Quản lý quá trình

- Khái niệm cơ bản
- Định thời CPU
- Các tác vụ cơ bản: tạo/kết thúc quá trình
- Sự cộng tác giữa các quá trình
- Giao tiếp giữa các quá trình

Khái niệm cơ bản

- Hệ thống máy tính thực thi nhiều chương trình khác nhau
  - Batch system: job
  - Time-shared system: user program, task

- *Quá trình* (*process*)
  - một chương trình đang thực thi

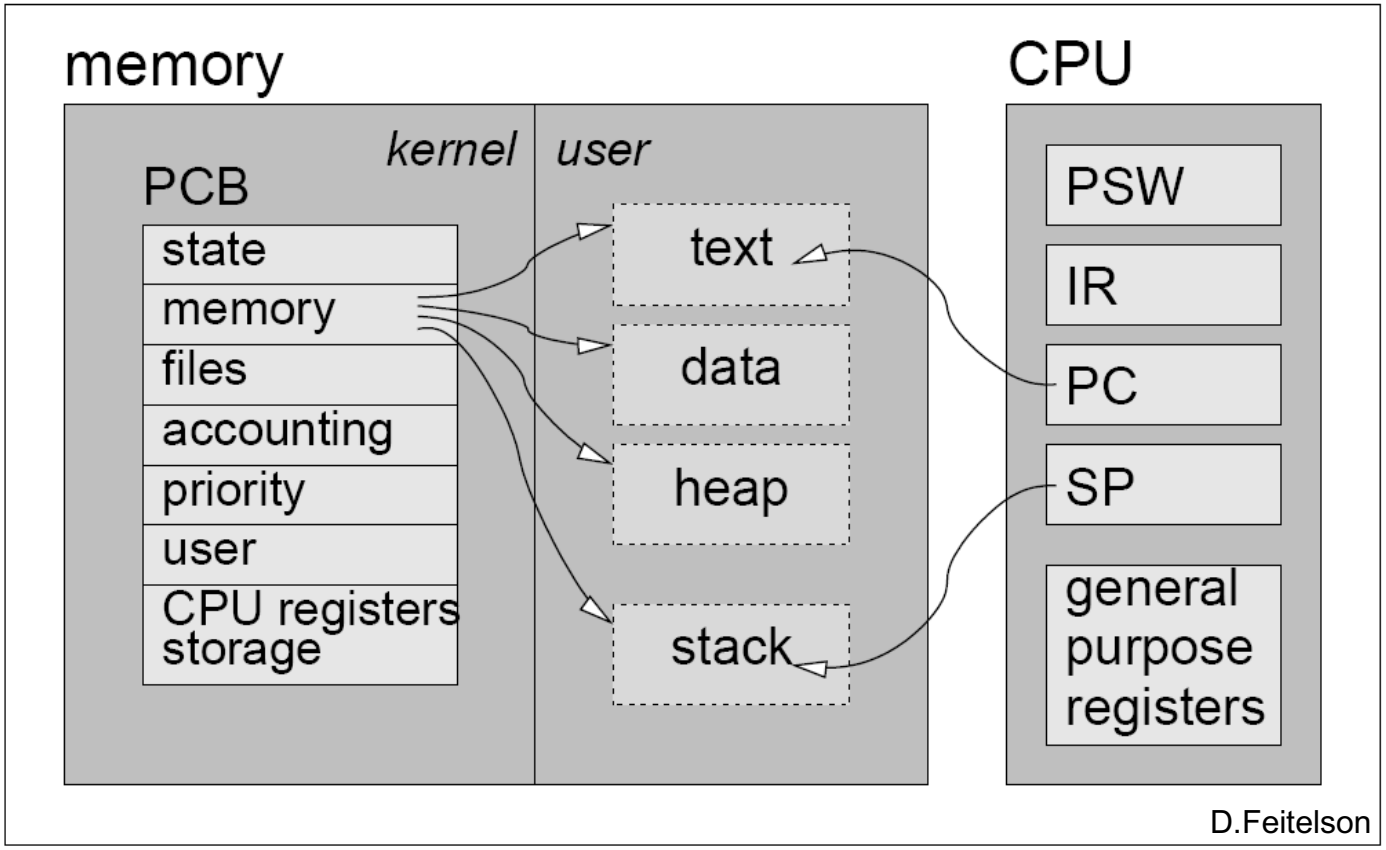
  Một quá trình được định nghĩa bởi
  - Trạng thái CPU (trị của các thanh ghi)
  - Không gian địa chỉ (nội dung bộ nhớ)
  - Môi trường (environment, xác định thông qua các bảng của hệ điều hành)

Khái niệm cơ bản (tt)

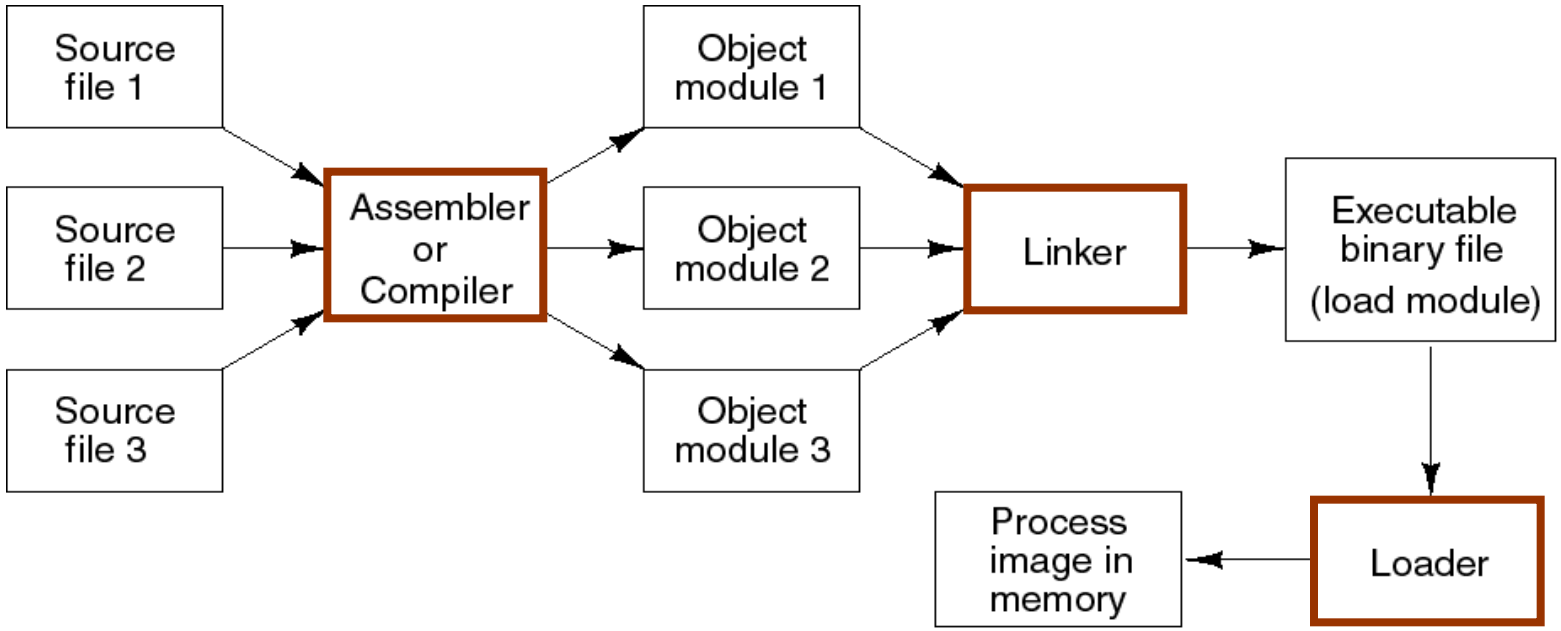
- Trạng thái CPU
  - Processor Status Word (PSW)
  - Instruction Register (IR)
  - Program Counter (PC)
  - Stack Pointer (SP)
  - Các general purpose register
- Không gian địa chỉ
  - Text (code)
  - Data
  - Heap
  - Stack
- Môi trường
  - Vd: terminal đang dùng, các open file, các kênh giao tiếp với các quá trình khác
  - Được liệt kê trong các bảng của hệ điều hành

Process control block

- Hệ điều hành lưu thông tin về quá trình trong process control block (PCB)

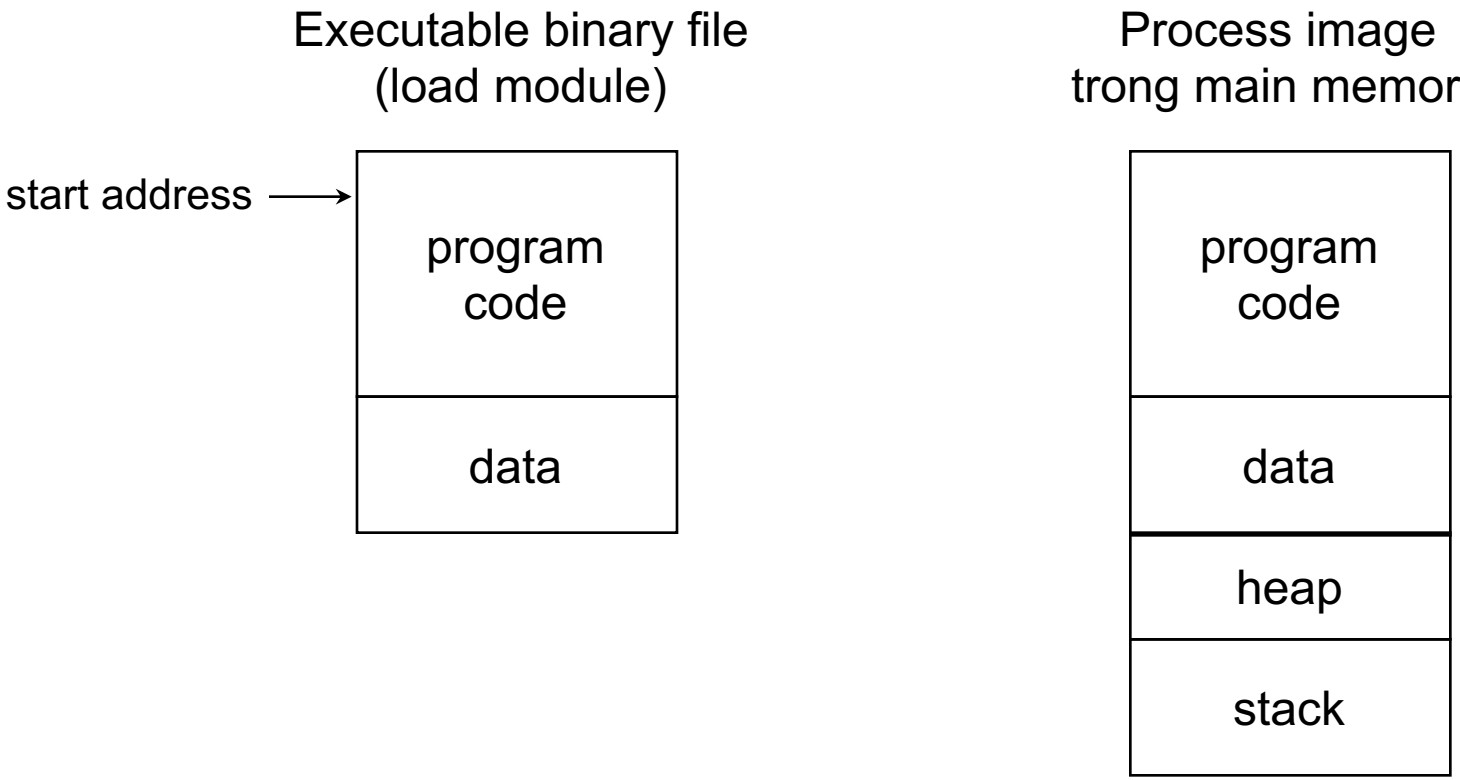


Các bước nạp chương trình vào bộ nhớ



Từ chương trình đến quá trình

- Chương trình thực thi có định dạng *load module* mà trình nạp (loader) "hiểu" được
  - Vd định dạng elf trong Linux
- Layout luận lý của *process image*



Khởi tạo quá trình

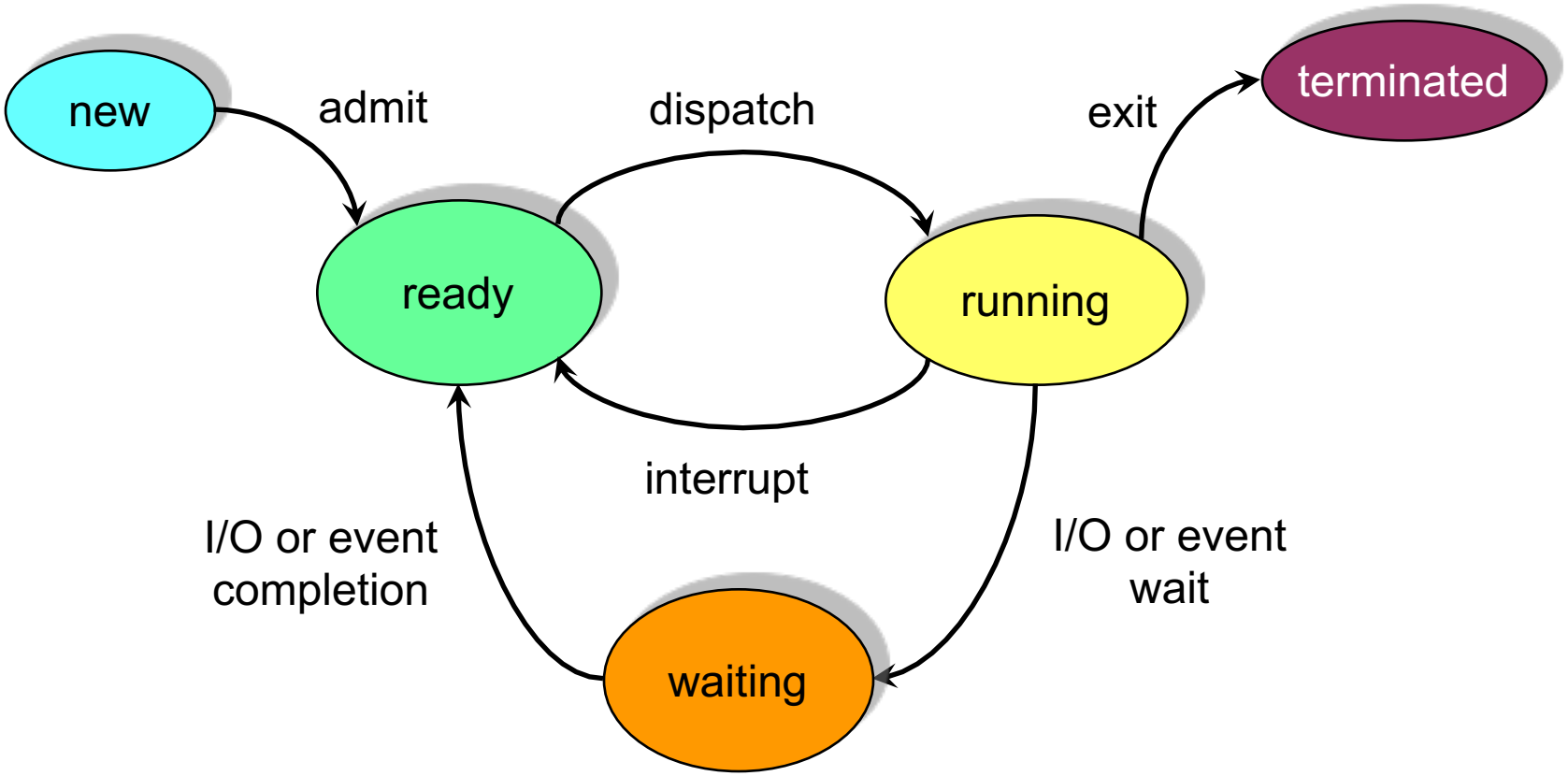
- Các bước hệ điều hành khởi tạo quá trình
  - Cấp phát một *định danh* duy nhất (process number hay process identifier, pid) cho quá trình
  - Cấp phát không gian nhớ để nạp quá trình
  - Khởi tạo khối dữ liệu Process Control Block (PCB) cho quá trình
  - Thiết lập các mối liên hệ cần thiết (vd: sắp PCB vào hàng đợi định thời,…)

Các trạng thái của quá trình (1/2)

- Các *trạng thái cơ bản* của một quá trình:
  - *new:* quá trình vừa được tạo
  - *ready:* quá trình đã có đủ tài nguyên, chỉ còn cần CPU
  - *running:* các lệnh của quá trình đang được thực thi
  - *waiting:* hay là *blocked*, quá trình đợi I/O hoàn tất, hay đợi tín hiệu
  - *terminated:* quá trình đã kết thúc

Các trạng thái của quá trình (2/2)

- Chuyển đổi giữa các trạng thái của quá trình



Ví dụ về trạng thái quá trình

```
/* test.c */
int main(int argc, char** argv)
{
   printf("Hello world\n");
   exit(0);
}
```

Biên dịch chương trình trong Linux
`gcc test.c –o test`

Thực thi chương trình test
`./test`

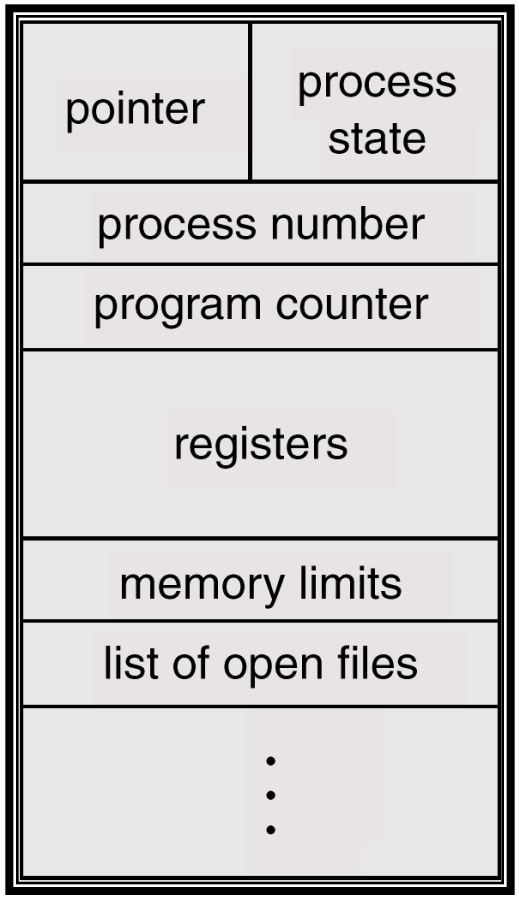
Trong hệ thống sẽ có một quá trình *test* được tạo ra, thực thi và kết thúc.

- Chuỗi trạng thái của quá trình test như sau (trường hợp tốt nhất):
  - new
  - ready
  - running
  - waiting (do chờ I/O khi gọi printf)
  - ready
  - running
  - terminated

Process Control Block

- Đã thấy là mỗi quá trình trong hệ thống đều được cấp phát một *Process Control Block* (PCB)
- PCB là một trong các cấu trúc dữ liệu quan trọng nhất của hệ điều hành

Ví dụ layout của một PCB:
(trường pointer dùng để liên kết các PCB thành một linked list)



Các trường tiêu biểu của PCB

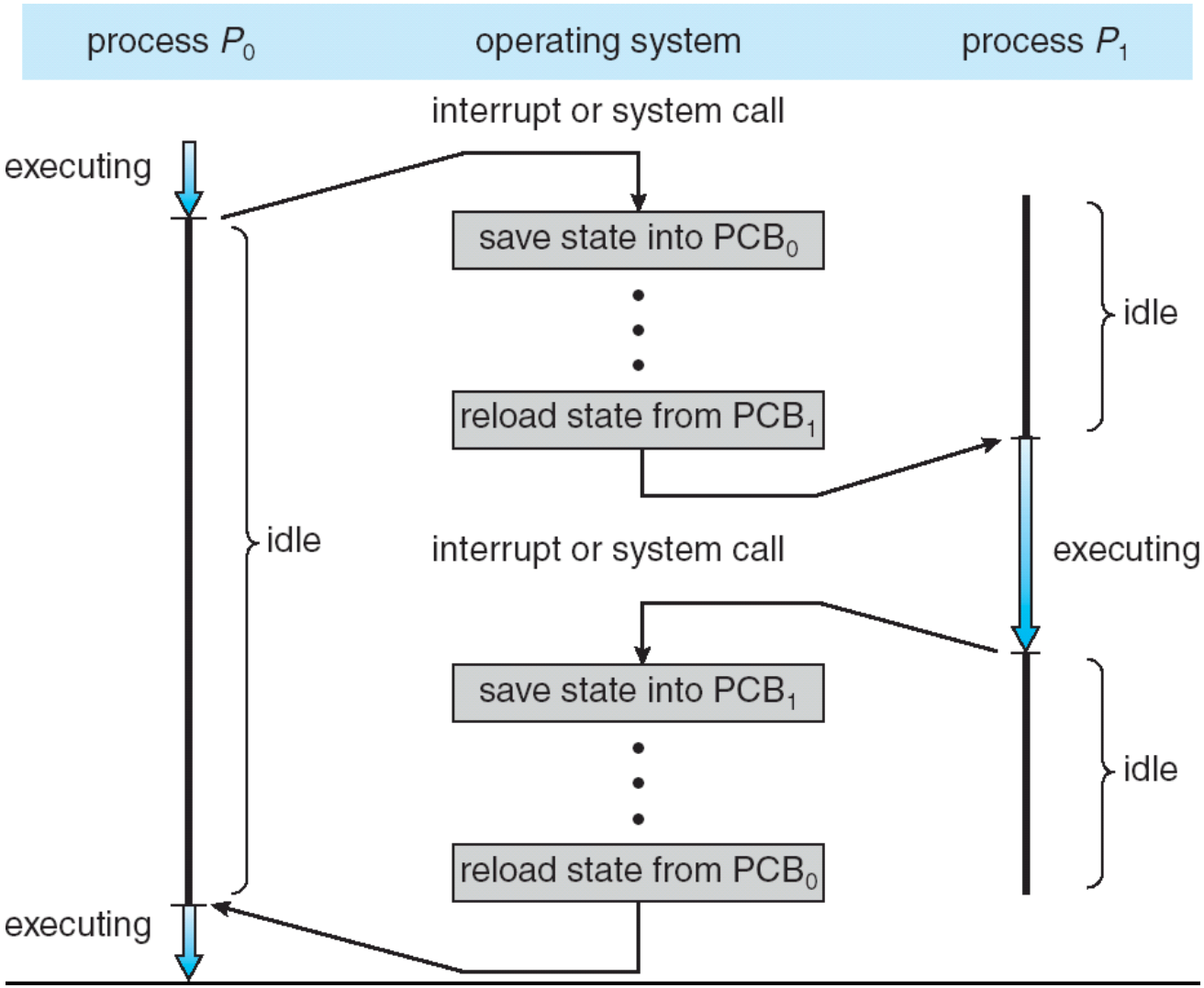
Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Tanenbaum

Chuyển ngữ cảnh (1/2)

- Làm thế nào để chia sẻ CPU giữa các quá trình?
- *Ngữ cảnh* (context) của một quá trình là tình trạng hoạt động của quá trình
  - Trị của các thanh ghi, trị của program counter, bộ nhớ,…
- Ngữ cảnh của quá trình được lưu trong PCB của nó
- *Chuyển ngữ cảnh* (context switch) là công việc ngưng quá trình đang thực thi và chạy một quá trình khác. Khi đó cần:
  - lưu ngữ cảnh của quá trình vào PCB của nó
  - nạp ngữ cảnh từ PCB của quá trình mới để quá trình mới thực thi

Chuyển ngữ cảnh (2/2)

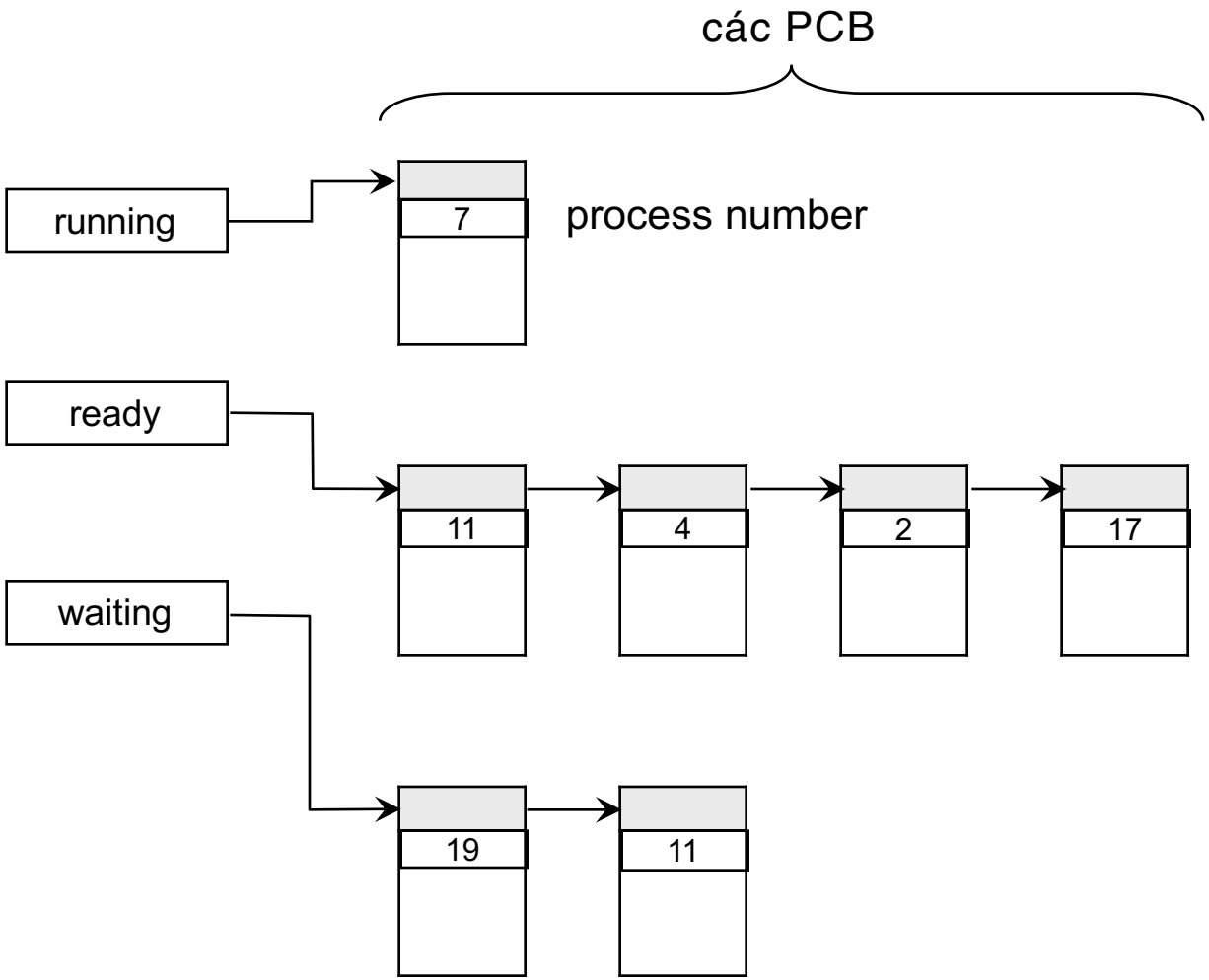


Yêu cầu đối với hệ điều hành về quản lý quá trình

- Hỗ trợ sự thực thi luân phiên giữa nhiều quá trình
  - Khi nào chọn và chọn quá trình nào để thực thi có thể tùy thuộc vào tiêu chí như
    - Hiệu suất sử dụng CPU
    - Thời gian đáp ứng
    - …
- Phân phối tài nguyên hệ thống hợp lý
  - Vấn đề deadlock, trì hoãn vô hạn định,…
- Cung cấp cơ chế hỗ trợ user tạo/kết thúc quá trình
- Cung cấp cơ chế đồng bộ và giao tiếp giữa các quá trình

Quản lý quá trình: các hàng đợi

- Ví dụ

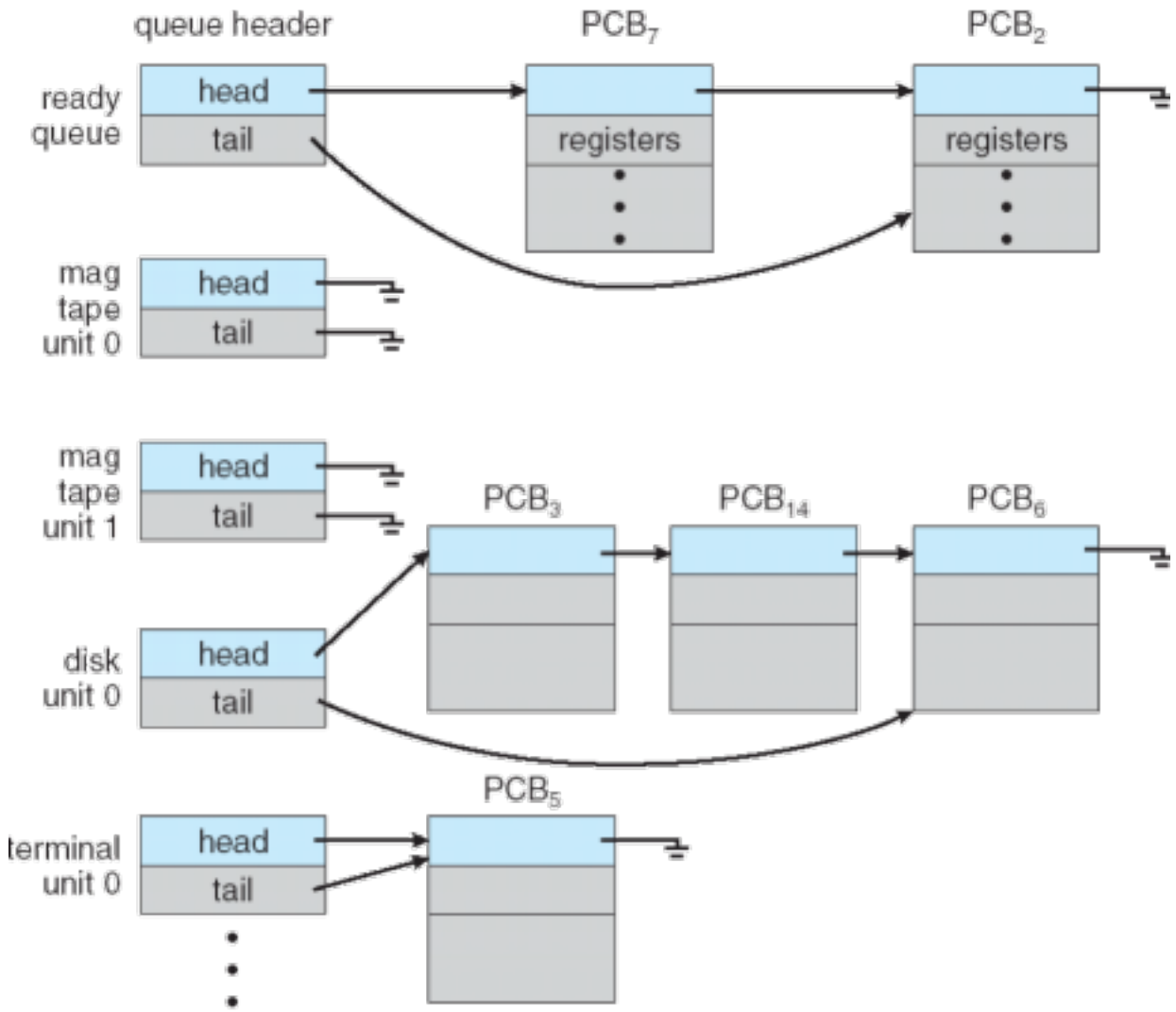


Định thời quá trình

- Tại sao phải định thời?
  - Multiprogramming
    - Có nhiều quá trình thực thi luân phiên nhau
    - Mục tiêu (ví dụ): cực đại hiệu suất sử dụng của CPU
  - Time-sharing
    - User tương tác với quá trình
    - Mục tiêu: tối thiểu thời gian đáp ứng
- Một số khái niệm cơ bản
  - Các *bộ định thời* (scheduler)
  - Các *hàng đợi định thời* (scheduling queue)

Các hàng đợi định thời

- Job queue
- Ready queue
- Các device queue
- …



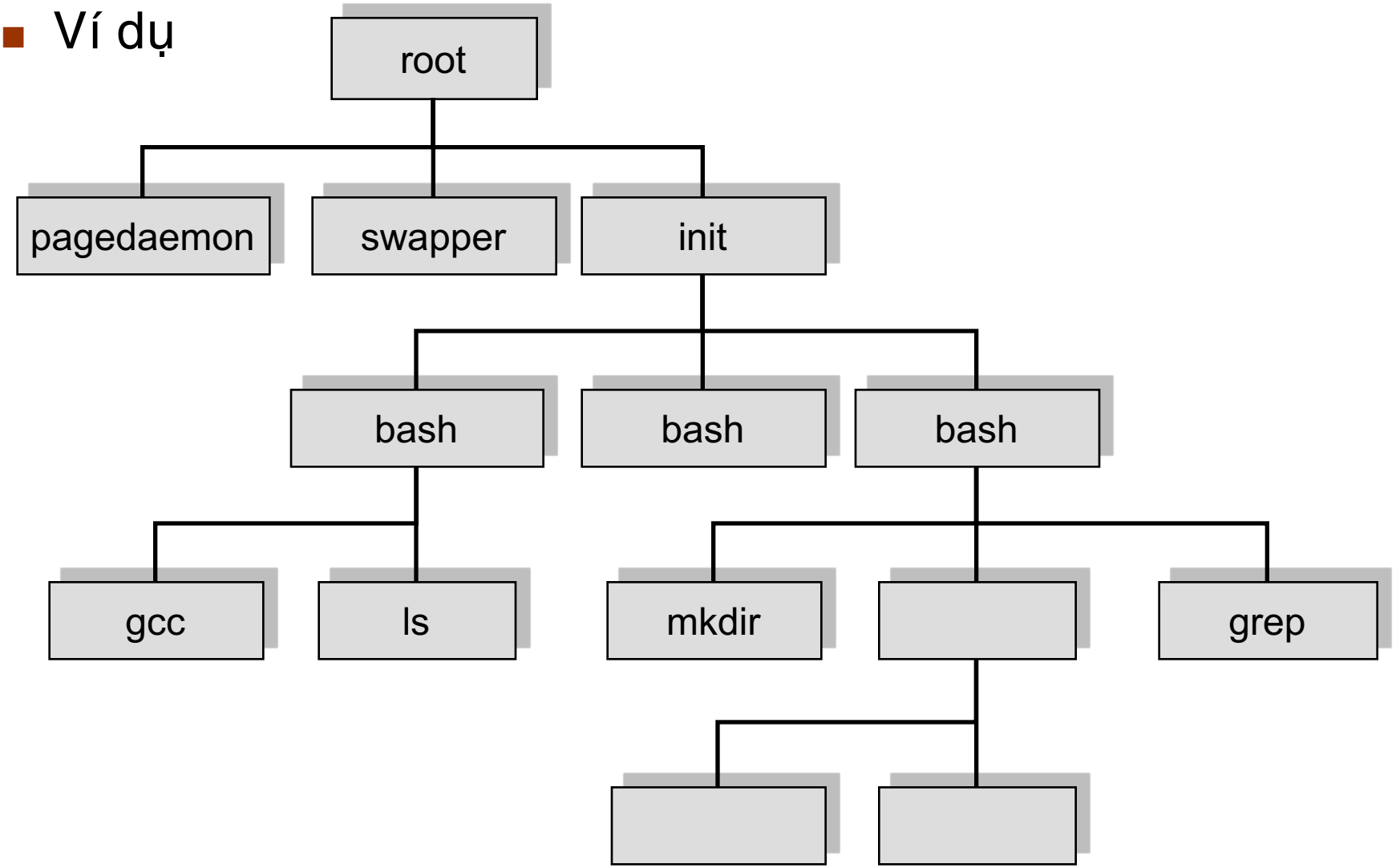
Các tác vụ đối với quá trình (1/4)

- Tạo quá trình mới
  - Quá trình có thể tạo một quá trình mới thông qua một system call (vd: hàm fork trong UNIX)
    - Ví dụ: (UNIX) Khi user đăng nhập hệ thống, một command interpreter (shell) sẽ được tạo ra cho user

  Quá trình được tạo là quá trình *con* của quá trình tạo (quá trình *cha*). Quan hệ cha-con định nghĩa một *cây quá trình.*

Cây quá trình trong Linux/Unix

- Ví dụ



Các tác vụ đối với quá trình (2/4)

- Tạo quá trình mới (tt)
  - Chia sẻ tài nguyên của quá trình cha: các khả năng
    - Quá trình cha và con chia sẻ mọi tài nguyên
    - Quá trình con chia sẻ một phần tài nguyên của cha
    - Cha và con không chia sẻ tài nguyên
  - Trình tự thực thi: hai khả năng
    - Quá trình cha và con thực thi đồng thời (concurrently)
    - Quá trình cha chạy khi quá trình con kết thúc
  - Trong Unix, quá trình gọi fork sẽ được tạo một quá trình con
    - hoàn toàn giống nó vào thời điểm gọi --- cùng trạng thái CPU, không gian địa chỉ, môi trường
    - chỉ khác nhau ở process ID và trị trả về từ fork

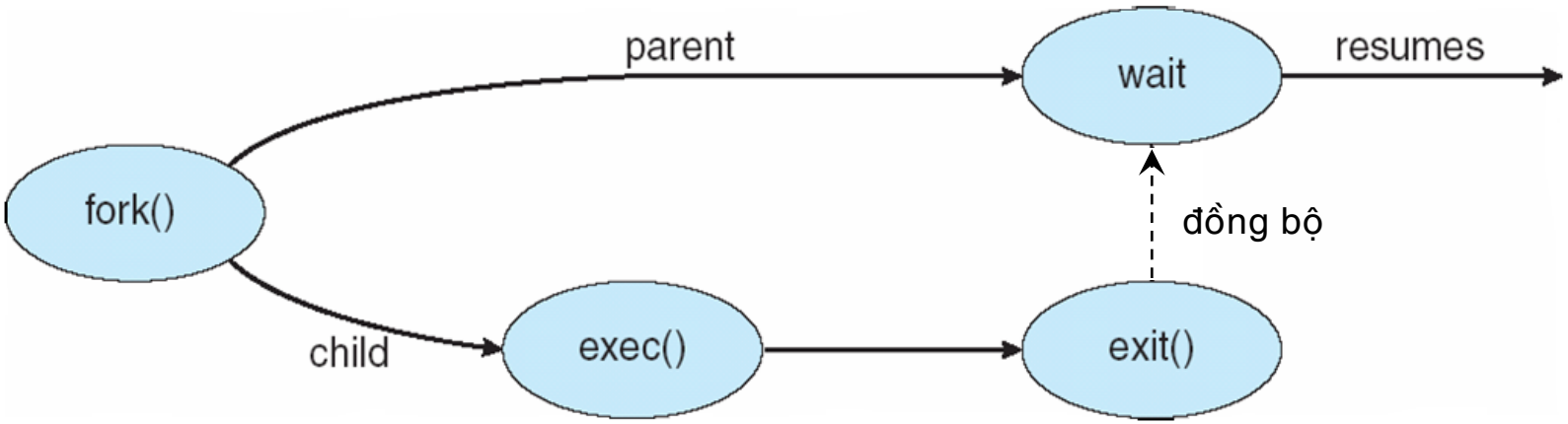
Các tác vụ đối với quá trình (3/4)

- Tạo quá trình mới (tt)
  - Không gian địa chỉ: tùy hệ điều hành
    - UNIX: Không gian địa chỉ của quá trình con được nhân bản từ không gian địa chỉ của cha vào thời điểm gọi
    - Windows: phức tạp hơn, Win32 API CreateProcess() cần 10 tham số

```
BOOL WINAPI CreateProcess(
                _In_opt_      LPCTSTR lpApplicationName,
                _Inout_opt_   LPTSTR lpCommandLine,
                _In_opt_      LPSECURITY_ATTRIBUTES lpProcessAttributes,
                _In_opt_      LPSECURITY_ATTRIBUTES lpThreadAttributes,
                _In_          BOOL bInheritHandles,
                _In_          DWORD dwCreationFlags,
                _In_opt_      LPVOID lpEnvironment,
                _In_opt_      LPCTSTR lpCurrentDirectory,
                _In_          LPSTARTUPINFO lpStartupInfo,
                 _Out_        LPPROCESS_INFORMATION lpProcessInformation
);
```

Về quan hệ cha/con

- Ví dụ áp dụng fork trong UNIX/Linux
  - Quá trình gọi fork() để tạo một quá trình con
  - Quá trình con gọi exec() để nạp và thực thi một chương trình trong không gian nhớ của nó
  - Quá trình cha làm việc khác… hay gọi wait() để đợi con xong



Ví dụ tạo process với fork()

```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[]){
   int  return_code;
   /* create a new process */
   return_code = fork();

   if (return_code > 0){
      printf("This is parent process");
      wait(NULL);
      return 0;
   }
   else if (return_code == 0){
      printf("This is child process");
      return 1;
   }
   else {
        printf("Fork error\n");
        return 1;
    }
}
```

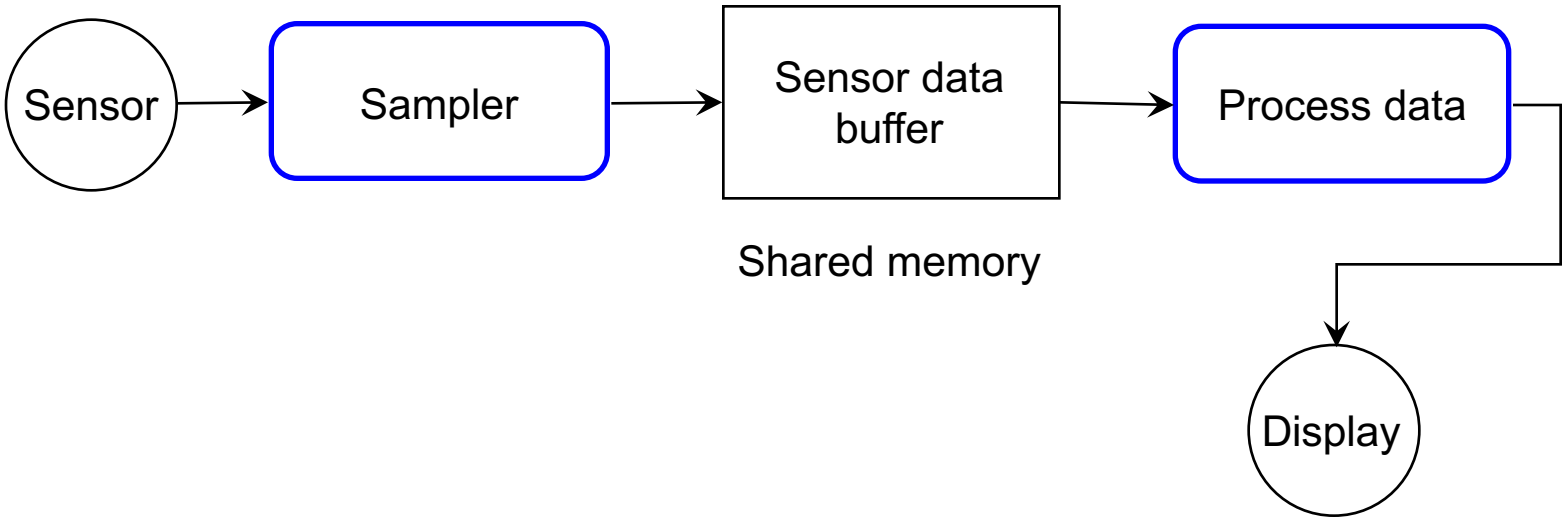
```
int main() {
    printf(“Hello world \n”);
    fork();
    printf("Hello world  \n”);
    fork();
    return 0;
   }
}
```

Các tác vụ đối với quá trình (4/4)

- Tạo quá trình mới
- Kết thúc quá trình
  - Quá trình tự kết thúc
    - Quá trình kết thúc khi thực thi lệnh cuối và gọi system routine exit
  - Quá trình kết thúc do quá trình khác (có đủ quyền, vd: quá trình cha của nó)
    - Gọi system routine abort với tham số là pid (process identifier) của quá trình cần được kết thúc
  - Hệ điều hành thu hồi tất cả các tài nguyên của quá trình kết thúc (vùng nhớ, I/O buffer,…)

Cộng tác giữa các quá trình

- Các quá trình có thể *cộng tác* (cooperate) để hoàn thành công việc
  - Vd



Cộng tác giữa các quá trình

- Thiết kế ứng dụng
  - Phân chia một ứng dụng lớn thành các process cộng tác nhau → kiến trúc client-server
- Áp dụng cộng tác giữa các quá trình để
  - → Bài toán producer-consumer
  - Modul hóa
  - Tăng tốc tính toán
    - Nếu hệ thống có nhiều CPU, chia công việc tính toán thành nhiều công việc tính toán nhỏ chạy song song
- Sự cộng tác giữa các quá trình đòi hỏi hệ điều hành cung cấp giải pháp đồng bộ hoạt động (chương 3) và giao tiếp cho các quá trình

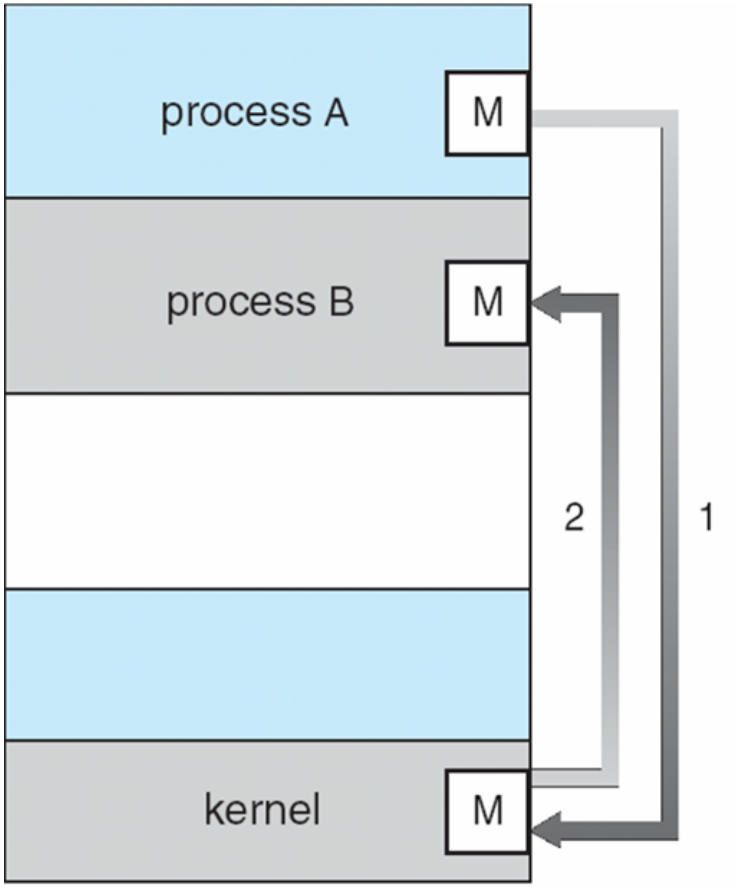
Bài toán producer-consumer

- Bài toán tiêu biểu về sự cộng tác giữa các quá trình: *bài toán producer-consumer*
  - *Producer* tạo ra các dữ liệu và *consumer* tiêu thụ / sử dụng các dữ liệu đó. Sự trao đổi dữ liệu được thực hiện qua buffer
    - *unbounded buffer:* kích thước buffer vô hạn (không thực tế)
    - *bounded buffer:* kích thước buffer có hạn
  - Producer và consumer phải hoạt động đồng bộ vì
    - Consumer không được tiêu thụ khi producer chưa sản xuất
    - Producer không được tạo thêm dữ liệu khi buffer đầy

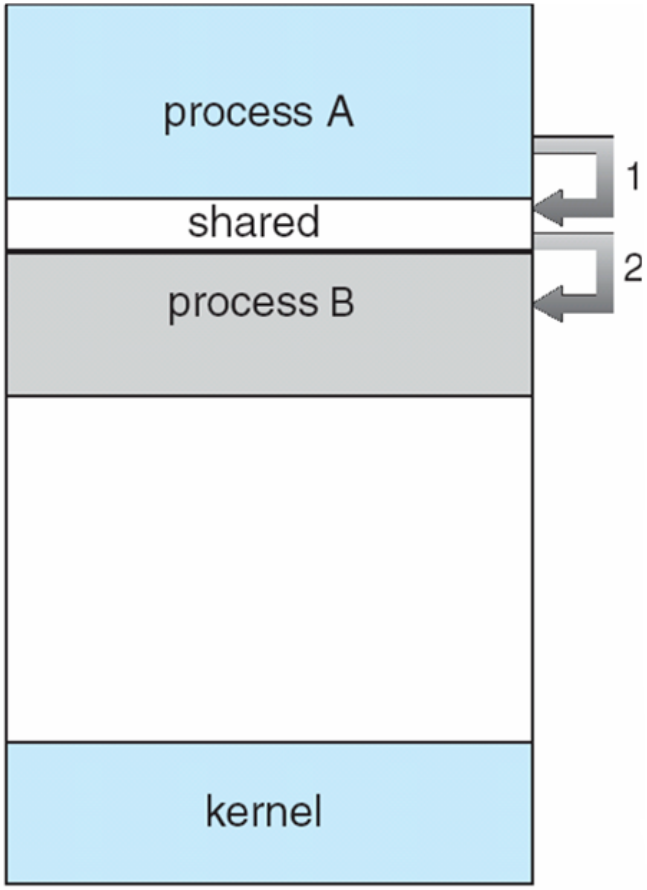
Interprocess communication (IPC)

- *IPC* là các kỹ thuật cung cấp bởi hệ điều hành nhằm giúp các quá trình giao tiếp với nhau
  - Các quá trình có thể trên cùng máy hoặc khác máy
- Hai kỹ thuật IPC
  - Truyền thông điệp (message passing)
  - Dùng bộ nhớ chia sẻ (shared memory)

Mô hình giao tiếp



Truyền thông điệp



Dùng bộ nhớ chia sẻ

3.4 Fig 3.12

Truyền thông điệp

- Các vấn đề
  - *Naming*
    - Giao tiếp *trực tiếp*
      - send(P, msg): gửi thông điệp đến quá trình P
      - receive(Q, msg): nhận thông điệp đến từ quá trình Q
    - Giao tiếp *gián tiếp*: thông qua *mailbox* hay *port*
      - send(A, msg): gửi thông điệp đến mailbox A
      - receive(B, msg): nhận thông điệp từ mailbox B
  - *Synchronization*:
    - blocking/nonblocking send
    - blocking/nonblocking receive

Example of shared memory for IPC

- POSIX Shared Memory
  - Process first creates shared memory segment

```
segment_id = shmget(IPC_PRIVATE, size, S_IRUSR |
  S_IWUSR);
```

  - Process wanting access to that shared memory must attach to it

```
shared_memory = (char *) shmat(id, NULL, 0);
```

  - Now the process could write to the shared memory

```
sprintf(shared_memory, "Writing to shared memory");
```

  - When done a process can detach the shared memory from its address space

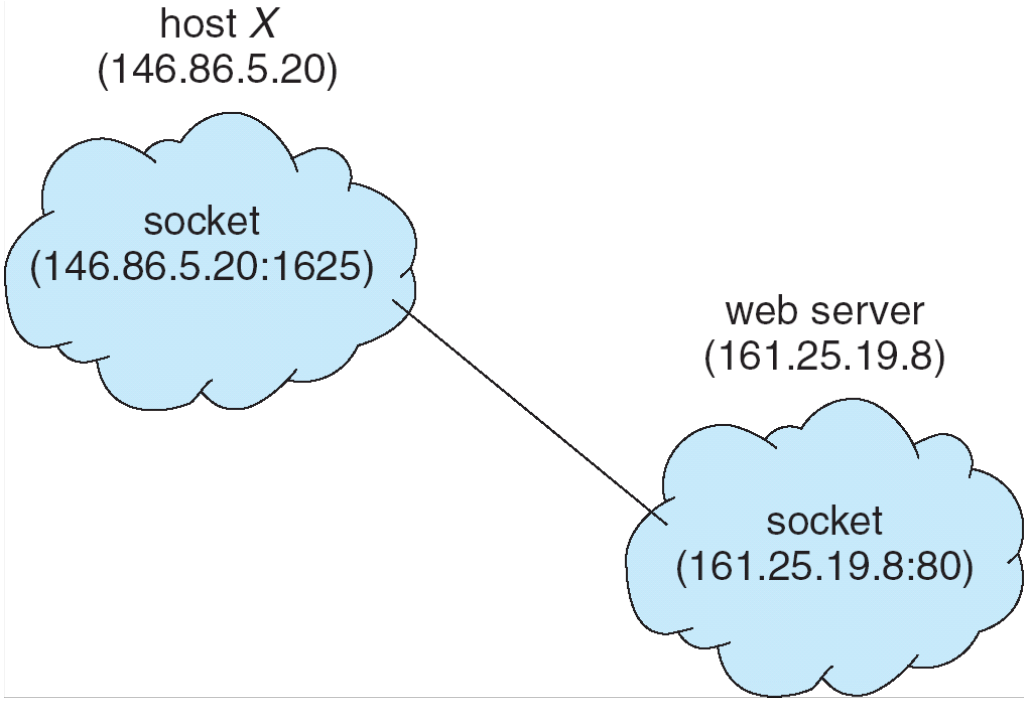
```
shmdt(shared_memory);
```

Giao tiếp trong hệ thống client-server

- Socket
- Remote Procedure Call (RPC)
- Remote Method Invocation (RMI)

Socket

- *Socket* là một đối tượng trừu tượng dùng để tượng trưng một đầu cuối của một kênh giao tiếp
  - Gồm địa chỉ IP và port number
  - Vd socket **161.25.19.8:1625** dùng để tham chiếu port **1625** trên máy có địa chỉ IP **161.25.19.8**



Socket

- (tt) 'well-known' port xác định các dịch vụ chuẩn

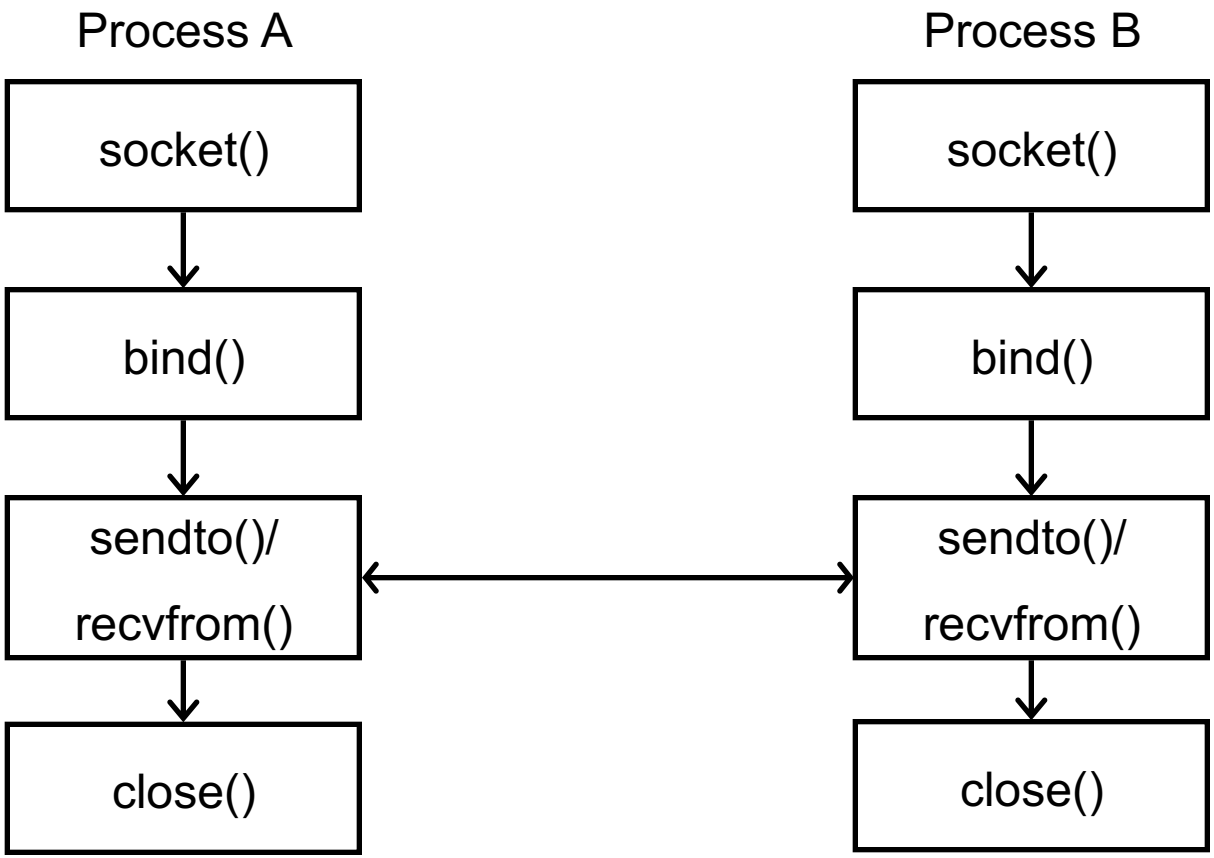
port	*usage*
21	ftp
23	telnet
25	smtp (email)
42	name server
70	gopher
79	finger
80	http (web)

- Cung cấp cơ chế giao tiếp mức thấp: gửi nhận một chuỗi byte dữ liệu không cấu trúc
- Hai loại giao tiếp qua socket: *connectionless* và *connection-oriented*

Gửi/nhận qua socket

Hàm thư viện	***Diễn giải***
socket()	Tạo một socket
bind()	Gán một địa chỉ cục bộ vào socket
listen()	Sẵn sàng để chấp nhận kết nối
accept()	(server) Chờ kết nối đến từ client
connect()	(client) kết nối đến một server
send() sendto()	Gửi dữ liệu qua kênh giao tiếp đã thiết lập Gửi dữ liệu đến một địa chỉ
recv() recvfrom()	Nhận dữ liệu qua kênh giao tiếp đã thiết lập Nhận dữ liệu đến từ một địa chỉ
close()	Đóng kết nối

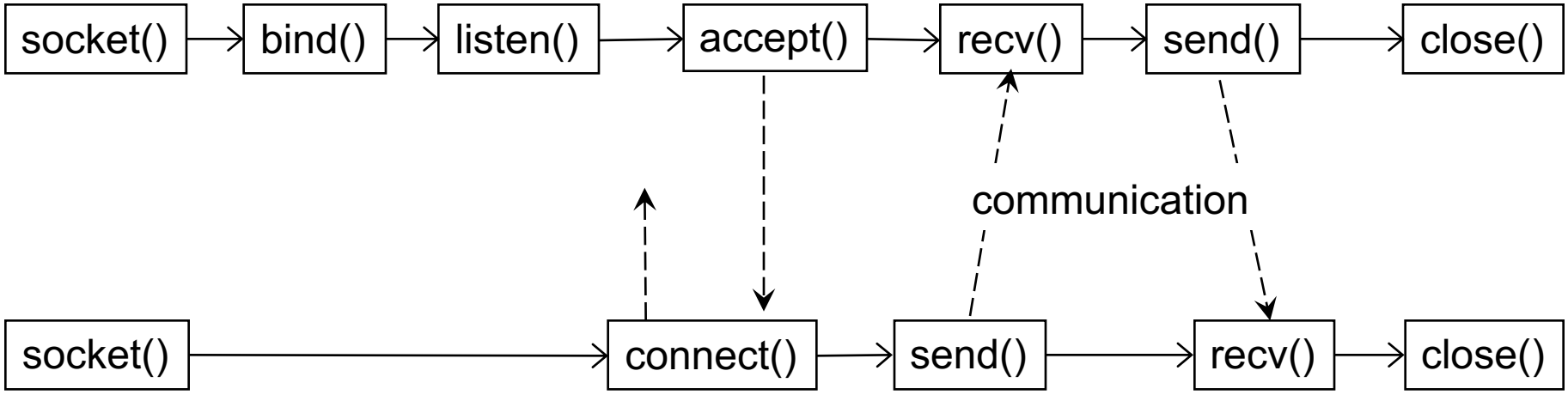
Connectionless Transport Service



- sendto(socket, buffer, buffer_length, flags, *destination_address*, addr_len)
- recvfrom(socket, buffer, buffer_length, flags, *from_address*, addr_len)

Connection-Oriented Transport Service

Server (Google)



Client (Firefox)

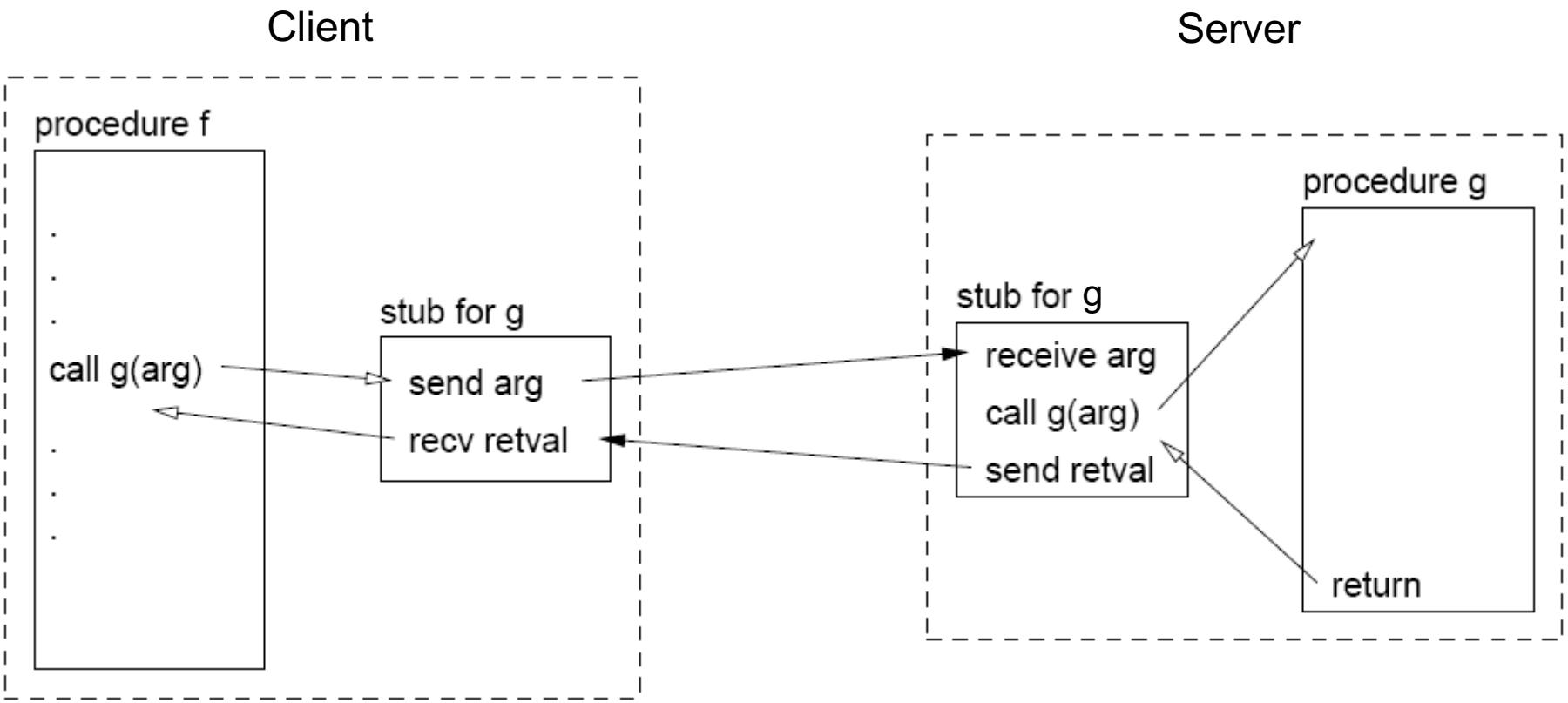
- send(socket, buffer, buffer_length, flags)
- recv(socket, buffer, buffer_length, flags)

Remote procedure call (RPC)

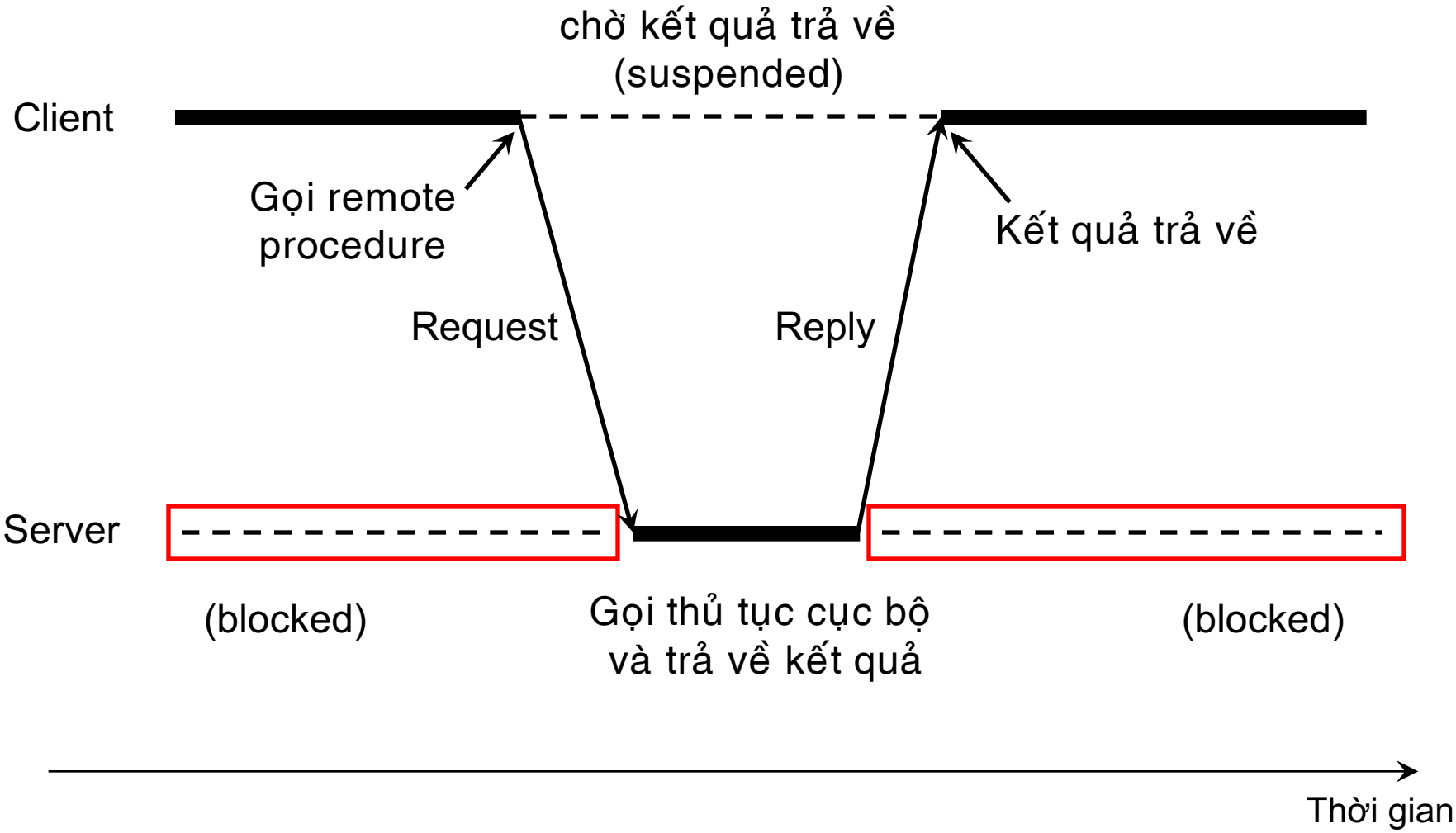
- Mở rộng cơ chế gọi thủ tục (procedure call) thông thường
  - Dùng RPC một quá trình có thể gọi một thủ tục nằm trên máy tính ở xa qua mạng

- Các vấn đề khi hiện thực RPC
  - Truyền tham số
  - Biểu diễn dữ liệu
  - Kết nối từ client đến server (binding)
  - Giao thức vận chuyển
  - Xử lý lỗi (exception handling)
  - An toàn (security)

Hiện thực RPC: các stub



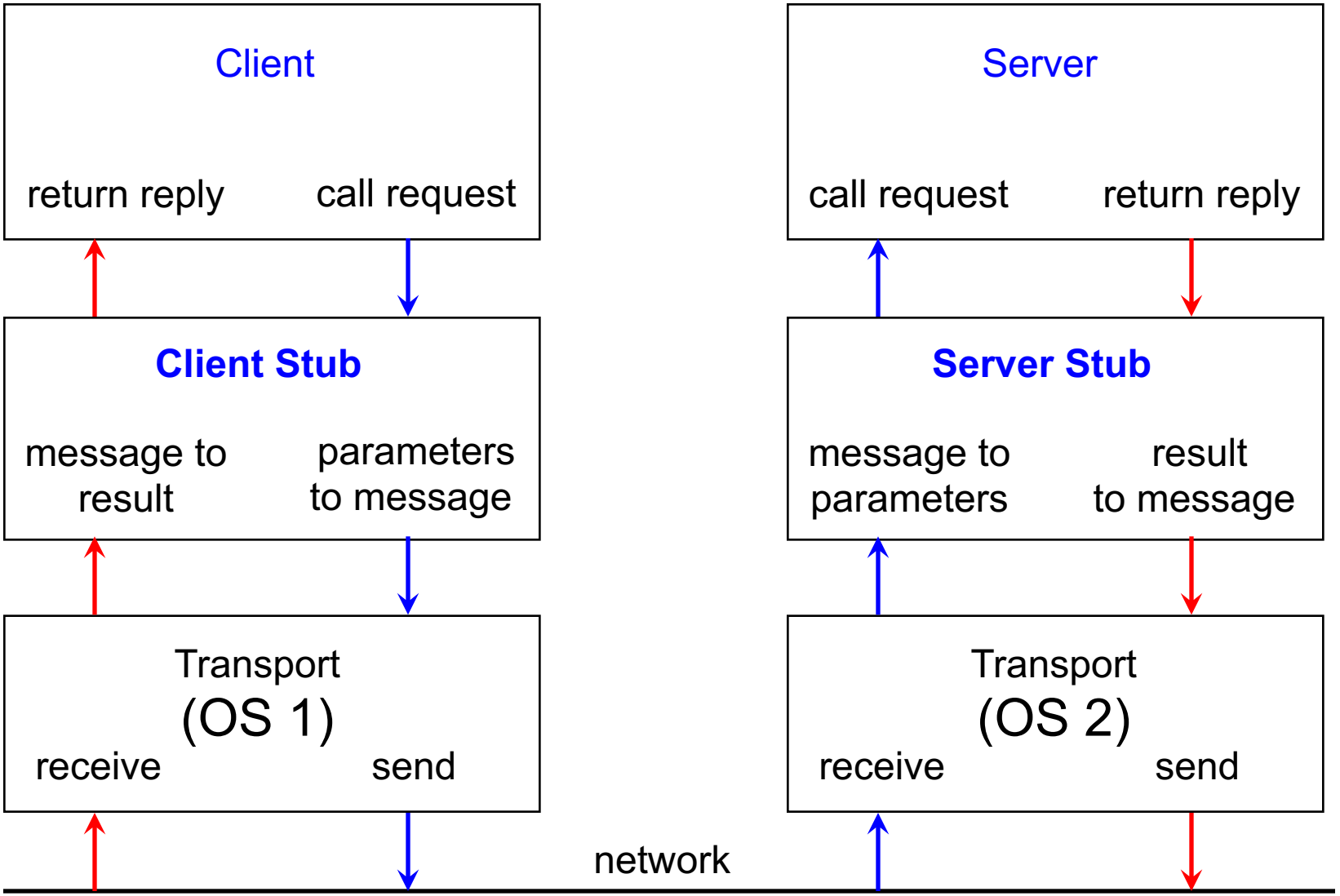
Hiện thực RPC: lưu đồ thời gian



Marshalling / Unmarshalling

- Marshalling là công việc đóng gói dữ liệu vào trong một thông điệp dưới một định dạng thích hợp để vận chuyển qua mạng
  - Unmarshalling là công việc ngược lại
- Dùng marshalling / unmarshalling trong hiện thực RPC trong các công đoạn
  - Gửi yêu cầu đến server
    - Client stub đóng gói các tham số của thủ tục vào trong gói thông điệp và gửi nó đến server stub
    - Server stub mở gói lấy các tham số từ thông điệp và dùng chúng để gọi thủ tục của server
  - Server trả về kết quả
    - Server stub đóng gói kết quả trước khi gửi đến client stub
    - Client stub mở gói lấy kết quả và chuyển nó đến client

Marshalling / Unmarshalling

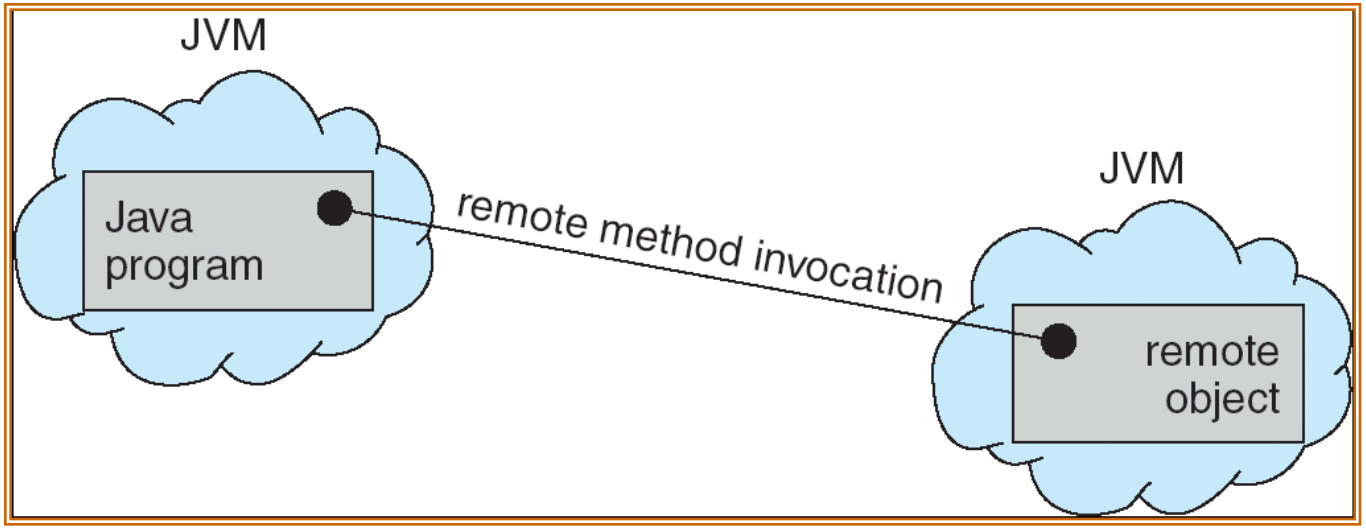


Hiện thực RPC: biểu diễn dữ liệu

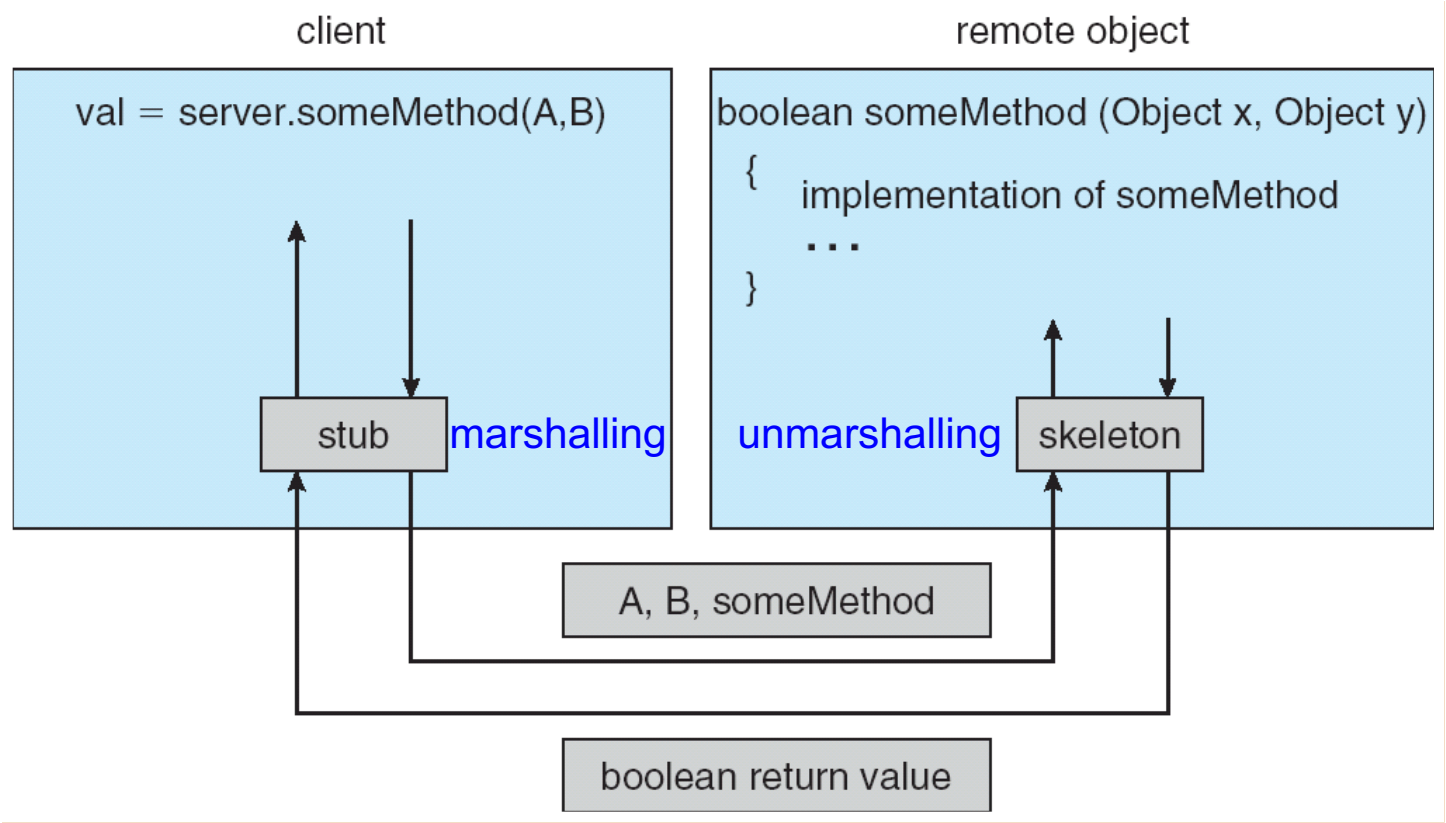
- Dữ liệu trên các hệ thống khác nhau có thể được biểu diễn khác nhau
  - Mã hóa ký tự: ASCII, EBCDIC
  - Ví dụ biểu diễn 32-bit integer trong bộ nhớ
    - Motorola: *big-endian* → most significant byte tại high memory address ()
    - Intel x86: *little-endian* → least significant byte tại high memory address ()
- Giải pháp: mã hóa dữ liệu dùng dạng biểu diễn độc lập máy XDR (External Data Representation) khi trao đổi dữ liệu giữa các hệ thống máy khác nhau

Remote method invocation (RMI)

- Cho phép một chương trình Java có thể gọi một phương thức (method) của một *đối tượng ở xa*, nghĩa là một đối tượng ở tại một máy ảo Java khác



Marshalling tham số trong RMI



Phương thức được triệu gọi có dạng sau:

boolean someMethod(Object x, Object y)