Chương 3. Tiến Trình (Process)

Mục Tiêu

Giới thiệu các **khái niệm về Tiến trình** và những thao tác cơ bản trong quản lý Tiến trình như tạo, định thời và kết thúc tiến trình. Các phương thức **giao tiếp liên tiến trình** cũng sẽ được trình bày.

Nội Dung

Các khái niệm cơ bản

Định thời cho Tiến trình (Process Scheduling)

Các thao tác trên Tiến trình

Hợp tác Tiến trình (Cooperating process)

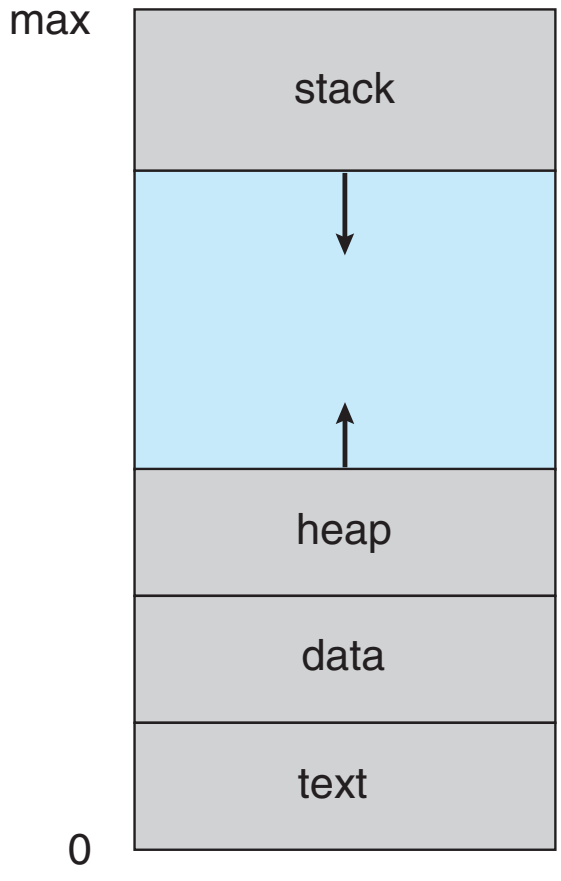
Giao tiếp trong hệ thống Client–Server

Khái Niệm Tiến Trình

- Tiến trình là thể hiện (instance) của một chương trình máy tính trong bộ nhớ, đang thực thi hoặc chờ thực thi.
- Mỗi tiến trình thường được gán 1 số định danh tiến trình (process identifier, pid), dùng để định danh các tiến trình.
- Một tiến trình bao gồm:
  - Mã lệnh chương trình (program code)
  - Bộ đếm chương trình (program counter) và các thanh ghi của CPU
  - Ngăn xếp (stack)
  - Phần dữ liệu (data section)
  - Có thể gồm phần bộ nhớ cấp phát động khi tiến trình thực thi (heap)

Chương Trình & Tiến Trình

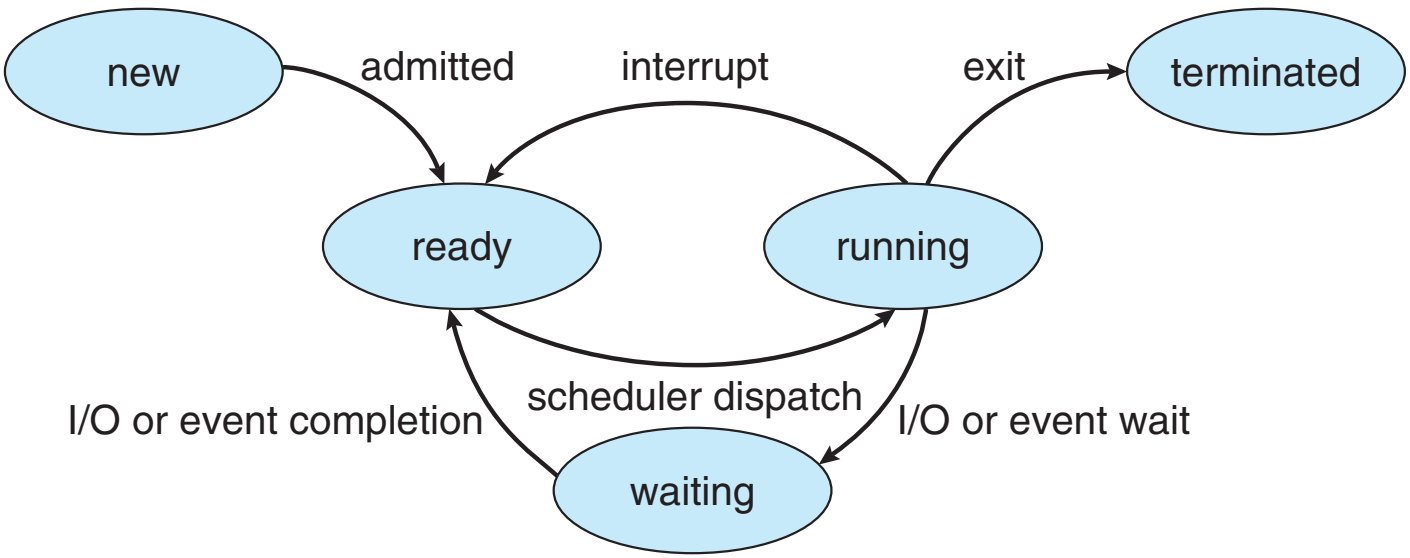
- Chương trình là một thực thể bị động, được lưu trữ trên đĩa.
- Tiến trình là một thực thể chủ động, lưu trú trên bộ nhớ chính.
- Khi một chương trình được kích hoạt (nhấp chuột, CLI, ...), một thể hiện của chương trình sẽ được nạp lên bộ nhớ, tạo ra 1 tiến trình.
- Một chương trình có thể có vài tiến trình trong bộ nhớ.



Trạng Thái Của Tiến Trình (Process State)

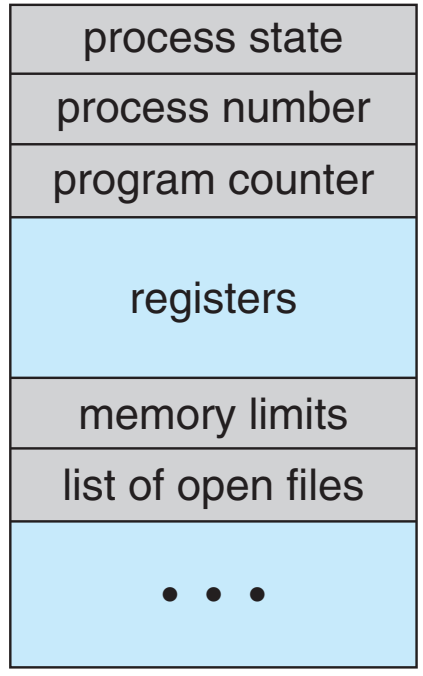
- Một tiến trình có thể có một trong các trạng thái sau:
  - new: tiến trình đang được khởi tạo.
  - running: các chỉ thị của tiến trình đang được thực thi.
  - waiting: tiến trình đang chờ đợi một sự kiện nào đó xảy ra (hoàn thành I/O, tín hiệu từ tiến trình khác, ...).
  - ready: tiến trình sẵn sàng để thực thi (đang đợi để được sử dụng CPU).
  - terminated: tiến trình đã kết thúc.

Sơ Đồ Chuyển Trạng Thái Của Tiến Trình



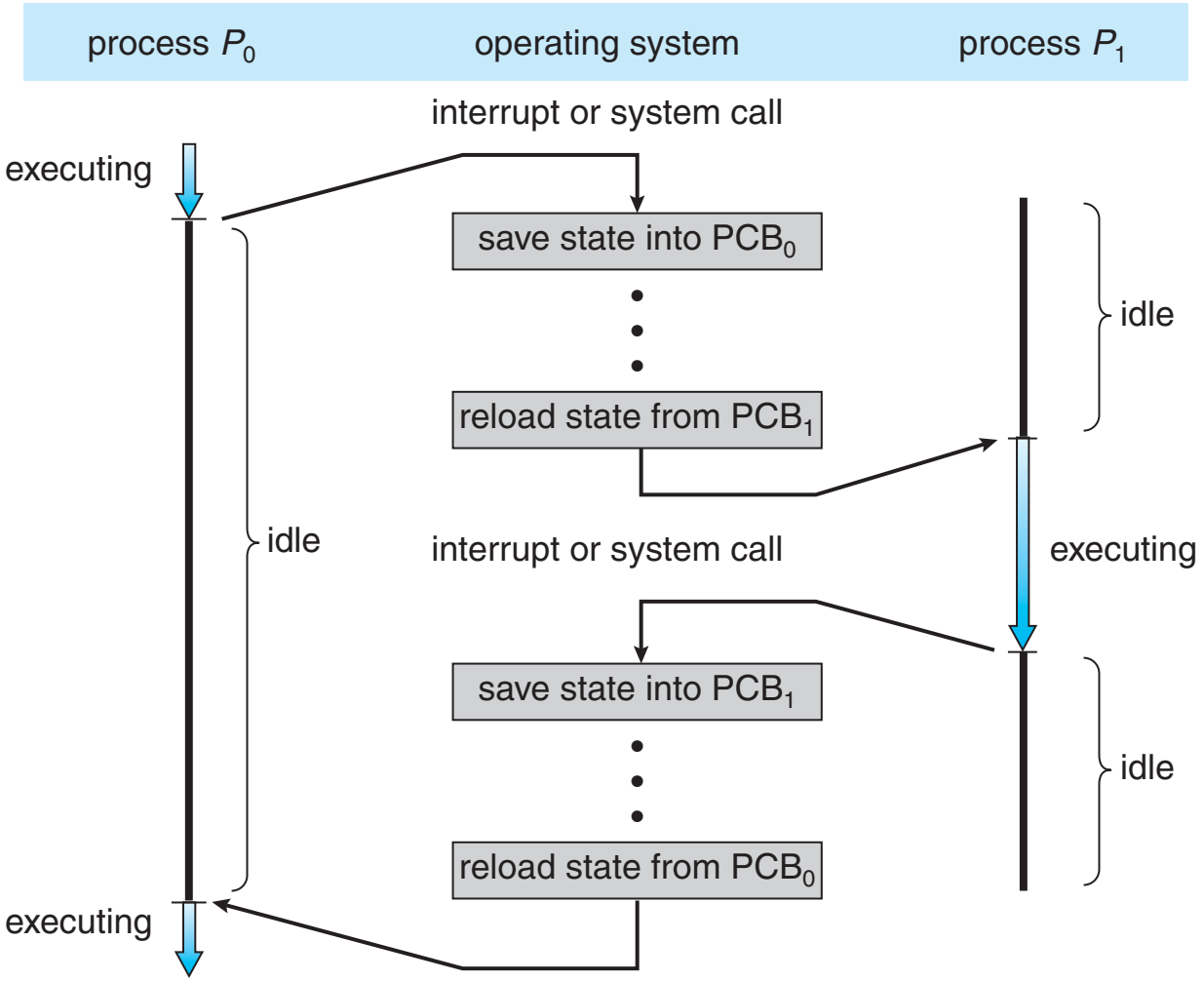
Khối Điều Khiển Tiến Trình (PCB)

- Chứa thông tin của tiến trình trong Hệ điều hành:
  - Trạng thái của quá trình: ready, running, ...
  - Bộ đếm chương trình: chỉ thị kế tiếp sẽ được thực thi
  - Các thanh ghi: phụ thuộc vào k/trúc máy tính
  - Thông tin về định thời sử dụng CPU
  - Thông tin về quản lý bộ nhớ
  - Thông tin về chi phí: t/gian sử dụng CPU, pid, ...
  - Thông tin về trạng thái nhập/xuất: các thiết bị đang được cấp phát, danh sách tập tin đang mở, ...



Chuyển CPU Giữa Các Tiến Trình

- PCB là nơi lưu giữ trạng thái của tiến trình
- Trạng thái của tiến trình phải được lưu trữ vào PCB khi một interrupt xuất hiện, nhằm cho phép tiến trình có thể tiếp tục chính xác về sau.
- Tác vụ chuyển CPU còn được gọi là chuyển ngữ cảnh (context switch).



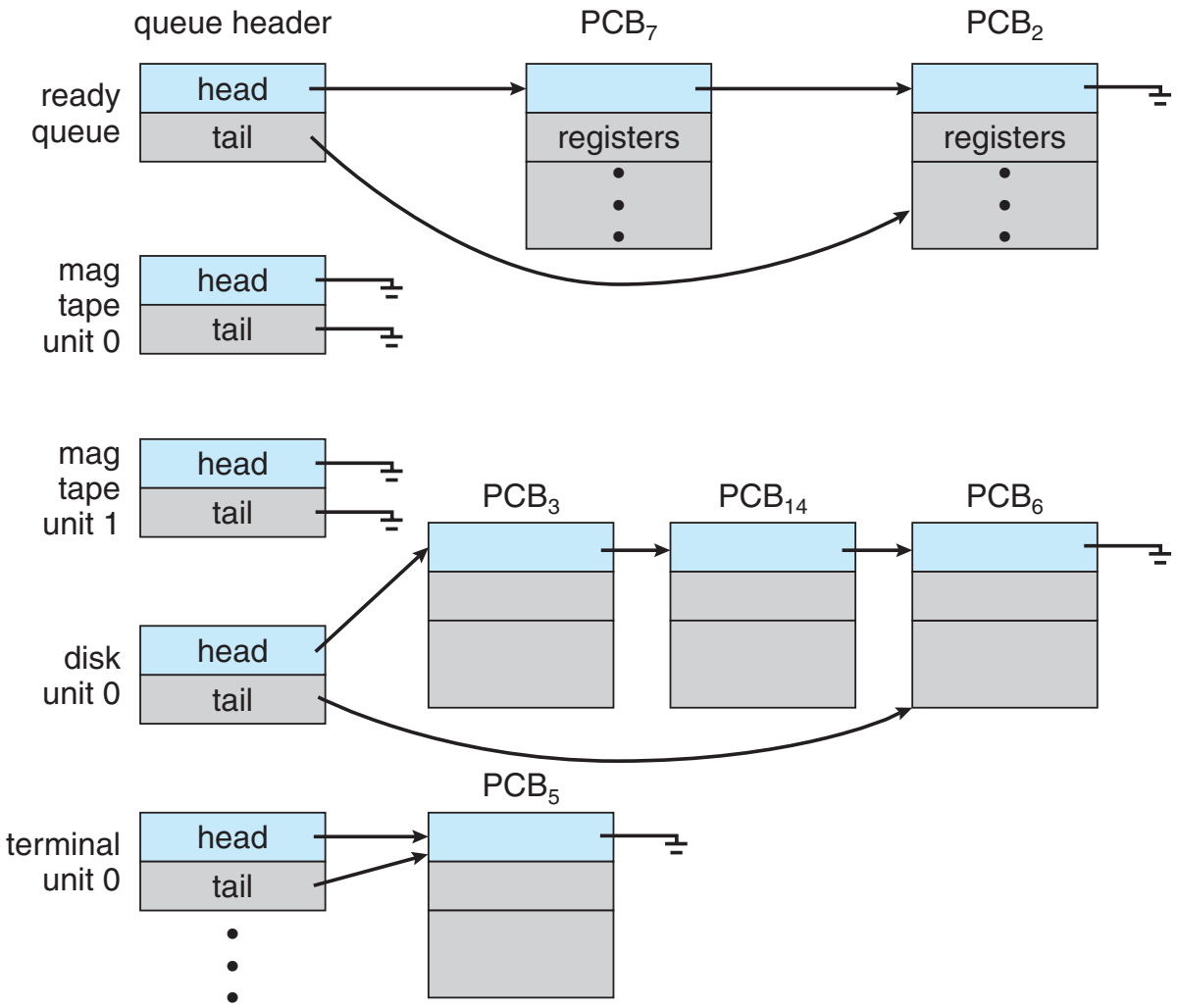
Định Thời Tiến trình (Process Scheduling)

- Là một tác vụ của hệ điều hành trong các hệ thống đa chương dựa trên phân chia thời gian (time-sharing) nhằm lựa chọn một tiến trình được phép sử dụng CPU và phân bổ thời gian sử dụng CPU của tiến trình.
- Thành phần lựa chọn/định thời cho các tiến trình được gọi là bộ định thời tiến trình (process scheduler).
- Bộ định thời tiến trình dùng 1 hệ thống các hàng đợi (queue) để sắp xếp và định thời cho các tiến trình.

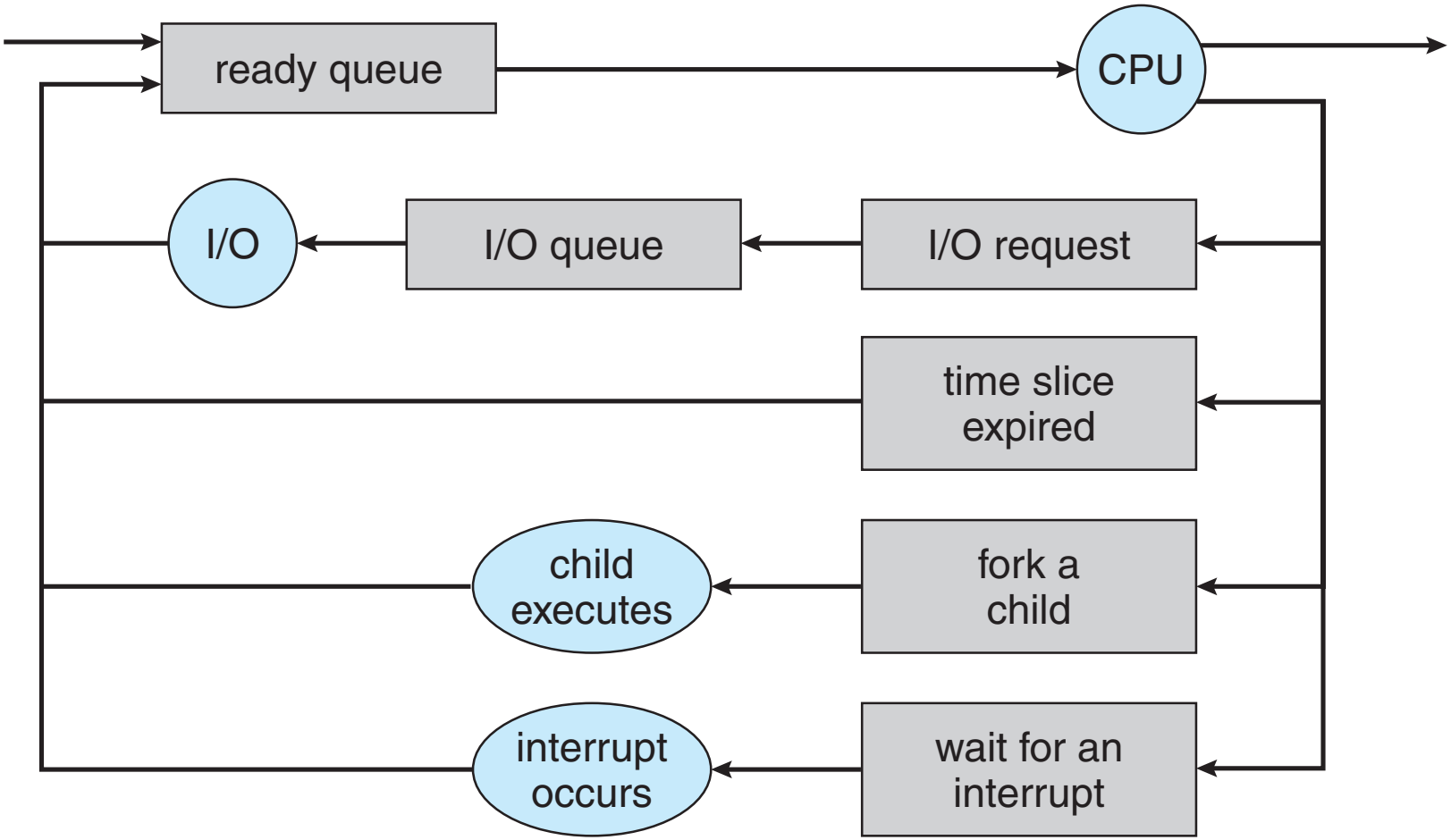
Hàng Đợi Tiến Trình (Process Queues)

- Các hàng đợi dùng cho việc định thời tiến trình:
  - Hàng đợi công việc (job queue): tập hợp tất cả các tiến trình trong hệ thống.
  - Hàng đợi sẵn sàng (ready queue): tập hợp tất cả các tiến trình đang nằm trong bộ nhớ, sẵn sàng và đang chờ để thực thi.
  - Hàng đợi thiết bị (device queue): tập hợp các tiến trình đang đợi sử dụng một thiết bị vào ra.
- Tiến trình có thể di chuyển giữa các hàng đợi khác nhau.

Hàng Đợi Sẵn Sàng & Hàng Đợi Thiết Bị



Sơ Đồ Định Thời Tiến Trình

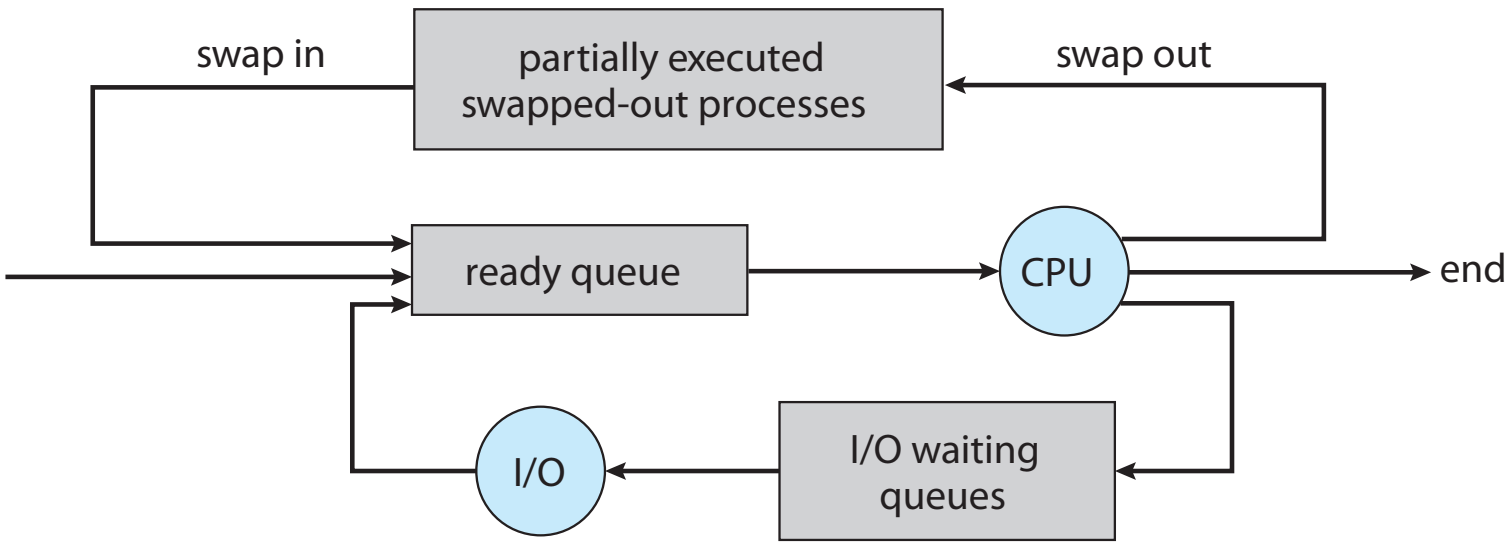


Các Loại Bộ Định Thời (Shcedulers)

- Bộ định thời dài kỳ (long-term scheduler/job scheduler):
  - chọn tiến trình nào sẽ được đặt vào hàng đợi sẵn sàng (nạp vào bộ nhớ)
  - được gọi rất không thường xuyên (seconds, minutes) $\Rightarrow$ có thể chậm
  - khống chế cấp độ đa chương (degree of multiprogramming)
- Bộ định thời ngắn kỳ (short-term scheduler/CPU scheduler):
  - chọn ra tiến trình sẽ được thực thi kế tiếp và cấp CPU cho nó.
  - được gọi rất thường xuyên (milliseconds) $\Rightarrow$ phải nhanh

Bộ Định Thời Trung Kỳ (Medium-term)

- là mức trung gian giữa bộ định thời ngắn và dài kỳ
- thực hiện hoán vị (swapping) các tiến trình ra/vào bộ nhớ/đĩa do cạnh tranh CPU, bộ nhớ
- thường được sử dụng trong các hệ thống phân chia thời gian.

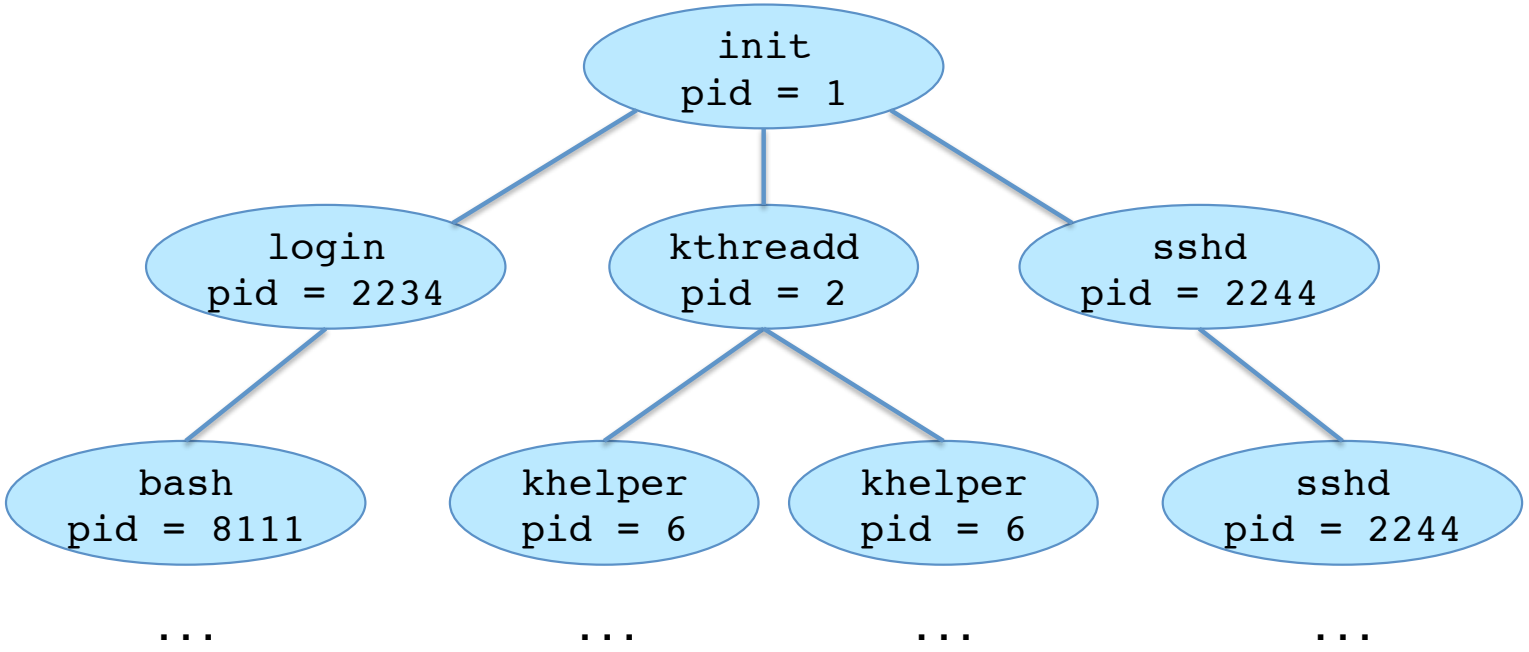


Các Thao Tác Cơ Bản Trên Tiến Trình

- Có 2 thao tác cơ bản trên tiến trình:
  - Tạo tiến trình
  - Kết thúc tiến trình

Tạo Tiến Trình

- Một tiến trình (cha) có thể tạo những tiến trình khác (con) . . .
- Quan hệ "cha–con" của các tiến trình tạo nên cây tiến trình.



Một Số Vấn Đề Giữa Tiến Trình Cha – Con

- **Chia sẻ tài nguyên:**
  - Tiến trình cha và con chia sẻ tất cả các tài nguyên
  - Tiến trình cha chia sẻ một phần tài nguyên cho tiến trình con
  - Tiến trình cha và con không chia sẻ gì cả
- **Dữ liệu khởi tạo**: được chuyển từ tiến trình cha sang con.
- **Thực thi**: song song hoặc tuần tự theo thứ tự cha – con.
- **Không gian địa chỉ**:
  - Tiến trình con sao chép từ tiến trình cha (cả code và dữ liệu)
  - Tiến trình con tự nạp chương trình riêng.

Tạo Tiến Trình Trên UNIX & Windows NT

- **UNIX**:
  - `fork()`: lời gọi hệ thống để tạo tiến trình mới.
  - `execlp()`: thay thế không gian địa chỉ của tiến trình gọi bằng một tiến trình mới.
- **Windows NT**:
  - `CreateProcess(...)`: lời gọi hệ thống để tạo 1 tiến trình con và thay thế không gian địa chỉ tiến trình con bằng 1 tiến trình mới.
  - Tiến trình mới được chỉ định trong đối số của lời gọi hệ thống.

Ví Dụ Tạo Tiến Trình Trên UNIX

```
#include <stdio.h>
#include <unistd.h>
int main() {
  int pid;   /* fork  another process */
  pid = fork();
  if (pid < 0) {   /* error occurred */
    fprintf(stderr, “Fork failed”);
    exit(-1);
  }
  else if (pid ==0) {   /* child process */
    execlp(“/bin/ls”, “ls”, NULL);
  }
  else {  /*parent waits for the child to complete*/
    wait(NULL);
    printf(“Child completed”);
    exit (0);
  }
}
```

Kết Thúc Tiến Trình

- T/trình thực thi câu lệnh cuối cùng và yêu cầu HĐH xóa nó (`exit()`)
  - Truyền dữ liệu từ tiến trình con lên tiến trình cha (`wait()`).
  - Thu hồi tài nguyên đã được cấp phát cho tiến trình.
- Tiến trình con kết thúc trước khi t/trình cha gọi `wait()` $\Rightarrow$ zombie
- Tiến trình con còn thực thi khi t/trình cha đã kết thúc $\Rightarrow$ orphan
- Tiến trình cha có thể kết thúc tiến trình con (`abort()`):
  - Tiến trình con đã có vượt quá số tài nguyên được cấp.
  - Công việc giao cho tiến trình con làm nay không còn cần thiết nữa.
  - Tiến trình cha đang thoát: một vài HĐH không cho phép orphan.

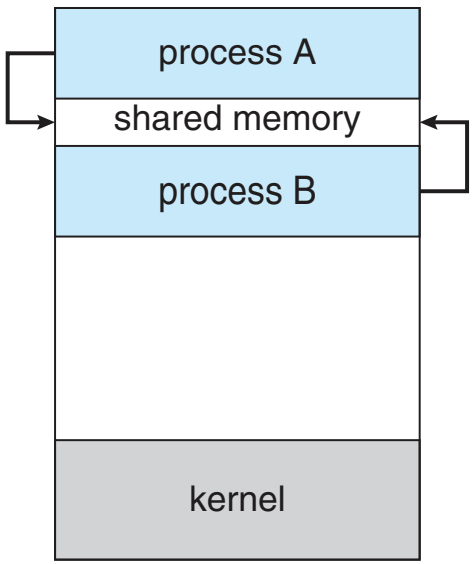
Hợp Tác Tiến Trình (Cooperating Process)

- Tiến trình độc lập: không thể ảnh hưởng hoặc không bị ảnh hưởng bởi sự thực thi của quá trình khác.
- Hợp tác tiến trình: có thể ảnh hưởng hoặc bị ảnh hưởng bởi sự thực thi của quá trình khác.
- Thuận lợi của sự hợp tác quá trình:
  - Chia sẻ thông tin
  - Gia tăng tốc độ tính toán (nếu máy có nhiều CPU)
  - Module hóa
  - Tiện dụng

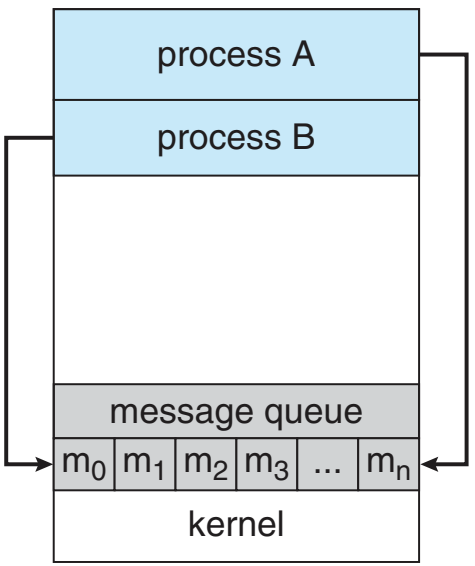
Giao Tiếp Liên Tiến Trình

- Các tiến trình muốn trao đổi dữ liệu với nhau cần sử dụng cơ chế giao tiếp liên tiến trình (interprocess communication, IPC):

bộ nhớ chia sẻ



truyền thông điệp



Bộ Nhớ Chia Sẻ (Shared-Memory)

- Một vùng đệm (buffer) được dùng để chia sẻ dữ liệu giữa các t/trình:
  - kích thước không giới hạn (unbounded buffer): tiến trình đọc có thể chờ, tiến trình ghi không bao giờ chờ.
  - kích thước có giới hạn (bounded buffer): cả tiến trình đọc và ghi có thể chờ.
- Ví dụ kinh điển "Nhà sản xuất – Người tiêu thụ": tiến trình Nhà sản xuất sinh dữ liệu, được sử dụng bởi tiến trình Người tiêu thụ.
  - Tạo 1 vùng nhớ đệm (buffer) chung.
  - Tiến trình Nhà sản xuất ghi dữ liệu lên buffer.
  - Tiến trình Người tiêu thụ lấy dữ liệu từ buffer.

Tạo Vùng Đệm (Buffering)

```
#define BUFFER_SIZE 10

typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];

int in_item = 0;
int out_item = 0;
```

Nhà Sản Xuất (Producer)

```
item next_produced;

while (true) {
  /* produce an item in next produced */

  while (((in_item + 1) % BUFFER_SIZE) == out_item)
     ; /* do nothing */

  buffer[in_item] = next_produced;
  in_item = (in_item + 1) % BUFFER_SIZE;
}
```

Người Tiêu Dùng (Consumer)

```
item next_consumed;

while(true){
  while(in_item == out_item)
    ; /* do nothing */

  next_consumed = buffer[out_item];
  out_item = (out_item + 1) % BUFFER_SIZE;

  /* consume the item in next consumed */
}
```

Truyền Thông Điệp (Message Passing)

- Giao tiếp giữa các tiến trình không cần dùng bộ nhớ chia sẻ
  $\Rightarrow$ hữu ích trong môi trường phân tán, giao tiếp qua mạng.
- Cần hai thao tác: `send(msg)` và `receive(msg)`.
- Tiến trình P và Q muốn giao tiếp với nhau:
  - Tạo một nối kết giao tiếp (communication link)
  - Trao đổi thông điệp thông qua `send`/`receive`
- Phương thức cài đặt nối kết giao tiếp (mức luận lý):
  - Giao tiếp trực tiếp hay gián tiếp
  - Đồng bộ hay bất đồng bộ
  - Kích thước thông điệp cố định hay biến đổi

Giao Tiếp Trực Tiếp (Direct Communication)

- Các quá trình phải được đặt tên rõ ràng:
  - `Send(P, msg)`: gởi thông điệp tới quá trình P.
  - `Receive(Q, msg)`: nhận thông điệp từ quá trình Q.
- Các thuộc tính của nối kết giao tiếp:
  - Các nối kết được thiết lập tự động.
  - Một nối kết kết hợp với chính xác một cặp quá trình.
  - Giữa mỗi cặp quá trình tồn tại chính xác một nối kết.
  - Nối kết có thể một hướng, nhưng thường là hai hướng.
  - Giao tiếp bất đối xứng: `Send(P, msg)`, `Receive(id, msg)`.

Giao Tiếp Gián Tiếp (Indirect Communication)

- Các thông điệp được gửi và nhận thông qua mailbox hay port.
  - Mỗi mailbox có một định danh (id) duy nhất.
  - Các quá trình chỉ có thể giao tiếp nếu chúng dùng chung mailbox.
    - `Send/Receive(A, msg)`: gởi/nhận thông điệp tới/từ hộp thư A.
- Các thuộc tính của nối kết gián tiếp:
  - Nối kết chỉ được thiết lập nếu các quá trình chia sẻ một hộp thư chung.
  - Một nối kết có thể kết hợp với nhiều quá trình.
  - Mỗi cặp quá trình có thể dùng chung nhiều nối kết giao tiếp.
  - Nối kết có thể một hướng hay hai hướng.

Các Tác Vụ Trong Giao Tiếp Gián Tiếp

- **Các tác vụ cơ bản**: tạo mailbox mới, gửi và nhận thông điệp qua mailbox, và xóa mailbox.
- **Chia sẻ mailbox**:
  - Các tiến trình có thể chia sẻ cùng 1 mailbox.
  - Vấn đề: nếu 1 tiến trình gửi thì tiến trình nào sẽ nhận?
- **Giải pháp cho việc chia sẻ mailbox**:
  - Một liên kết chỉ tương ứng với 2 tiến trình.
  - Chỉ cho phép 1 tiến trình thực hiện thao tác nhận tại 1 thời điểm.
  - HĐH chỉ định tiến trình nhận (1 tiến trình), và thông báo cho tiến trình gửi biết người nhận.

Đồng Bộ Hóa (Synchronisation)

- Truyền thông điệp có thể chặn (blocking) hay không chặn (non-blocking).
- Blocking được xem là đồng bộ (synchronous):
  - Blocking send: tiến trình gửi chờ cho đến khi thông điệp được nhận.
  - Blocking receive : tiến trình nhận chờ cho đến khi thông điệp sẵn sàng .
- Non-blocking được xem là bất đồng bộ (asynchronous):
  - Non-blocking send: gửi thông điệp và tiếp tục thực hiện công việc khác.
  - Non-blocking receive: nhận một thông điệp hay rỗng.

Tạo Vùng Đệm (Buffering)

- Vùng đệm dùng để chứa các thông điệp của 1 nối kết.
- Ba cách cài đặt:
  - Sức chứa là 0 (zero capacity): tiến trình gửi bị chặn đến khi thông điệp được nhận (no buffering!?).
  - Sức chứa giới hạn (bounded capacity): kích thước vùng đệm giới hạn n thông điệp. Tiến trình gửi bị chặn khi vùng đệm bị đầy.
  - Sức chứa không giới hạn (unbounded capacity): kích thước không giới hạn. Tiến trình gửi không bao giờ bị chặn.

Giao Tiếp Trong Hệ Thống Client–Server

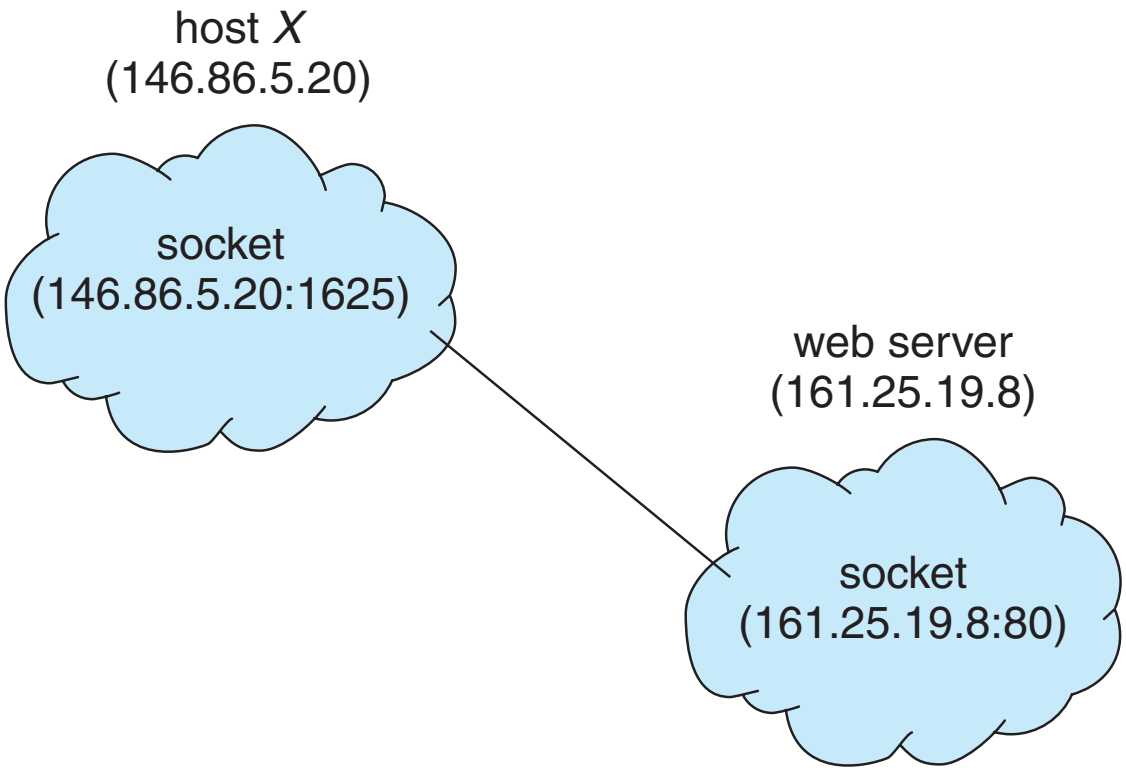
- Có 3 phương pháp cơ bản thường được dùng để giao tiếp trong mô hình client–server:
  - **Socket**
  - *Remote Procedure Calls* (*RPCs*)
  - *Pipe* (*ống dẫn*)

- Các phương pháp chia sẻ bộ nhớ và truyền thông điệp cũng có thể dùng cho giao tiếp client–server.

Socket

- Socket là một điểm nút của giao tiếp (endpoint of communication) của hai tiến trình.
- Hình thành từ 1 địa chỉ IP và 1 số hiệu cổng (port, 0–65535).
  - Socket **192.168.10.20:8080** là socket có cổng 8080 trên máy 192.168.10.20
- Mỗi cặp tiến trình giao tiếp dùng 1 cặp socket – 1 client socket và 1 server socket.
- Số hiệu cổng của client socket được gán tự động khi tạo ra, còn số hiệu cổng của server socket phải khai báo tường minh.
- Các tiến trình trong cùng hệ thống không được dùng port trùng nhau.

Mô Hình Giao Tiếp Dùng Socket



Date Server

```
public static void main(String[] args) {
  try {
    ServerSocket sock = new ServerSocket(6013);
    /* now listen for connections */
    while (true) {
       Socket client = sock.accept();
       PrintWriter pout = new
          PrintWriter(client.getOutputStream(), true);

       /* write the Date to the socket */
       pout.println(new java.util.Date().toString());

       /* close the socket and resume listening */
       client.close();
    }
  }
  catch (IOException ioe) {
    System.err.println(ioe);
  }
}
```

Date Client

```
public static void main(String[] args) {
  try {
    /* make connection to server socket */
    Socket sock = new Socket("127.0.0.1",6013);
    InputStream in = sock.getInputStream();
    BufferedReader bin = new
       BufferedReader(new InputStreamReader(in));

    /* read the date from the socket */
    String line;
    while ( (line = bin.readLine()) != null)
      System.out.println(line);
    /* close the socket connection*/
    sock.close();
  }
  catch (IOException ioe) {
    System.err.println(ioe);
  }
}
```