Chương 2
KHÁI NIỆM VỀ QUÁ TRÌNH

Nội dung

- Khái niệm cơ bản
- Định thời process (CPU scheduling)
- Các tác vụ trên process (tạo process, kết thúc process)
- Sự cộng tác giữa các process
- Interprocess Communication (IPC)
- Mô hình giao tiếp Client-Server

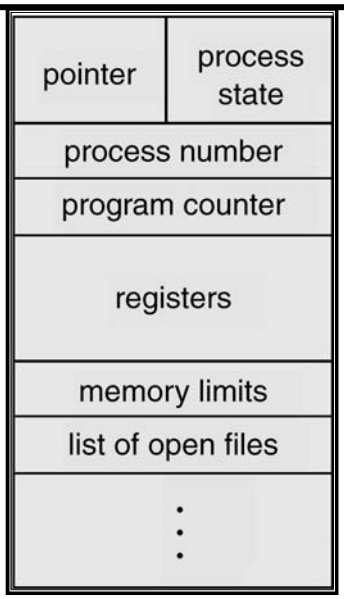
Khái niệm cơ bản

- OS thực thi nhiều chương trình khác nhau
  - Batch system: jobs
  - Time-shared systems: user programs, tasks
  - Job ≈ process
- Process
  - một *chương trình* đang thực thi (executing program).
- Một process bao gồm các phần
  - Text section(program code), data section(global variable), stack (local variable,…)
  - Hardware: Program Counter(PC), Process Status Word (PSW), Stack Pointer (SP), Memory Management Registers
- So sánh process và program
  - Process = active <> passive = programing

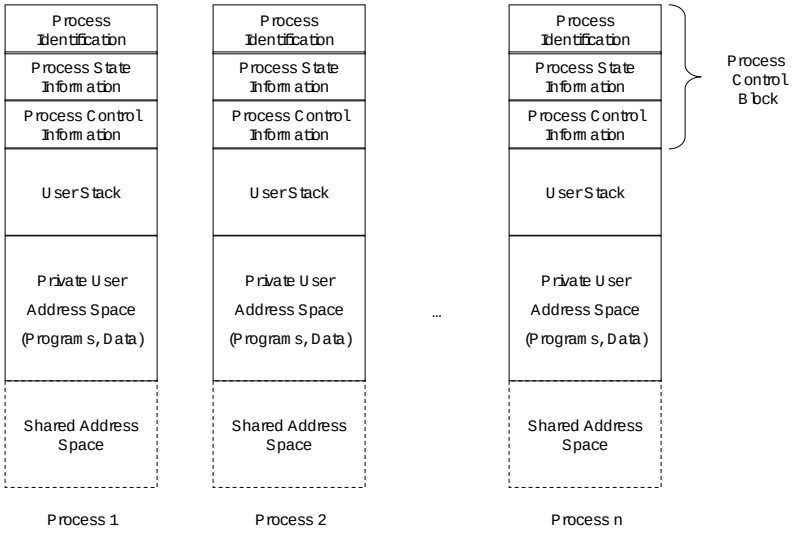
Process Control Block (PCB)

- Chứa các thông tin ứng với mỗi process.
  - Process ID, parent process ID
  - Credentials (user ID, group ID, effective ID,...)
  - Trạng thái process : new, ready, running, waiting…
  - Program counter: địa chỉ của lệnh kế tiếp sẽ thực thi
  - Các thanh ghi CPU
  - Thông tin dùng để định thời CPU: priority,...
  - Thông tin bộ nhớ: base/limit register, page tables…
  - Thông tin thống kê: CPU time, time limits…
  - Thông tin trạng thái I/O: danh sách thiết bị I/O được cấp phát, danh sách các file đang mở,...
  - Con trỏ (pointer) đến PCBs khác.

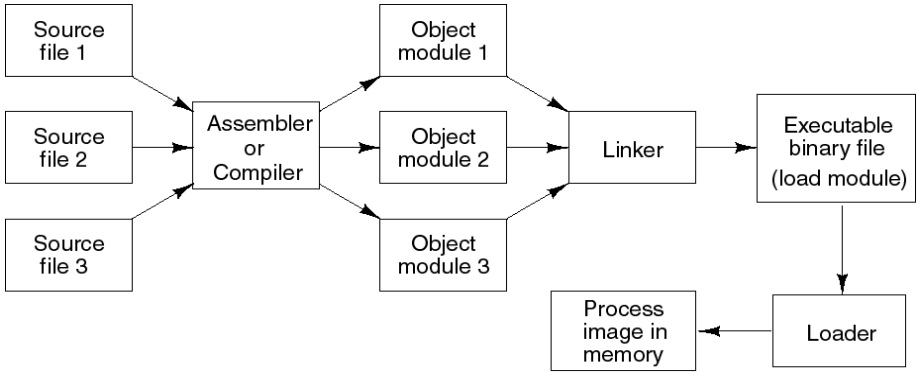
Process Control Block (PCB)



User Processes in Virtual Memory

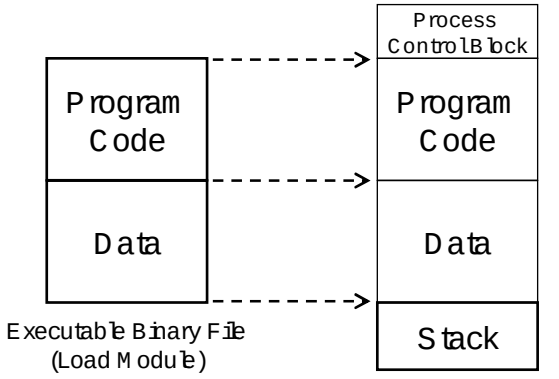


Các bước nạp process vào bộ nhớ



- Linker: kết hợp các object modules thành một file nhị phân có thể thực thi (executable binary file) gọi là load module
- Loader: nạp load module vào bộ nhớ chính

Loader



❖ Program → Process

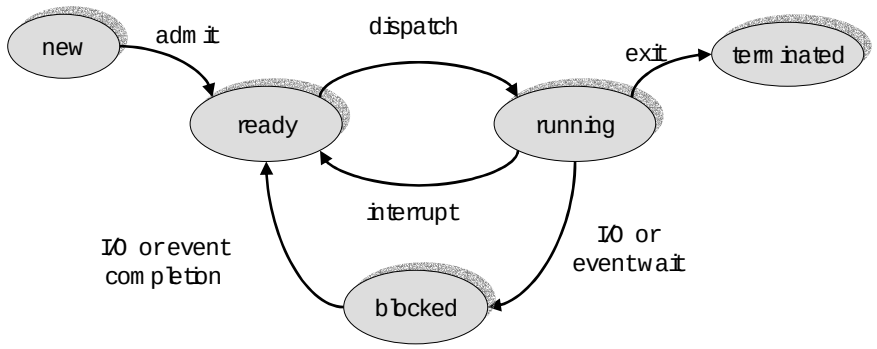
Yêu cầu đối với OS

- OS phải hỗ trợ sự thực thi luân phiên giữa nhiều process để tối ưu hiệu suất CPU với một thời gian đáp ứng có thể chấp nhận được (reasonable response time) → định thời CPU
- OS phải phân phối tài nguyên hệ thống (resources) cho processes (bộ nhớ, thiết bị I/O,...) đồng thời phải tránh hiện tượng deadlock
- OS phải cung cấp cơ chế giao tiếp giữa các process khác nhau (inter-process communication), cơ chế đồng bộ hoạt động các process (synchronization)
- OS phải cung cấp cơ chế hỗ trợ cho user tạo và hủy các process.

Các trạng thái của process

- Khi thực thi, process chuyển từ trạng thái này sang trạng thái khác, bao gồm
  - New: process mới vừa được tạo ra (được bỏ vào job queue)
  - Ready: process chờ được chiếm CPU để làm việc (được xếp vào ready queue)
  - Running: các lệnh của process đang được thực hiện.
  - Waiting: process chờ một sự kiện nào đó xảy ra, ví dụ một thao tác I/O vừa hoàn tất,... (xếp vào waiting queue)
  - Terminated: sự thực thi của process kết thúc

Lưu đồ 5-trạng thái process



- Chỉ có một process ở trạng thái *running* trên mỗi processor tại một thời điểm
- Có thể có nhiều process ở trạnh thái *ready* và *waiting*

Ví dụ về trạng thái process

test.c

```
void main()
{
   printf("Hello World\n");
}
```

Biên dịch trong Linux/Unix
$ gcc test.c –o test

Thực thi chương trình test
$ ./test

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

- Chuỗi trạng thái của process *test* như sau:
  - new
  - ready
  - running
  - blocked (chờ I/O)
  - ready
  - running
  - terminated

Các trạng thái của process (t.t)

- Trạng thái *New*
  - OS thực hiện các tác vụ cần thiết để tạo process
    - » Tạo một định danh cho process (process identifier – pid)
    - » Tạo các cấu trúc để quản lý process
      - – Memory table, file table, Process Control Block (PCB),...
  - Process mới tạo ra có thể chưa được thực thi ngay, bởi vì tài nguyên hệ thống có hạn, thông thường chỉ phục vụ một process tại một thời điểm. Process có thể đặt trong bộ nhớ thứ cấp để tiết kiệm không gian bộ nhớ chính
- Trạng thái *Terminated*
  - Process không còn được thực thi nữa
  - Process và các cấu trúc quản lý process không cần thiết sẽ bị xóa.

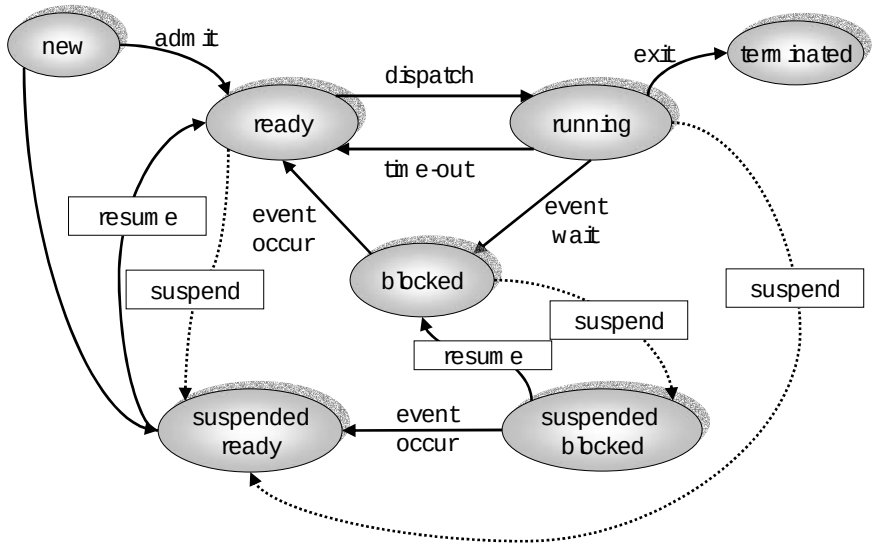
Các trạng thái của process (t.t)

- Nếu process đang trong trạng thái ready hay running và bị suspend, nó sẽ rơi vào trạng thái suspended ready
- Nếu process đang trong trạng thái blocked và bị suspend, nó sẽ rơi vào trạng thái suspended blocked
- Trạng thái suspend có thể thay đổi nếu users hoặc OS thực hiện tác vụ resume.
- Process có thể chuyển từ trạng thái suspended blocked sang suspended ready nếu có sự kiện I/O tương ứng làm cho quá trình đó bị blocked xảy ra.

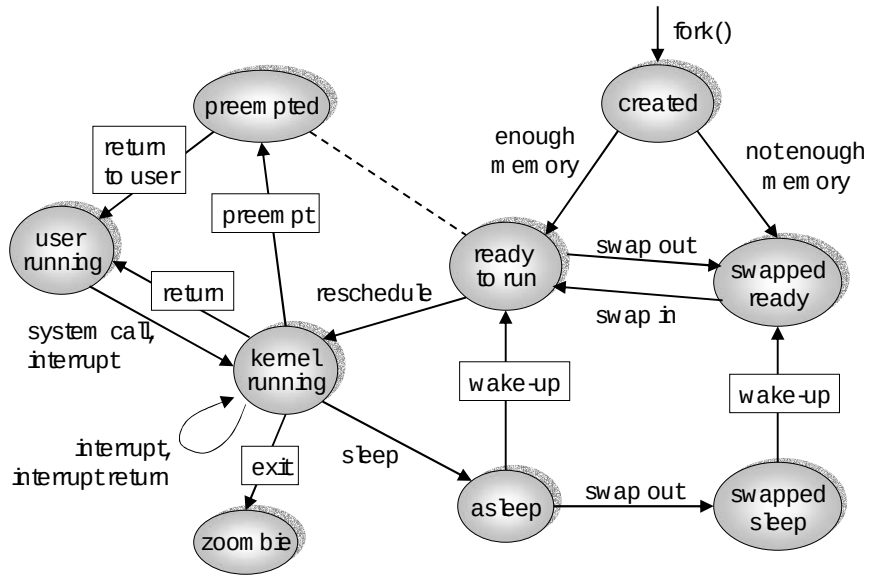
Các trạng thái của process (t.t)

- Process bị suspend trong các trường hợp sau
  - User muốn tạm dừng quá trình thực thi của process để xem kết quả thực hiện, phát hiện lỗi,…
  - Người quản trị hệ thống có thể suspend một số process để thu hồi một số tài nguyên và OS có thể cấp phát cho process khác nhằm giảm tình trạng quá tải trong hệ thống
  - Trường hợp có tranh chấp tài nguyên giữa các process, suspend có thể giúp hệ thống thoát khỏi tình trạng deadlock (tham khảo thêm phần Deadlock)
- Khi rơi vào trạng thái suspend, process được swap ra hệ thống lưu trữ thứ cấp, nhường chỗ trong bộ nhớ cho process khác.

Lưu đồ 7-trạng thái của process



Trạng thái process trong Unix



Trạng thái Unix process (t.t)

- User Running: process thực thi ở user-mode
- Kernel Running: process thực thi ở kernel-mode
- Ready To Run (in memory): trong hàng đợi ready
- Pre-empted: hàng đợi ready đặc biệt, process từ kernel-mode về user-mode nhưng bị OS đoạt quyền (preempt), thực hiện chuyển ngữ cảnh và chuyển quyền điều khiển cho process khác.
- Asleep (in Memory): trạng thái blocked, chờ sự kiện (event) hoặc tác vụ I/O
- Swapped-Ready: process sẵn sàng nhưng cần phải nạp process từ bộ nhớ thứ cấp.
- Swapped-Sleep: Đang đợi (blocked) thì bị swap out ra bộ nhớ thứ cấp, nhường bộ nhớ chính cho process khác.

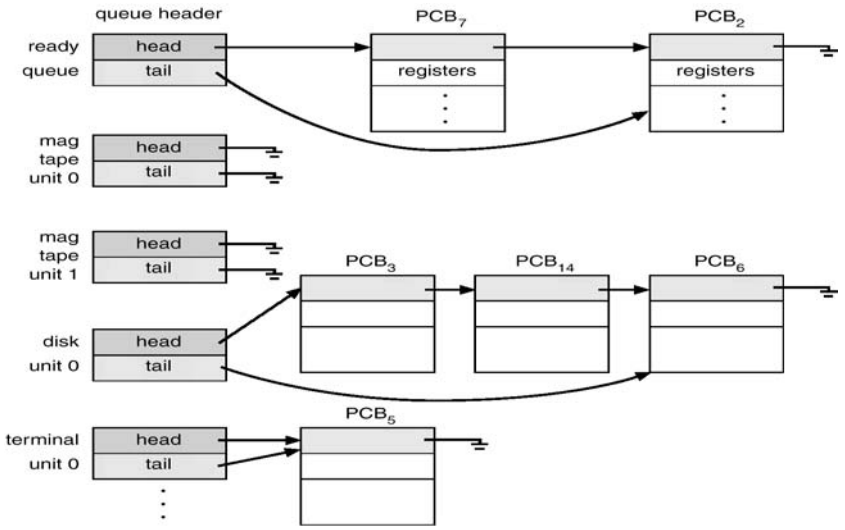
Định thời Process – Mục tiêu

- Multiprogramming
  - Có nhiều process phải thực thi luân phiên nhau
  - Cực đại hiệu suất của CPU
- Time Sharing
  - Cho phép users tương tác khi chương trình đang chạy
  - Tối thiểu thời gian đáp ứng

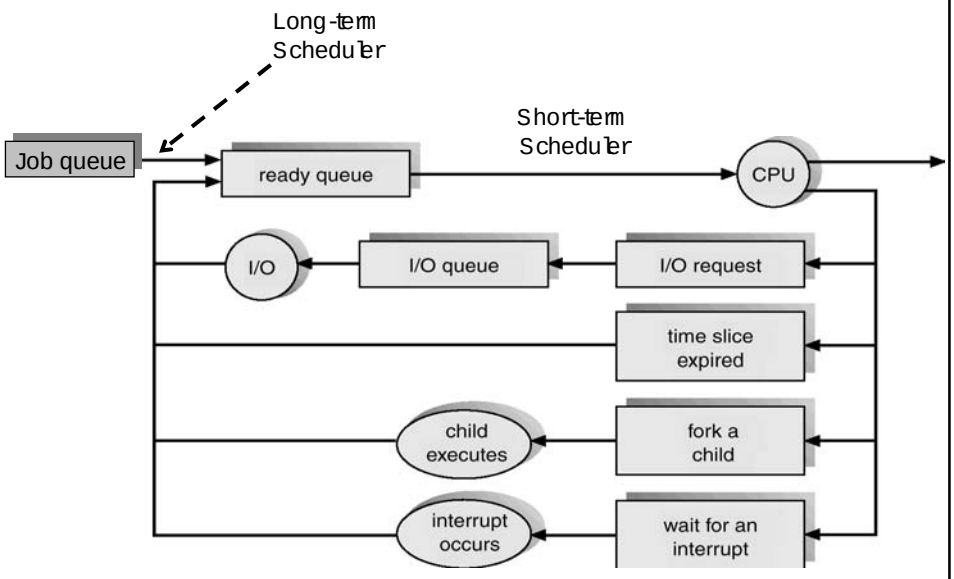
Các hàng đợi định thời (queue)

- Job queue (New): chứa các process mới được tạo ra trong hệ thống
- Ready queue (Ready): chứa các process đang nằm trong bộ nhớ chính sẵn sàng đợi được thực thi
- Device queues (Waiting): chứa các process đang chờ một thiết bị I/O, một sự kiện I/O
- Process được chuyển từ hàng đợi này sang hàng đợ khác trong suốt quá trình thực thi của nó
- Các hàng đợi định thời được hiện thực bằng danh sách liên kết (linked list)
  - Các liên kết là các con trỏ trong khối PCB

Hàng đợi Ready & I/O Device



Định thời Process



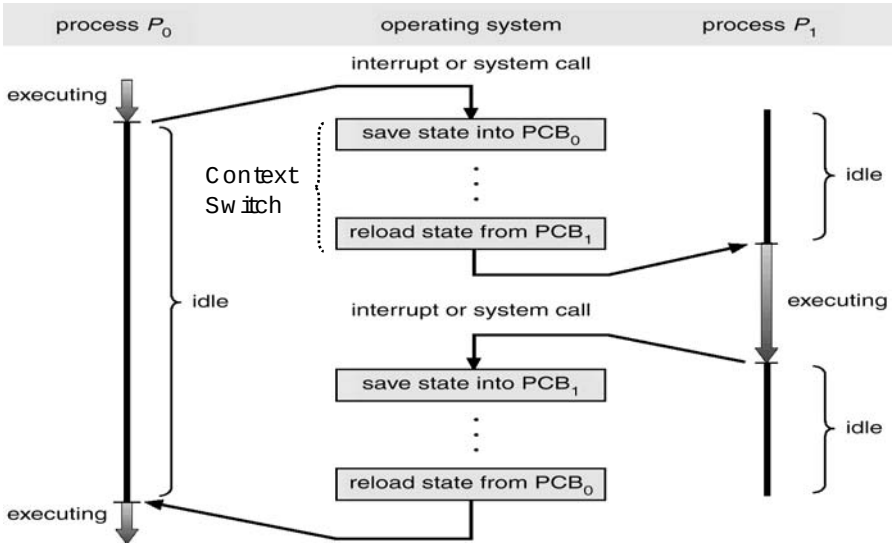
Các bộ định thời (schedulers)

- Long-term scheduler (or job scheduler)
  - Chọn process nào sẽ được đưa vào ready queue (từ New chuyển sang Ready)
- Short-term scheduler (or CPU scheduler)
  - Chọn process nào sẽ được chiếm CPU để xử lý (từ Ready chuyển sang Running)
- Medium-term scheduler
  - Chuyển process từ bộ nhớ chính sang sang bộ nhớ thứ cấp (nhưng vẫn nằm trong không gian bộ nhớ ảo); khi nào cần thì nạp process từ bộ nhớ thứ cấp vào bộ nhớ chính.

Các bộ định thời (t.t)

- Short-term schedule thường xảy ra rất thường xuyên (milli giây) → phải thực thi nhanh
- Long-term schedule thường thực hiện không thường xuyên (giây, phút) → có thể thực hiện chậm
  - Long-term scheduler điều khiển mức độ multi-programming
  - Nên chọn xen kẽ giữa I/O-bound và CPU- bound processes
- Thông thường, process chia làm 2 loại chính
  - I/O-bound process: phần lớn thời gian thực thi dùng để thực hiện các tác vụ I/O, thời gian chiếm CPU ít hơn.
  - CPU-bound process: thời gian thực thi chủ yếu là các tác vụ tính toán, chiếm CPU nhiều hơn so với thực hiện các tác vụ I/O.

Chuyển ngữ cảnh (context switch)



Chuyển ngữ cảnh (t.t)

- Khi CPU chuyển sang thực thi một process khác, hệ thống phải lưu trạng thái của process hiện tại đang thực thi và nạp trạng thái của process mới sẽ thực thi.
- Ngữ cảnh (context) của một process được biểu diễn trong khối PCB của process đó.
- Thời gian chuyển ngữ cảnh (context-switch time) là một trong các phí tổn (overhead) mà hệ thống phải gánh chịu; do đó, phải có chiến lược chuyển ngữ cảnh hợp lý để đạt hiệu quả xử lý cao.
- Chi phí chuyển ngữ cảnh phụ thuộc sự hỗ trợ cấp hardware, phụ thuộc phương thức quản lý bộ nhớ,…

Các bước chuyển ngữ cảnh

- Lưu ngữ cảnh CPU, bao gồm thanh ghi lệnh - program counter (PC) và các thanh ghi khác.
- Cập nhật PCB của process đang thực thi: ghi nhận trạng thái hiện tại và một số thông tin cần thiết khác
- Chuyển PCB của process đang thực thi đến hàng đợi tương ứng: ready, waiting
- Thực hiện việc chọn process khác để thực thi (short-term scheduler)
- Cập nhật PCB của process được chọn thực thi.
- Phục hồi ngữ cảnh CPU của process được chọn (nếu có)

Chuyển ngữ cảnh xảy ra khi nào?

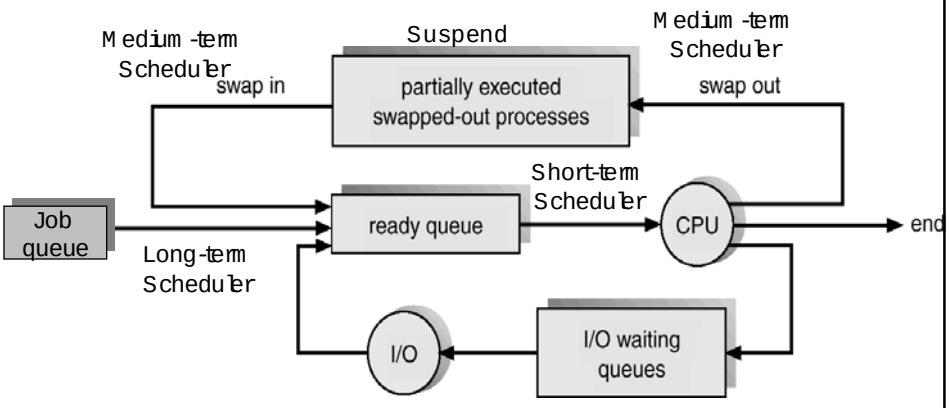
- Chuyển ngữ cảnh có thể xảy ra khi OS chiếm lại quyền điều khiển CPU, chẳng hạn như
  - System Call
    - » Được gọi tường minh trong chương trình (ví dụ: system call mở/đóng file). Process gọi system call có thể sẽ bị blocked chờ thực hiện system call.
  - Trap
    - » Một lỗi đã xảy ra. Process có thể chuyển vào trạng thái Exit và kết thúc thực thi.
  - Interrupt
    - » Quyền điều khiển chuyển sang cho Interrupt Handler.

Các ví dụ về Interrupt

- Clock:
  - process đã hết thời gian được chiếm CPU để thực thi (time slice, quantum) và phải chuyển sang trạng thái ready.
- I/O
  - Nếu có processes đang chờ sự kiện I/O này thì chuyển process đó sang trạng thái ready.
  - Sau đó, tiếp tục thực thi process hiện tại hoặc chọn một process khác có độ ưu tiên cao hơn.
- Memory fault
  - Địa chỉ bộ nhớ được tham chiếu nằm trong bộ nhớ ảo và phải được nạp vào bộ nhớ chính.
  - Process đang thực thi phải chuyển sang trạng thái blocked (chờ hoàn tất tác vụ I/O)

Medium Term Scheduling

OS có thể suspend một số process, nghĩa là chuyển các process đó ra bộ nhớ thứ cấp (đĩa cứng, mềm…)



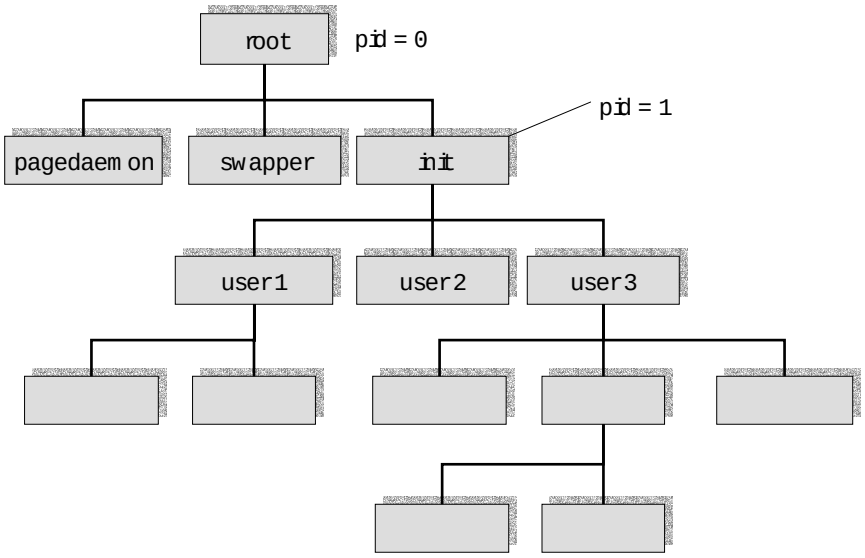
Tạo process (process creation)

- Process được tạo ra khi nào?
  - Có một công việc (job) mới yêu cầu được thực hiện.
  - Khi user đăng nhập (log on) → command interpreter (Unix shell)
  - Do OS tạo ra để cung cấp dịch vụ cho user (ví dụ:in một file)
  - Sinh ra bởi một process, ví dụ
    - » user program có thể tạo ra nhiều process
- Các bước OS khởi tạo process
  - Gán một định danh duy nhất (unique process identifier)
  - Cấp phát không gian nhớ cho process image.
  - Khởi tạo process control block (PCB)
    - » Một số giá trị mặc định (ví dụ: trạng thái=New, không có thiết bị I/O, không mở files...)
  - Thiết lập các mối liên hệ cần thiết
    - » Ví dụ: thêm process mới vào linked list của hàng đợi định thời

Mối quan hệ cha/con

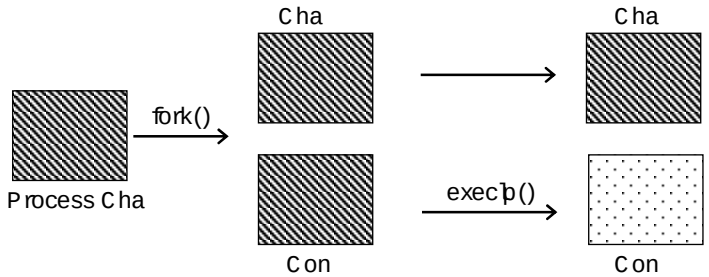
- Process cha tạo ra các process con, các process con tạo ra nhiều process khác,... và cứ như thế tạo thành một cây process trong hệ thống.
- 3 cách chia sẻ tài nguyên (resource sharing)
  - Process cha và con chia sẻ mọi tài nguyên
  - Process con chia sẻ một phần tài nguyên của cha
  - Process cha và con không chia sẻ tài nguyên nào
- Trình tự thực thi
  - Process cha và con thực thi đồng thời (concurrently)
  - Process cha đợi đến khi các process con kết thúc.

Cây process trong UNIX



Mối quan hệ cha/con (t.t)

- Không gian địa chỉ (address space)
  - Không gian địa chỉ của process con được nhân bản từ cha
  - Không gian địa chỉ của process con được nạp chương trình khác.
- Ví dụ trong UNIX/Linux
  - System call fork() tạo một process mới
  - System call execlp() dùng sau fork() để nạp một chương trình mới vào không gian nhớ của process mới



Ví dụ tạo process với fork()

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

   if ( pid < 0 )  {
        printf("Fork error\n");
        exit(-1);
   }
   else if ( pid == 0 )
   {
        printf("This is child process");
        execlp("/bin/ls", "ls", NULL);
        exit(0);
   }
   else  {
        printf("This is parent process");
        exit(0);
   }
}
```


Kết thúc thực thi process

- Kết thúc bình thường (normal completion)
- Vượt giới hạn thời gian (time limit exceeded)
- Không đủ bộ nhớ (memory unavailable)
- Xâm phạm vùng nhớ cấm (memory bounds violation)
- Lỗi bảo vệ (protection error)
  - Ví dụ: ghi vào file có thuộc tính read-only
- Lỗi số học (arithmetic error): chia cho 0, tràn số,...
- Time overrun
  - process chờ một sự kiện lâu hơn một khoảng thời gian tối đa được xác định trước

Kết thúc thực thi process (t.t)

- Thực hiện thất bại tác vụ I/O
- Lệnh không hợp lệ (invalid instruction)
- Privileged instruction
- Sự can thiệp của OS
  - Ví dụ: OS can thiệp khi có deadlock xảy ra
- Process cha yêu cầu kết thúc thực thi một process con
- Process cha kết thúc kéo theo các process con cũng kết thúc

Sự cộng tác giữa các process

- Một process thực thi độc lập thì không ảnh hưởng và không bị ảnh hưởng bởi các process khác trong hệ thống. Tuy nhiên, một số process có thể cộng tác, trao đổi dữ liệu với nhau để hoàn thành công việc.
- Ưu điểm của sự cộng tác
  - Chia sẻ thông tin
  - Hiệu suất tính toán cao
- Sự cộng tác của các process yêu cầu OS hỗ trợ cơ chế giao tiếp (communication) và cơ chế đồng bộ hoạt động của các process (synchronization)

Bài toán Producer-Consumer

- Mô hình cho sự cộng tác giữa các process, producer tạo ra các thông tin, dữ liệu và consumer tiêu thụ, sử dụng các dữ liệu đó. Sự trao đổi thông tin thực hiện qua buffer:
  - *unbounded-buffer:* kích thước buffer không giới hạn.
  - *bounded-buffer:* kích thước buffer có giới hạn.
- Producer và consumer phải được đồng bộ hoạt động
  - Consumer không thể sử dụng một dữ liệu mà producer chưa kịp tạo ra.
  - Producer không được tạo thêm sản phẩm khi buffer đã đầy (bounded buffer)
- Hiện thực buffer
  - Shared memory
  - Interprocess communication facility (IPC)

Ví dụ Shared Bounded-Buffer

- Shared memory
  - Shared buffer: danh sách xoay vòng.
    - » in: vị trí ghi kết tiếp
    - » out: vị trí đọc kế tiếp
    - » BUFFER_SIZE: kích thước của buffer (chỉ được dùng BUFFER_SIZE – 1 phần tử)

```
#define BUFFER_SIZE 10
typedef struct {
        ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Producer và consumer chia sẻ một buffer chung
- Programmer phải hiện thực đoạn mã truy xuất buffer chung cho producer và consumer

Shared Bounded-Buffer (t.t)

```
item nextProduced;

while (1){
    while (((in + 1)%
    BUFFER_SIZE)== out)
        ;/* do nothing */
    buffer[in]= nextProduced;
    in = (in + 1)%
        BUFFER_SIZE;
}
```

Producer Process

```
item nextConsumed;

while (1){
    while (in == out)
        ;/* do nothing */
    nextConsumed = buffer[out];
    out= (out+ 1)%
        BUFFER_SIZE;
}
```

Consumer Process

Interprocess Communication (IPC)

- IPC là cơ chế cung cấp bởi OS nhằm giúp các process giao tiếp và đồng bộ hoạt động.
- Shared Memory
  - threads
  - mmap()
- Message-passing
  - blocking (synchrous) và non-blocking
  - message format, buffer size, …
  - addressing (PID, port, mailbox, …)
- Files, pipes (FIFOs), sockets

Message-Passing IPC

- Cung cấp ít nhất hai phương tiện giao tiếp:
  - send(msg)
  - receive(msg)
- Quá trình giao tiếp cần tuân theo một giao thức định trước:
  - Thiết lập kênh giao tiếp
  - Trao đổi message

Hiện thực IPC ?

- Làm thế nào để thiết lập các kênh giao tiếp?
- Một kênh giao tiếp có thể liên kết với nhiều hơn 2 process hay không?
- Có thể tạo bao nhiêu kênh giao tiếp giữa các cặp process muốn thực hiện giao tiếp?
- Dung lượng của một kênh giao tiếp là bao nhiêu?
- Kích thước message truyền qua kênh giao tiếp là cố định (fixed) hay có tùy ý ?
- Kênh giao tiếp theo hai chiều hay một chiều?

Đồng bộ (synchronization)

- Blocking hay non-blocking: synchronous hay asynchronous
- Blocking send: sender bị block cho đến khi message đã được nhận bởi receiver hoặc message đã đến mailbox.
- Non-blocking send: sender gửi message và tiếp tục thực hiện công việc khác.
- Blocking receive: receiver sẽ bị block đến khi có một message được gửi đến.
- Non-blocking receive: không bị block, tuy nhiên receiver hoặc là nhận được một message hoặc không nhận được gì cả.
- Rendezvous: blocking-send và blocking-receive

Bộ đệm (buffering)

- Tương ứng với mỗi kênh giao tiếp có một hàng đợi message. Hiện thực của hàng đợi message là một bộ đệm (buffer) với một trong ba kiểu sau
  - Bộ đệm dung lượng bằng 0 (zero capacity), nghĩa là giữa sender và receiver không có hàng đợi, sender phải đợi receiver.
  - Bộ đệm có dung lượng hạn chế (bounded capacity): kích thước bộ đệm là một số hữu hạn định trước. Nếu bộ đệm bị đầy thì sender phải đợi.
  - Bộ đệm có dung lượng không hạn chế (unbounded capacity), sender không bao giờ phải đợi

Mô hình giao tiếp Client-Server

- Sockets
- Remote Procedure Calls (RPC)
- Remote Method Invocation (RMI)

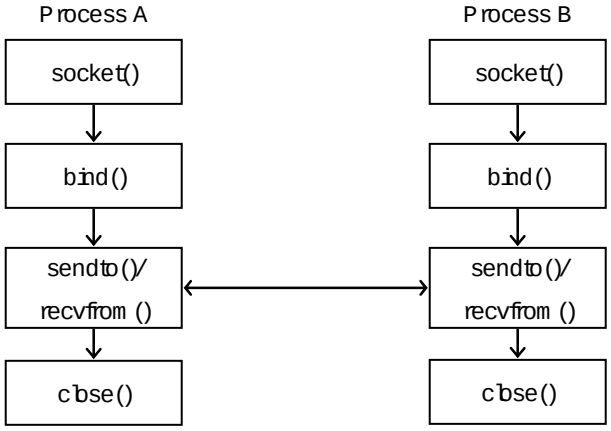
Socket

- Socket: đầu cuối (endpoint) của một kênh giao tiếp
  - Bao gồm địa chỉ IP và số hiệu port
    - » Ví dụ: socket 172.28.11.120:1234 tham chiếu đến port 1234 trên máy có địa chỉ IP là 172.28.11.120
  - Ví dụ: FTP server lắng nghe ở port 21, telnet ở port 23, web server tại port 80
  - Quá trình giao tiếp được thực hiện qua một cặp socket.
- Socket là cơ chế giao tiếp mức thấp (low-level)
  - Gửi nhận một chuỗi byte dữ liệu không có cấu trúc (unstructured)
- Có hai cơ chế giao tiếp qua socket
  - Connectionless
  - Connection-oriented
- Berkeley socket (BSD socket): trên Unix/Linux
- WinSock: thư viện socket trên họ MS Windows

Cơ chế gửi/nhận qua Socket

Hàm thư viện	*Diễn giải*
socket()	Tạo một socket
bind()	Gắn một địa chỉ cục bộ vào một socket
listen()	Xác định độ lớn/kích thước hàng đợi
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
recv()/recvfrom()	Nhận dữ liệu qua kênh giao tiếp
close()	Đóng kết nối

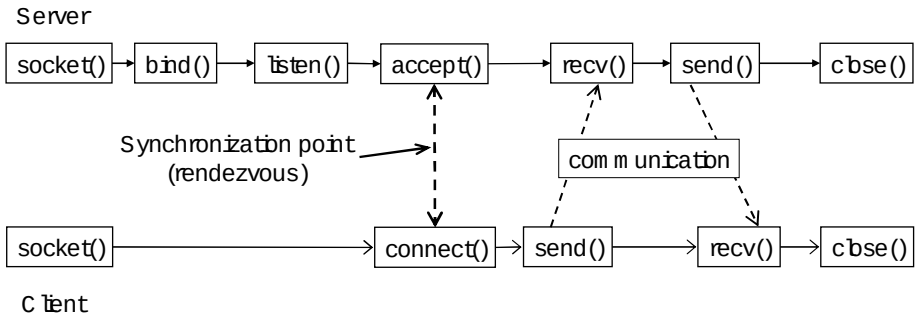
Connectionless Socket



» sendto(socket, buffer, buffer_length, flags, destination_address, addr_len)

» recvfrom (socket, buffer, buffer_length, flags, from _address, addr_len)

Connection-Oriented Client/Server



» send(socket, buffer, buffer_length, flags)

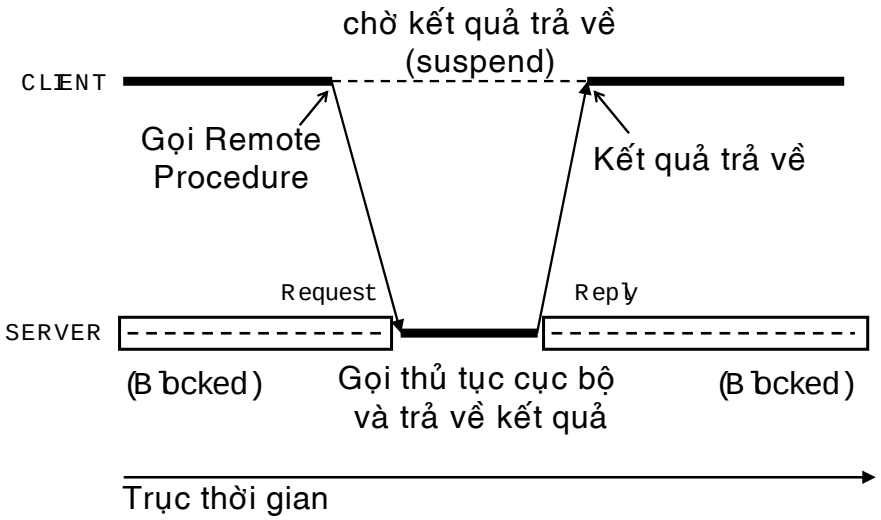
» recv(socket, buffer, buffer_length, flags)

Remote Procedure Call (RPC)

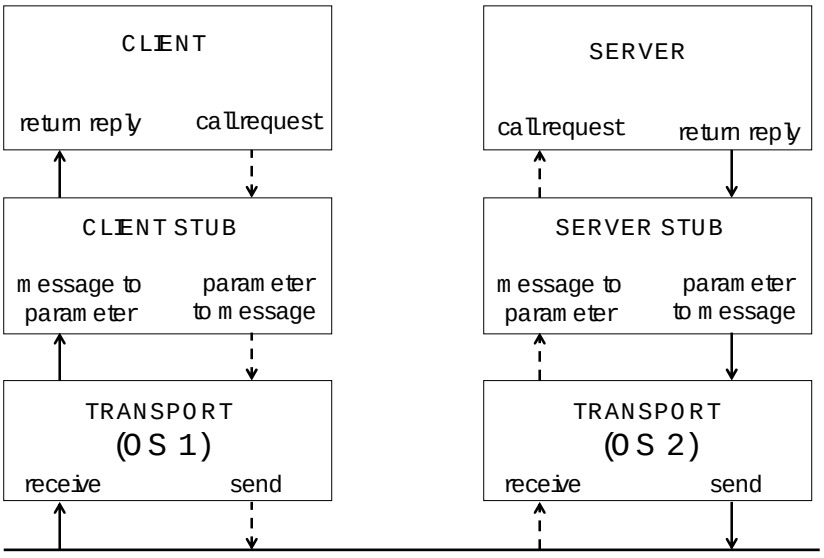
- Cơ chế RPC cho phép thực hiện tác vụ gọi thủ tục (procedure call) giữa các process nằm trên hai máy tính kết nối bằng hệ thống mạng.

- Các vấn đề khi hiện thực RPC
  - Truyền tham số và kết quả trả về của lời gọi thủ tục
  - Chuyển đổi dữ liệu khi truyền trên mạng (data conversion)
  - Kết nối client đến server
  - Biên dịch chương trình
  - Kiểm soát lỗi
  - Security

Nguyên lý của RPC



Lưu đồ thực hiện RPC

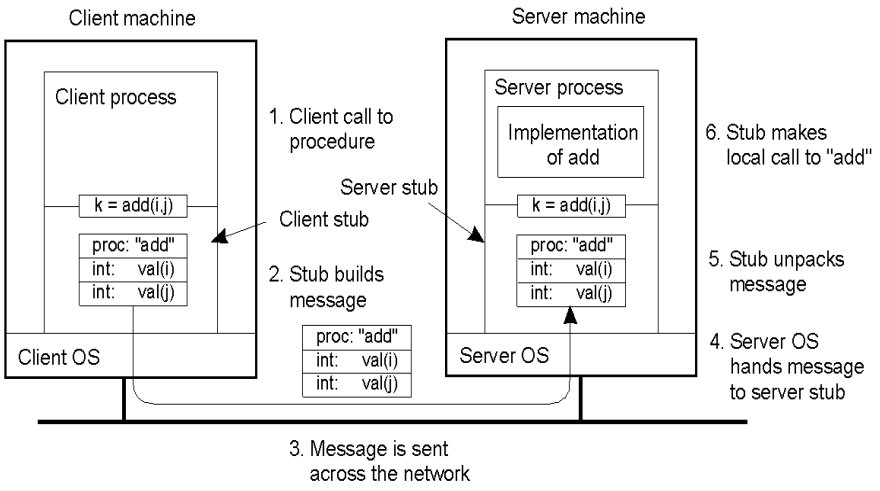


Truyền tham số trong RPC

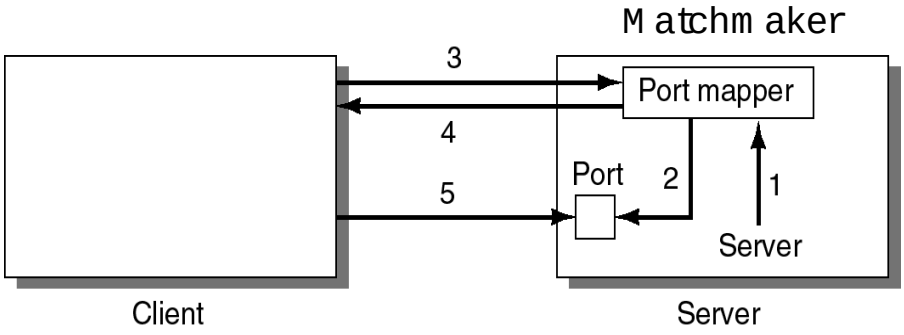
- Marshaling: qui tắc truyền tham số và chuyển đổi dữ liệu trong RPC bao gồm cả đóng gói dữ liệu thành dạng thức có thể truyền qua mạng máy tính.

- Biểu diễn dữ liệu và kiểm tra kiểu dữ liệu (type checking)
  - Dữ liệu biểu diễn khác nhau trên các hệ thống khác nhau
    - » ASCII, EBCDIC
    - » Little Endian và Big Endian
    - » Dạng biểu diễn XDR (External data representation): độc lập với hệ thống bên dưới (machine-independent)

Truyền tham số trong RPC (t.t)

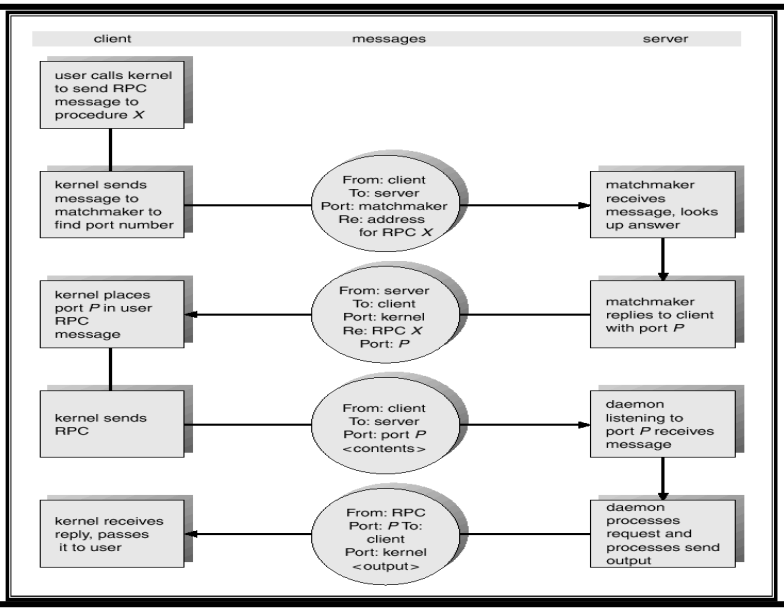


Kết nối (bind) client và server



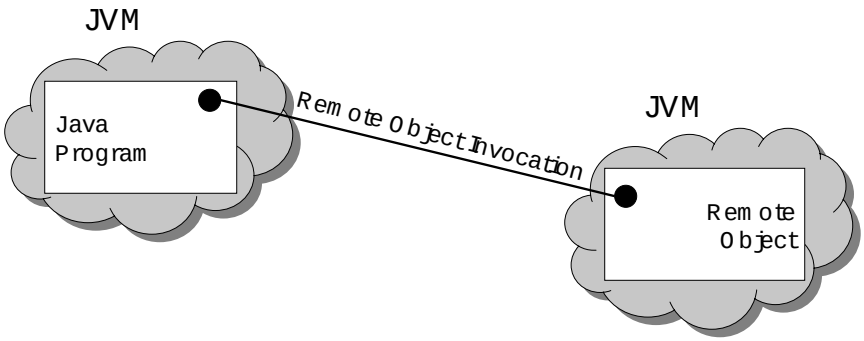
1. I need a port
2. Here is your port
3. I need a handle
5. Communicate using handle

Quá trình thực hiện RPC



Remote Method Invocation

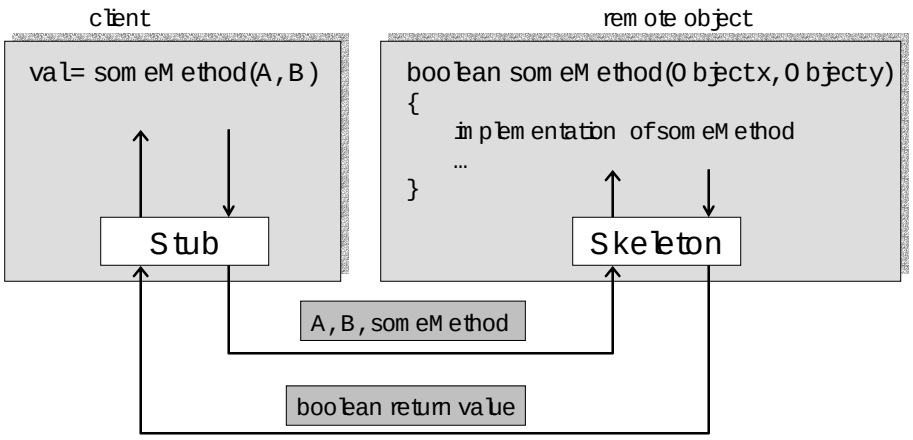
- Remote Method Invocation (RMI) là cơ chế tương tự RPC nhưng vận hành trên nền máy ảo Java.
- RMI cho phép một chương trình Java có thể triệu gọi một phương thức (method) của một đối tượng ở xa.



Remote Method Invocation (t.t)

- Khác biệt giữa RPC và RMI
  - RPC hỗ trợ lập trình cấu trúc (procedural programming)
    - » Chỉ cho phép triệu gọi thủ tục (procedure) hoặc hàm (functions) ở xa.
  - RMI hỗ trợ cơ chế hướng đối tượng (object-oriented)
    - » RMI cho phép triệu gọi phương thức (method) trên đối tượng ở xa (remote object)
  - Trong RPC, tham số truyền đến thủ tục/hàm ở xa là dữ liệu bình thường (số, chuỗi kí tự, dãy,...)
  - Trong RMI, tham số truyền đến đối tượng ở xa có thể là một đối tượng

Cơ chế marshalling trong RMI

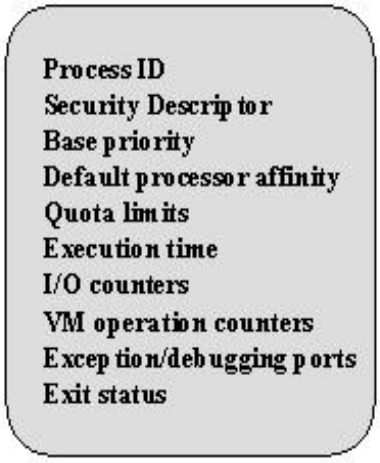


Phương thức được triệu gọi có dạng sau:

boolean someMethod(Object x, Object y)

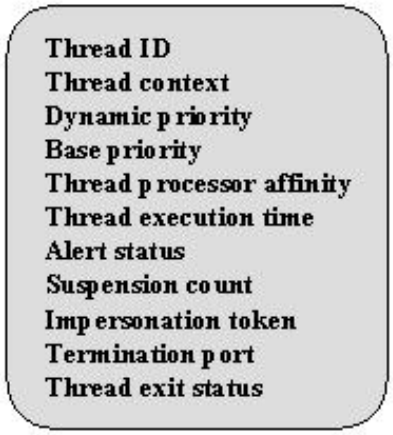
Windows NT Process Object

- Process ID
- Security Descriptor
- Base priority
- Default processor affinity
- Quota limits
- Execution time
- I/O counters
- VM operation counters
- Exception/debugging ports
- Exit Status

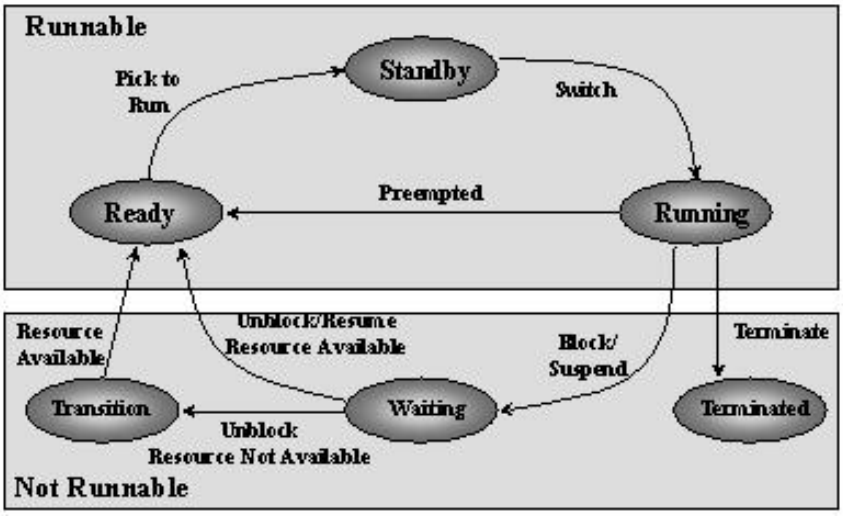


Windows NT Thread Object

- Thread ID
- Thread context
- Dynamic priority
- Base priority
- Thread processor affinity
- Thread execution time
- Alert status
- Suspension count
- Impersonation token
- Termination port
- Thread exit status



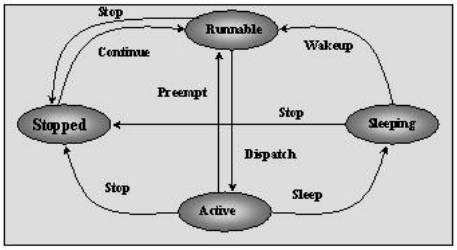
NT Thread Status



Solaris

- Process includes the user's address space, stack, and process control block
- User-level threads
- Lightweight processes
- Kernel threads

Solaris User Level Threads



Solaris Light weight Processes