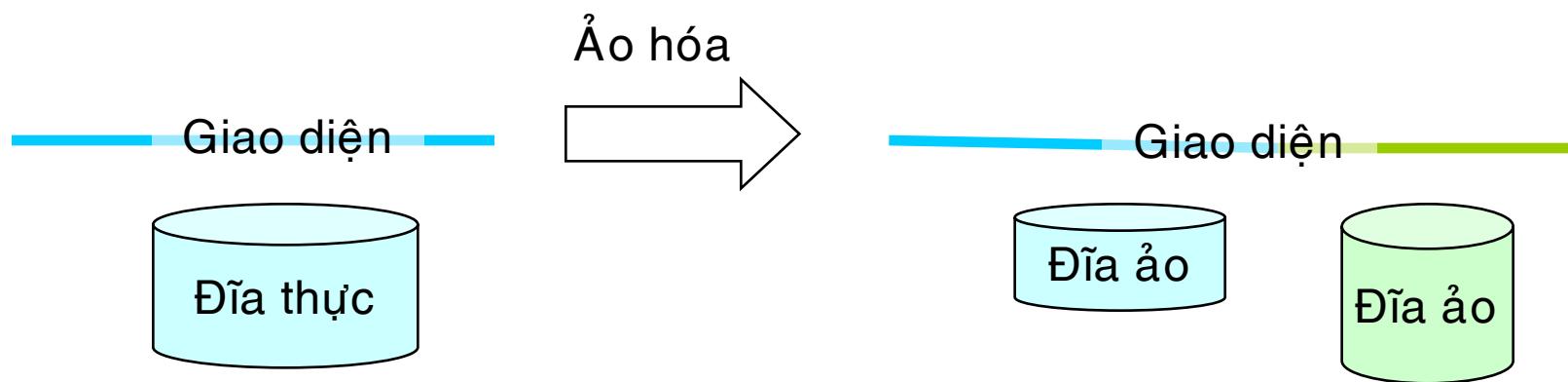




Ảo hóa

■ Ảo hóa cung cấp ở cùng một mức trừu tượng

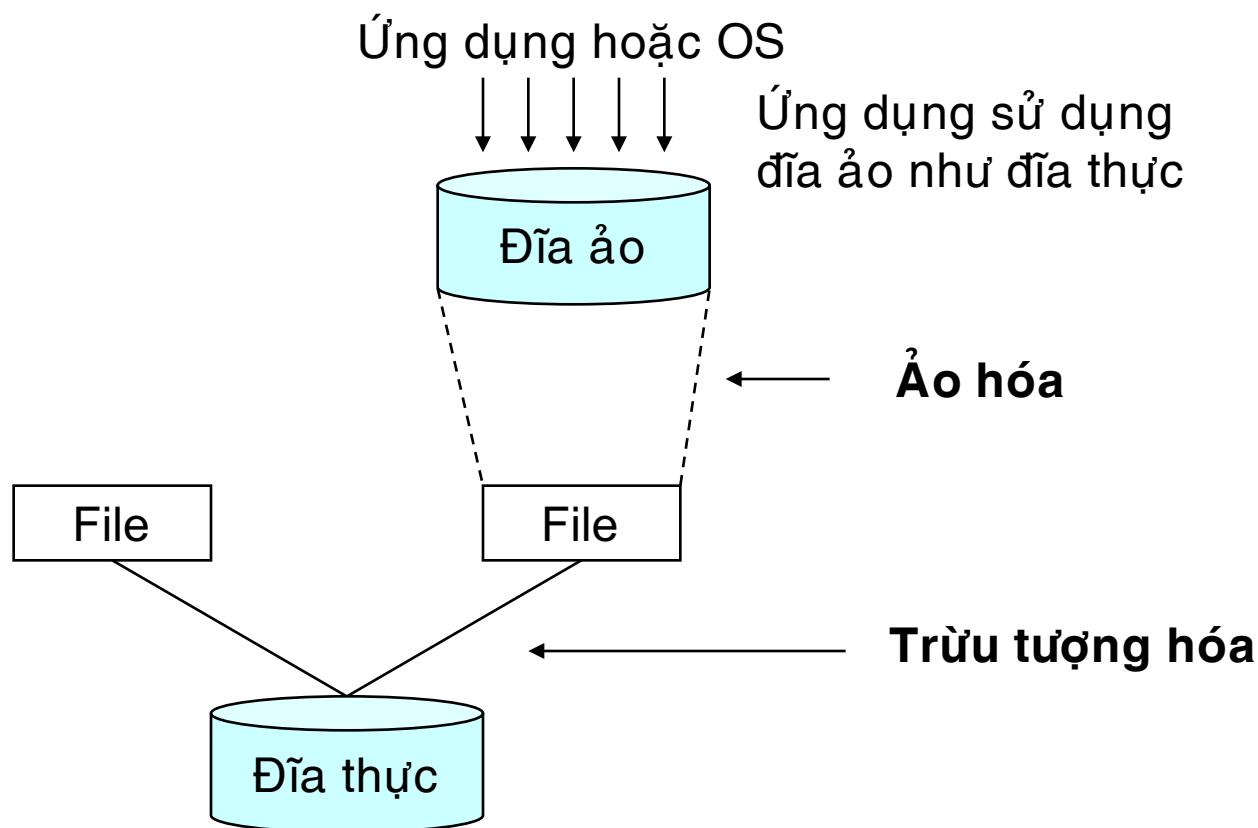
- Giao diện khác hoặc / và
- Tài nguyên khác



■ Ảo hóa **không nhất thiết** đơn giản hóa giao diện (như trong trường hợp trừu tượng hóa)

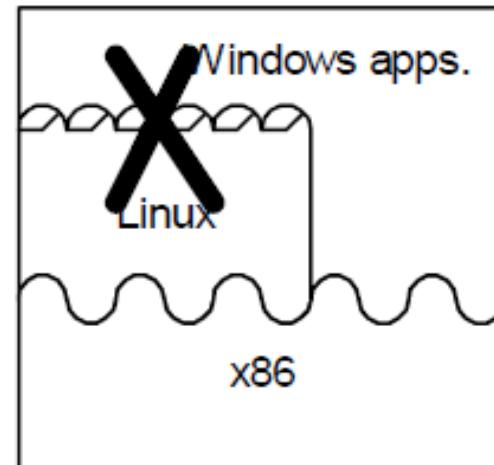
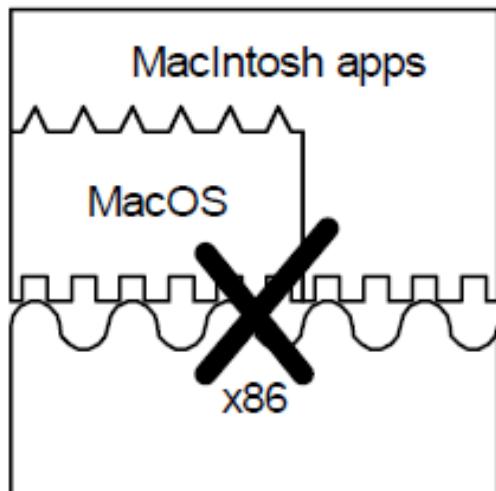
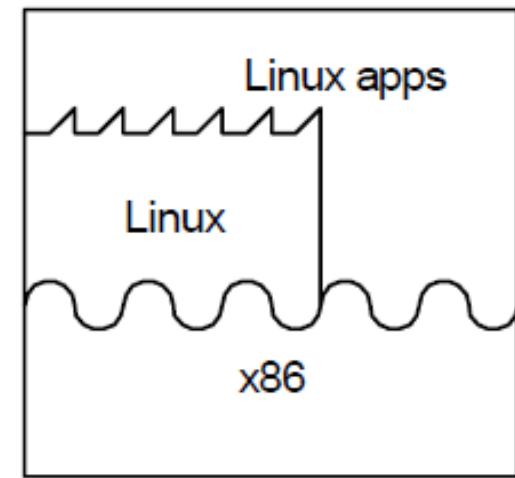
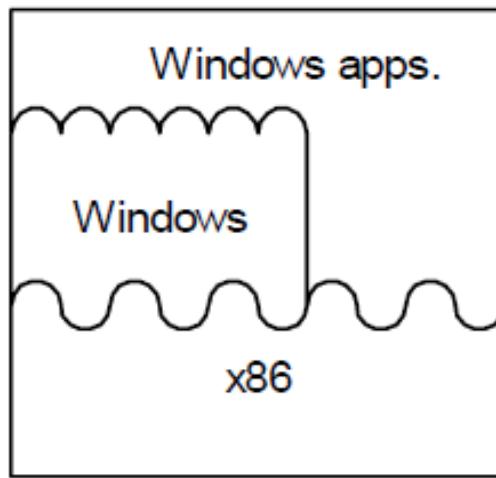
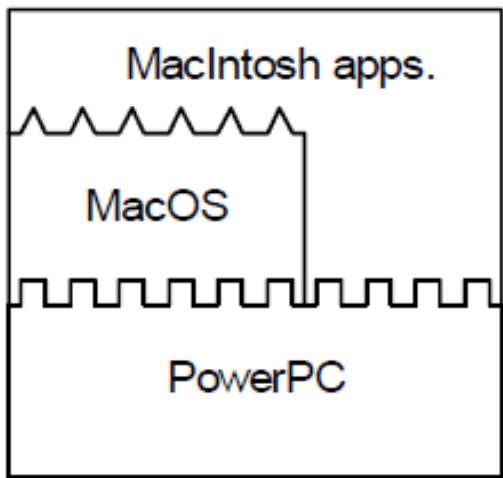


Trừu tượng hóa và ảo hóa





Sự giao tiếp giữa các thành phần khác nhau





Máy ảo

- Ảo hóa được áp dụng không những cho một hệ thống con như đĩa mà còn cho cả một hệ thống máy tính
- Hai loại máy ảo
 - Máy ảo mức quá trình (process virtual machine, process VM)
 - Máy ảo mức hệ thống (system virtual machine, system VM)
 - ▶ Trong trường hợp này, phần mềm ảo hóa còn được gọi là **Virtual Machine Monitor** (VMM):



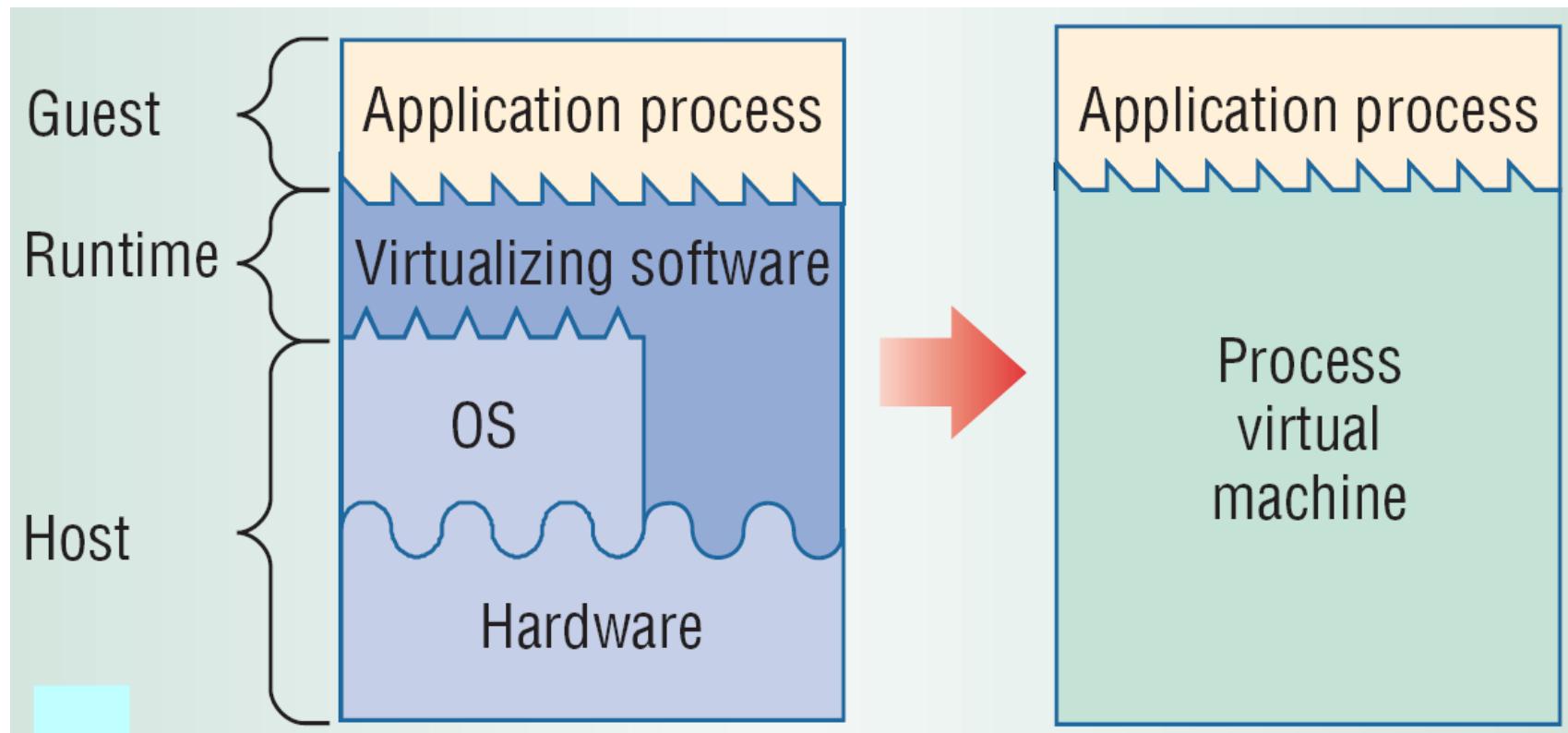
Lợi ích của máy ảo

- Tính an toàn: ứng dụng chạy trên máy ảo là an toàn hơn so với ứng dụng chạy trực tiếp trên máy phần cứng
- Độc lập phần cứng: phần cứng ảo không cần phải giống hệt với phần cứng vật lý bên dưới
- Tính linh hoạt trong trường hợp máy ảo mức hệ thống
 - Hỗ trợ cùng lúc nhiều hệ điều hành trên cùng một nền phần cứng
 - Giúp dễ dàng thử nghiệm hệ điều hành mới



Máy ảo mức quá trình

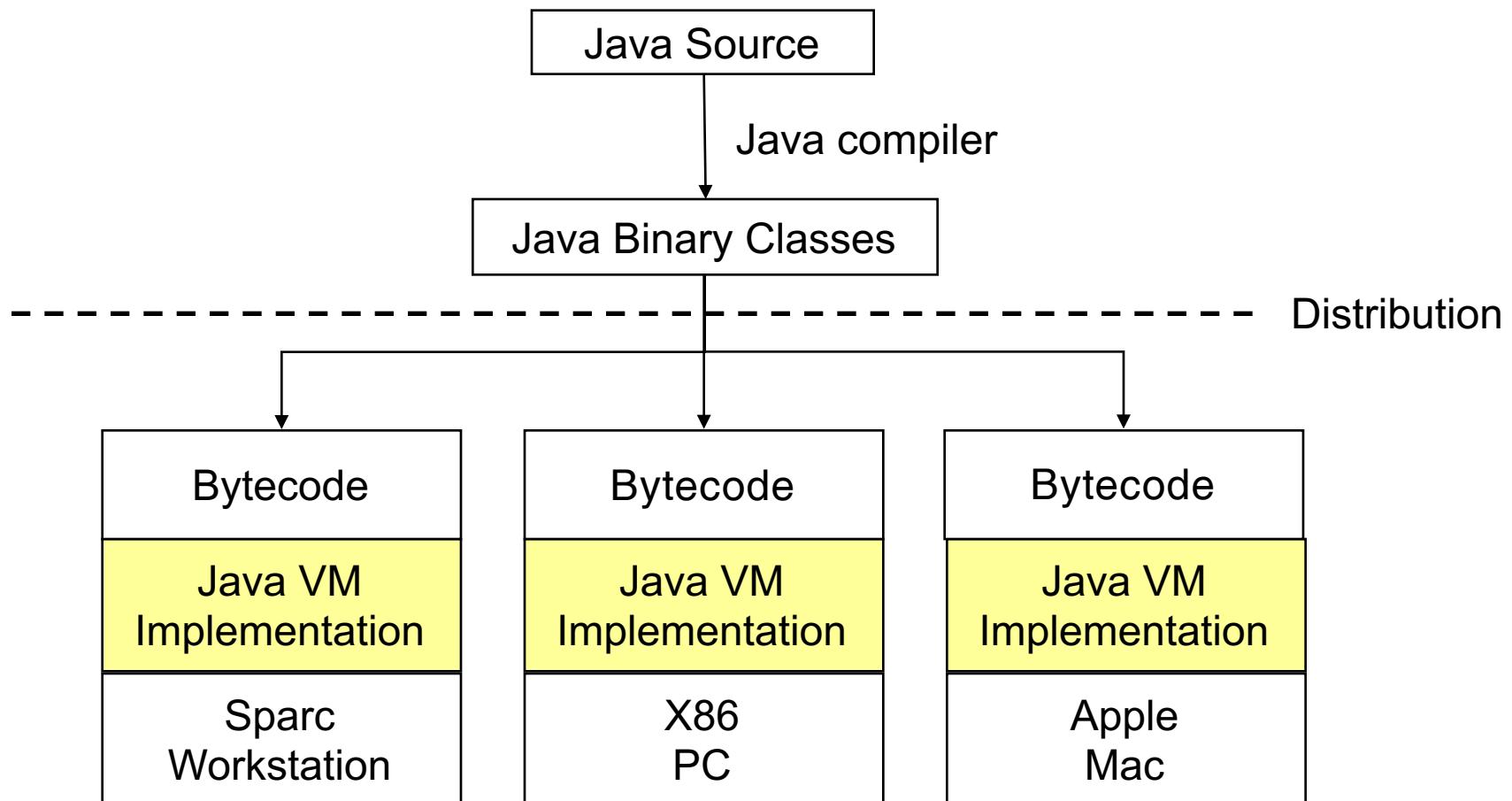
- Phần mềm ảo hóa dịch tập các lệnh OS và lệnh mức user tạo trên một platform sang các lệnh của platform khác





Máy ảo mức quá trình: ví dụ

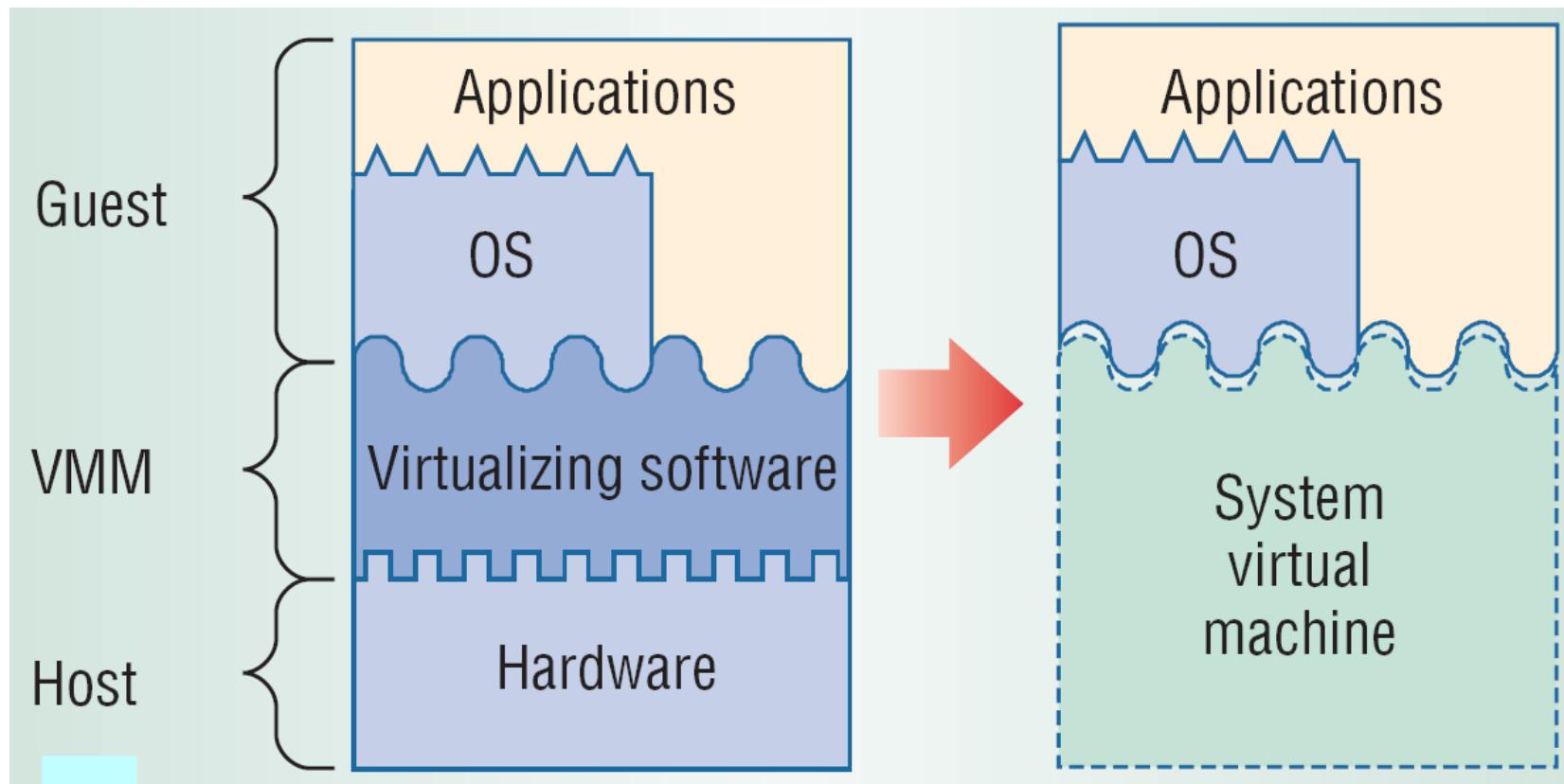
- High-level language Virtual machine (HLL VM): Java VM





Máy ảo mức hệ thống

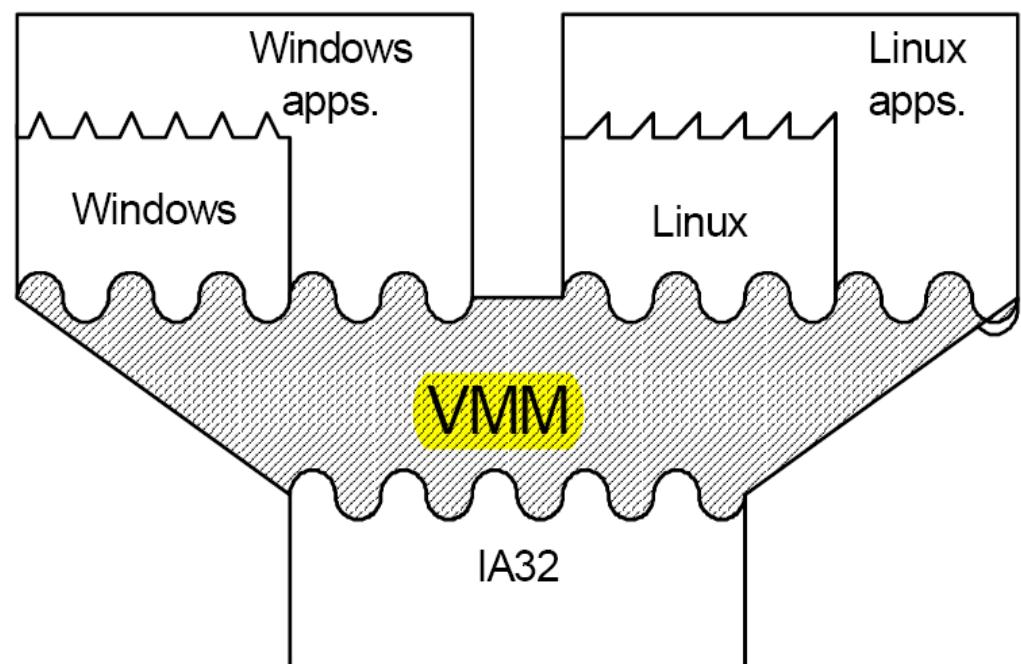
- Phần mềm ảo hóa dịch lệnh máy (ISA, Instruction Set Architecture) sử dụng bởi một hardware platform sang lệnh máy của HW platform khác





Máy ảo mức hệ thống: ví dụ

- Cách tiếp cận cổ điển cho kiến trúc máy ảo mức hệ thống





Máy ảo mức hệ thống: ví dụ

- VMWare

