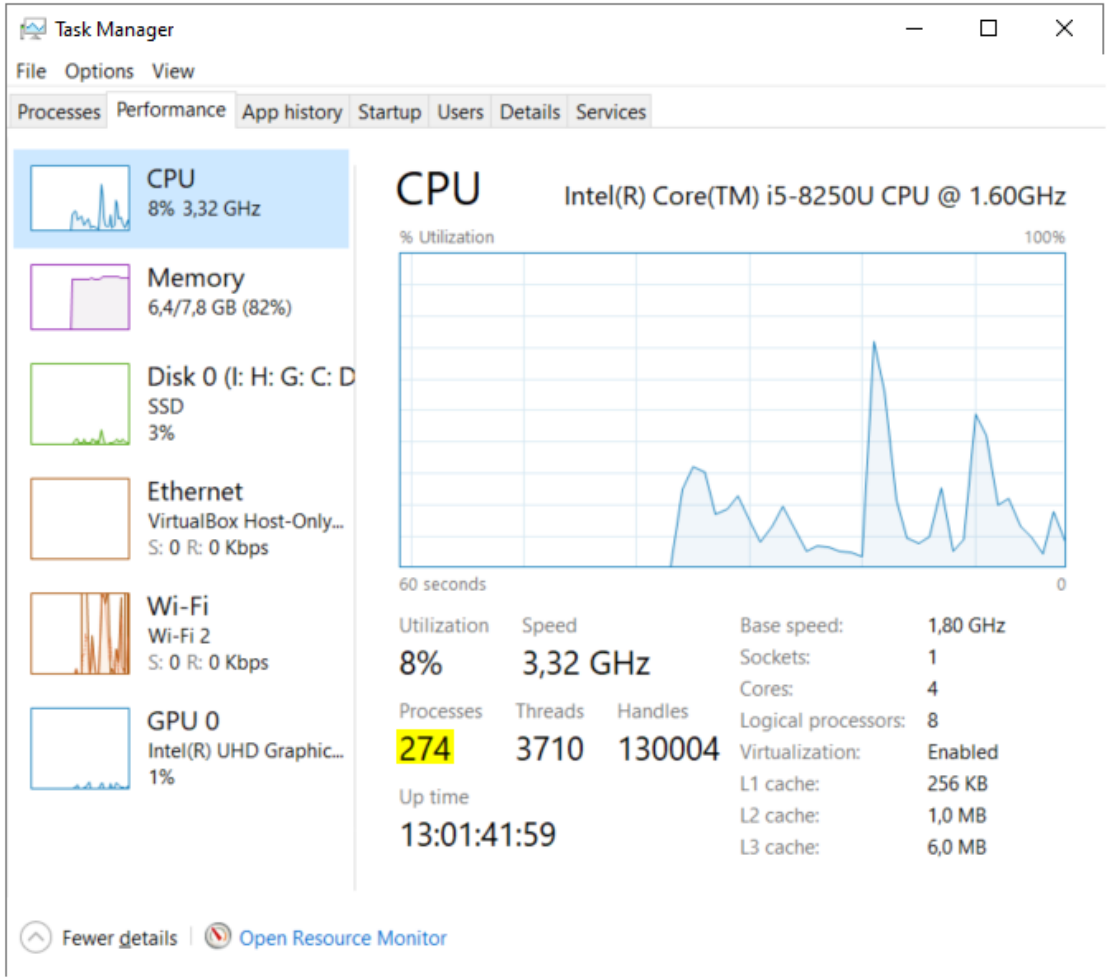
Bài 2
Process

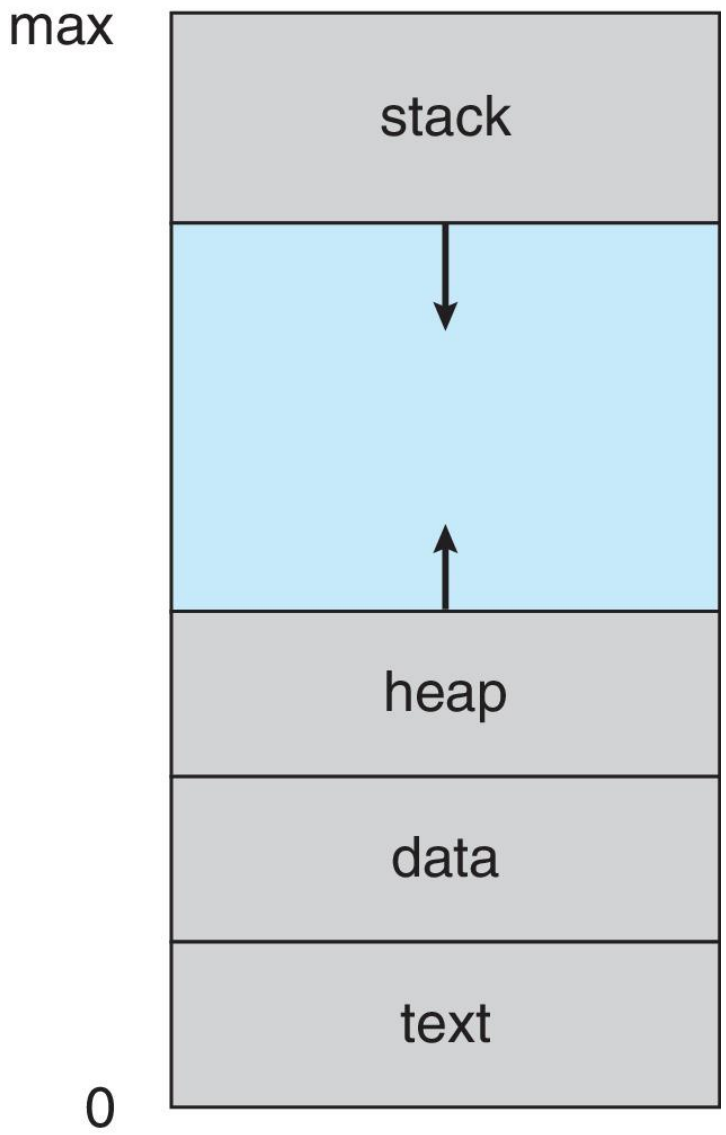
Khái niệm process

- Process (tiến trình) là một thực thể của một chương trình máy tính đang được thực thi ➔ Phân biệt *Process (tiến trình) vs Program (chương trình)*
- Program trở thành process khi nó được nạp vào bộ nhớ chính.
- Một chương trình khi thực thi có thể tạo nhiều process.

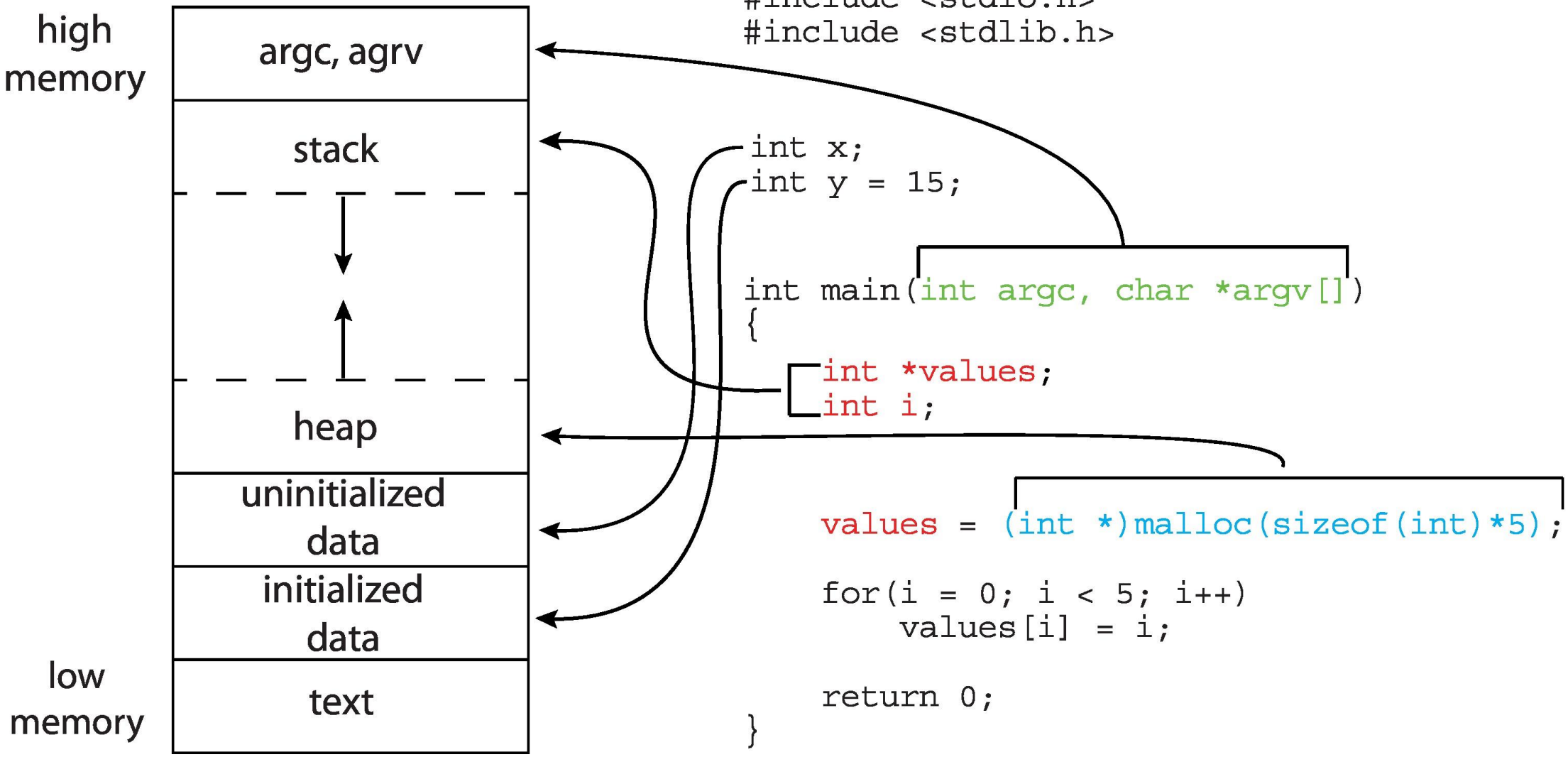


Process trong bộ nhớ

- Một tiến trình trong bộ nhớ có:
  - Stack: dữ liệu tạm (tham số của func, biến cục bộ)
  - Heap: bộ nhớ động được cấp phát khi tiến trình chạy.
  - Data section: biến toàn cục.
  - Text section: program code.



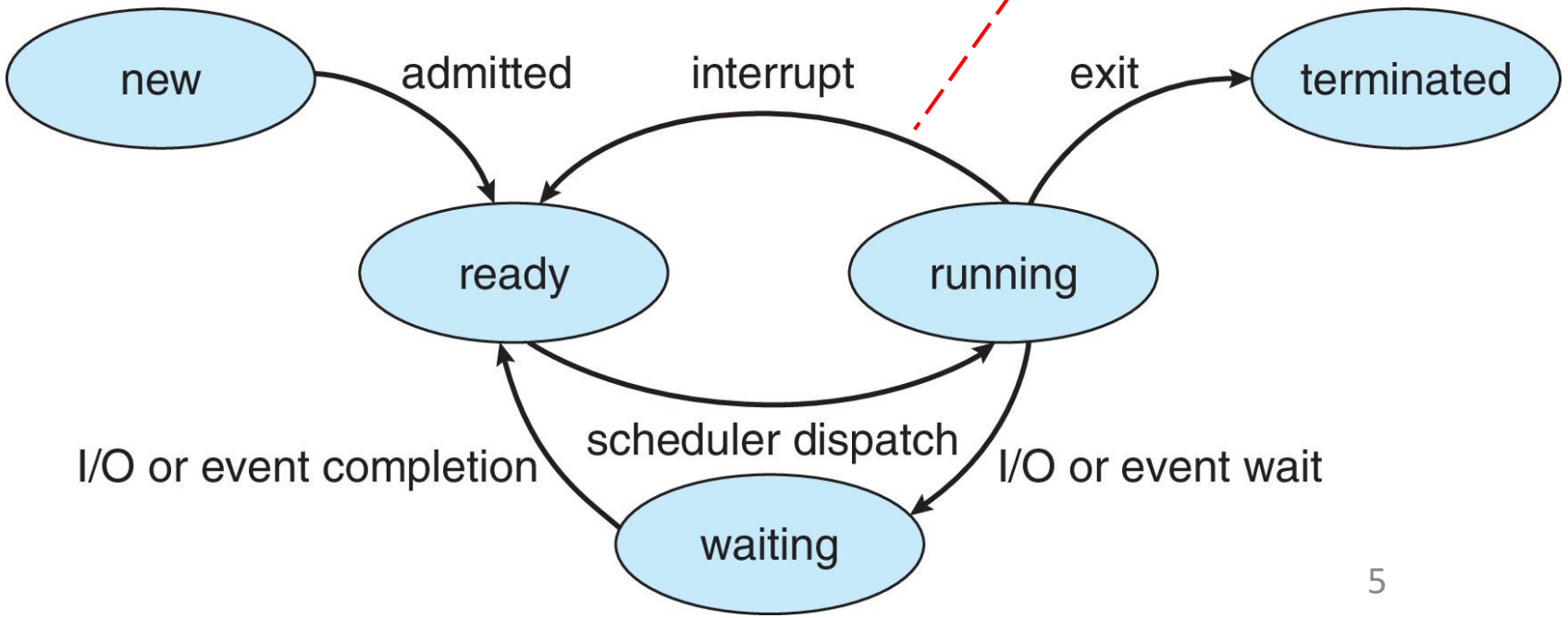
Bố trí bộ nhớ của một chương trình C



Trạng thái tiến trình

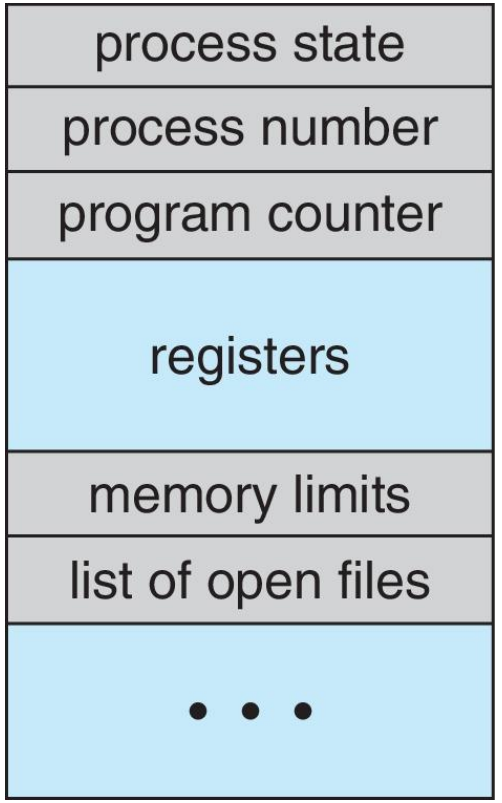
- Một process khi thực thi sẽ chuyển đổi giữa các trạng thái:
  - *New*: tiến trình vừa được tạo
  - *Running*: đang thực thi
  - *Waiting*: chờ một sự kiện xảy ra (ví dụ I/O)
  - *Ready*: sẵn sàng chờ được cấp CPU
  - *Terminated*: kết thúc thực thi.

Process được cấp/trả CPU thường xuyên, OS làm cách nào để quản lý các tài nguyên cũng như sự thực thi của process?



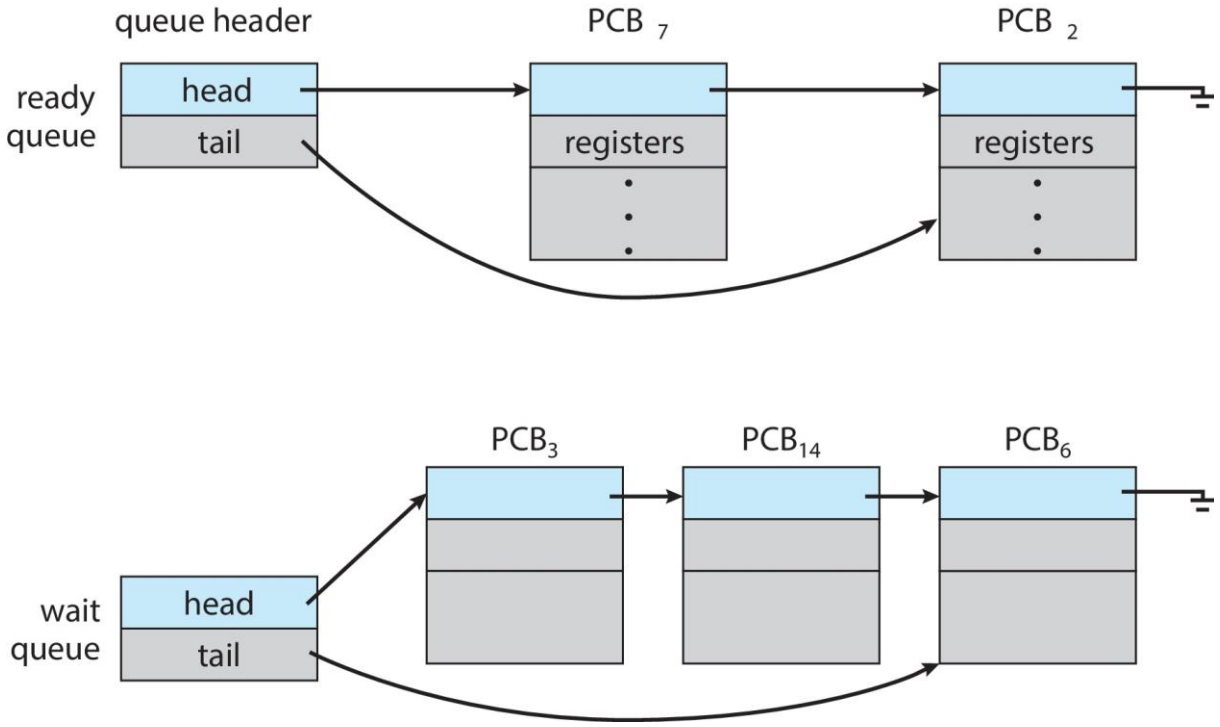
Process Control Block (PCB)

- Khối điều khiển tiến trình (Process Control Block - PCB) là một cấu trúc dữ liệu trong kernel chứa thông tin cần thiết để quản lý một tiến trình nhất định.
  - ***Process state*** – running, waiting, etc.
  - ***Process identifier (PID)***
  - ***Program counter*** – location of instruction to next execute
  - ***CPU registers*** – contents of all process-centric registers
  - ***CPU scheduling information*** - priorities, scheduling queue pointers
  - ***Memory-management information*** – memory allocated to the process
  - ***Accounting information*** – CPU used, clock time elapsed since start, time limits
  - ***I/O status information*** – I/O devices allocated to process, list of open files
- OS sử dụng PCB lưu thông tin process để thực hiện cơ chế đa chương.

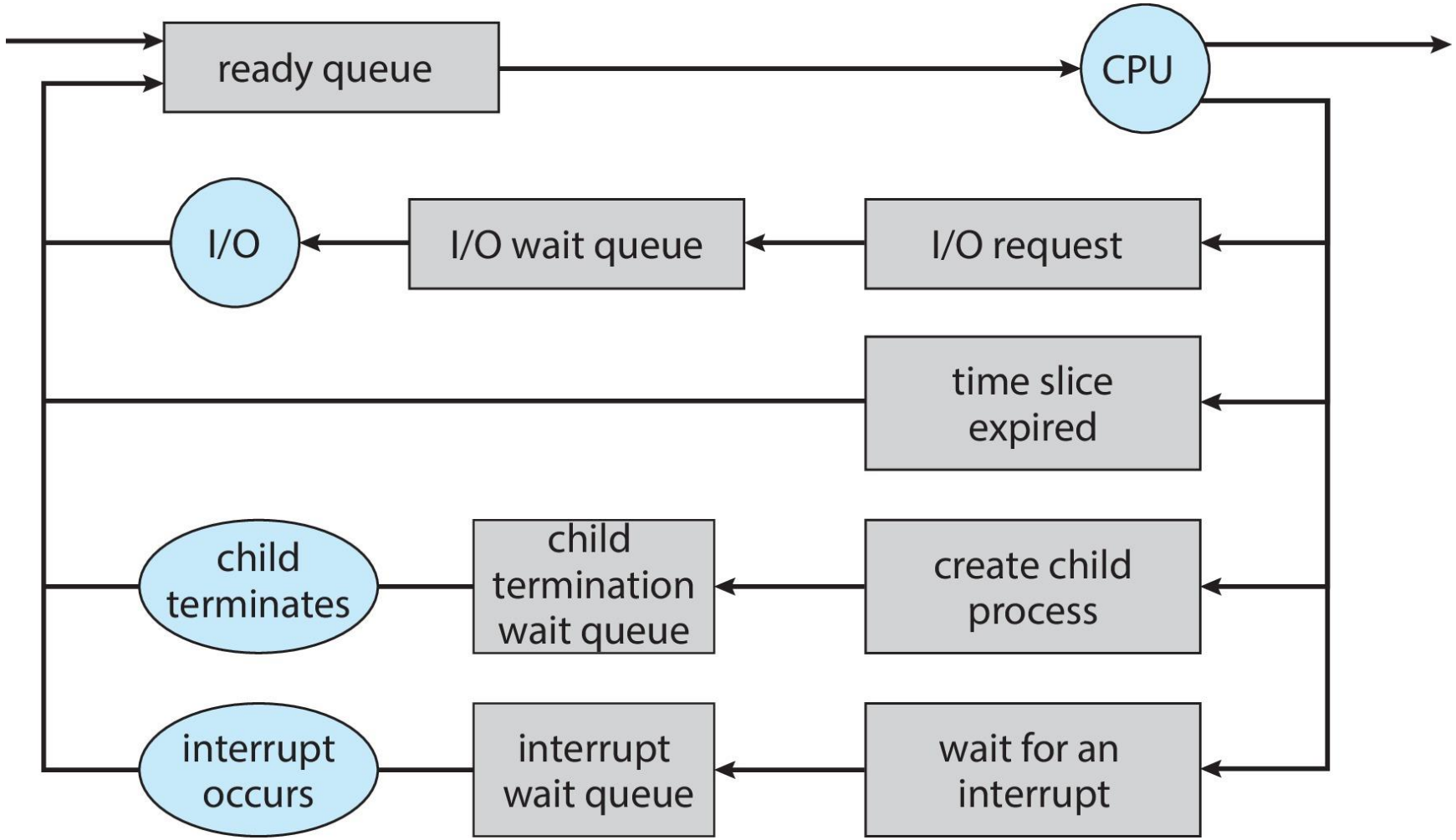


Process scheduling

- Process scheduler:
  - Chọn lựa process kế tiếp được giao CPU để thực thi nhằm tối ưu hóa việc sử dụng CPU.
  - Duy trì hàng đợi lập lịch (scheduling queues) tiến trình, gồm:
    - ✓ ***Ready queue***: các process nằm trên bộ nhớ chính, sẵn sàng và chờ thực thi.
    - ✓ ***Wait queue***: các process đang đợi một sự kiện (vd I/O)
  - Các process chuyển qua lại giữa 2 loại hàng đợi ready & wait.

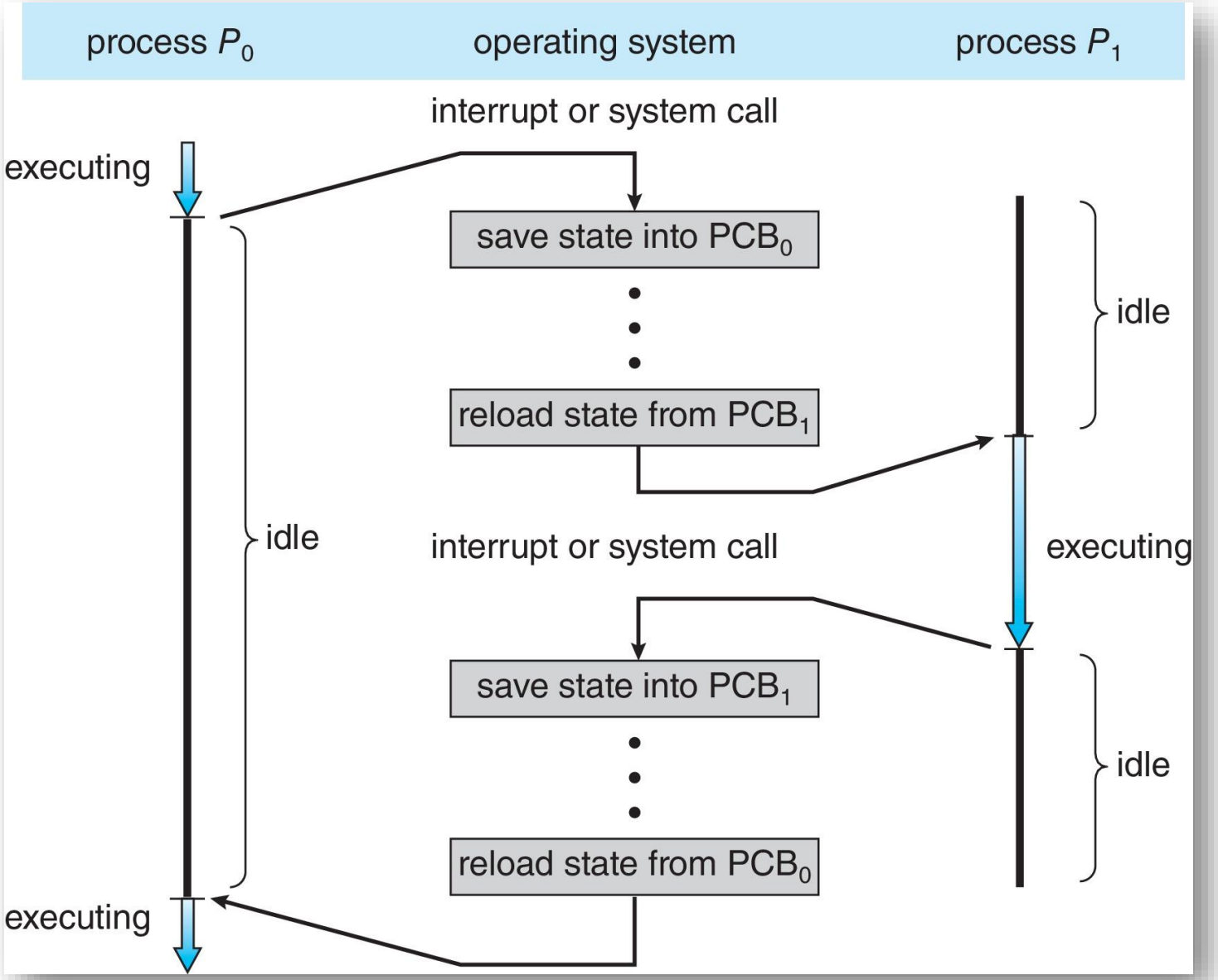


Process scheduling



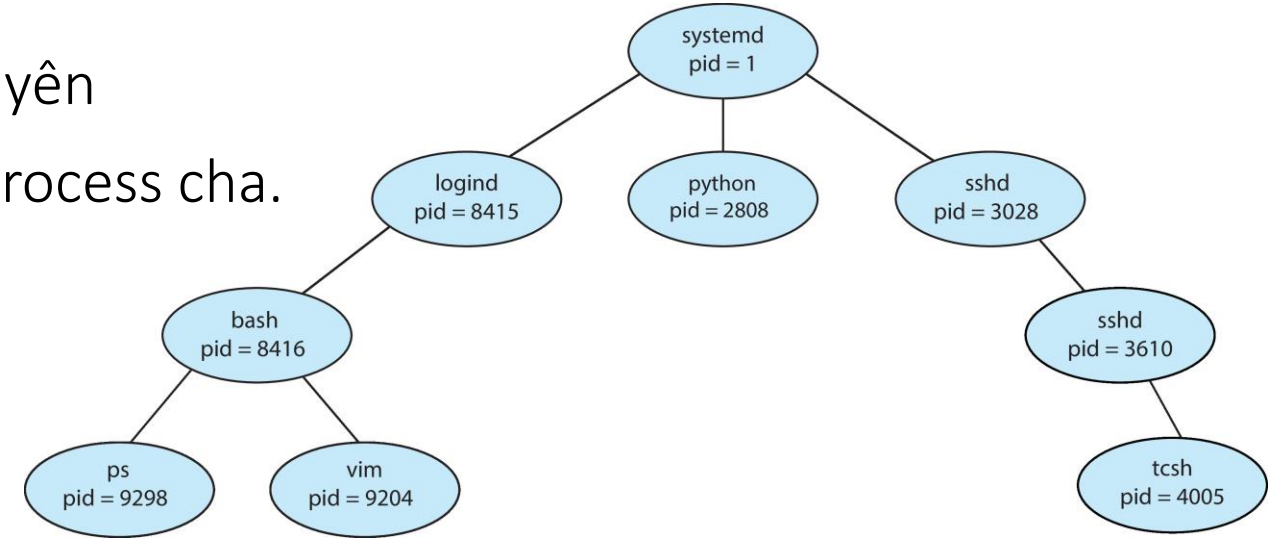
Context switch

- Context switch (chuyển ngữ cảnh) xảy ra khi CPU được chuyển từ một process này sang process khác.
- Sử dụng thông tin trong PCB.
- Thời gian chuyển ngữ cảnh thì CPU bị lãng phí
- OS & PCB càng phức tạp → thời gian chuyển ngữ cảnh càng lớn



Tạo & kết thúc process

- Một process có thể tạo ra process khác, được gọi là process cha và con → process tree.
- Process được nhận diện và quản lý qua Process Identifier (PID)
- Vấn đề chia sẻ tài nguyên:
  - Process cha và con chia sẻ tất cả tài nguyên
  - Process con chia sẻ 1 phần tài nguyên process cha.
  - Process cha và con độc lập tài nguyên
- Vấn đề thực thi:
  - Process cha và con thực thi đồng thời.
  - Process cha đợi cho đến khi process con kết thúc.



Tạo & kết thúc process

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

```c
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
     "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
     NULL, /* don't inherit process handle */
     NULL, /* don't inherit thread handle */
     FALSE, /* disable handle inheritance */
     0, /* no creation flags */
     NULL, /* use parent's environment block */
     NULL, /* use parent's existing directory */
     &si,
     &pi))
    {
       fprintf(stderr, "Create Process Failed");
       return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Tạo process trên Linux bằng lệnh fork() và trên Windows bằng Windows API

Tạo & kết thúc process

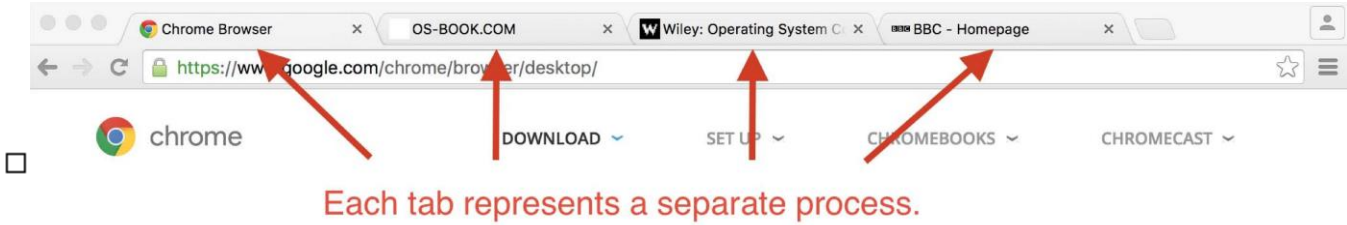
- Khi thực thi tới những statement cuối cùng, process yêu cầu OS kết thúc nó bằng cách gọi system call exit(), và:
  - Trả về trạng thái cho process cha (nếu có) qua system call wait()
  - Tài nguyên đã cấp phát sẽ bị OS thu hồi.
- Process cha có thể kết thúc process con bằng cách gọi system call abort() khi:
  - Process con sử dụng vượt mức tài nguyên được phép.
  - Việc xử lý của process con không cần thiết nữa.
  - Process cha sắp kết thúc, OS không cho phép một process con tiếp tục nếu process cha không còn nữa.

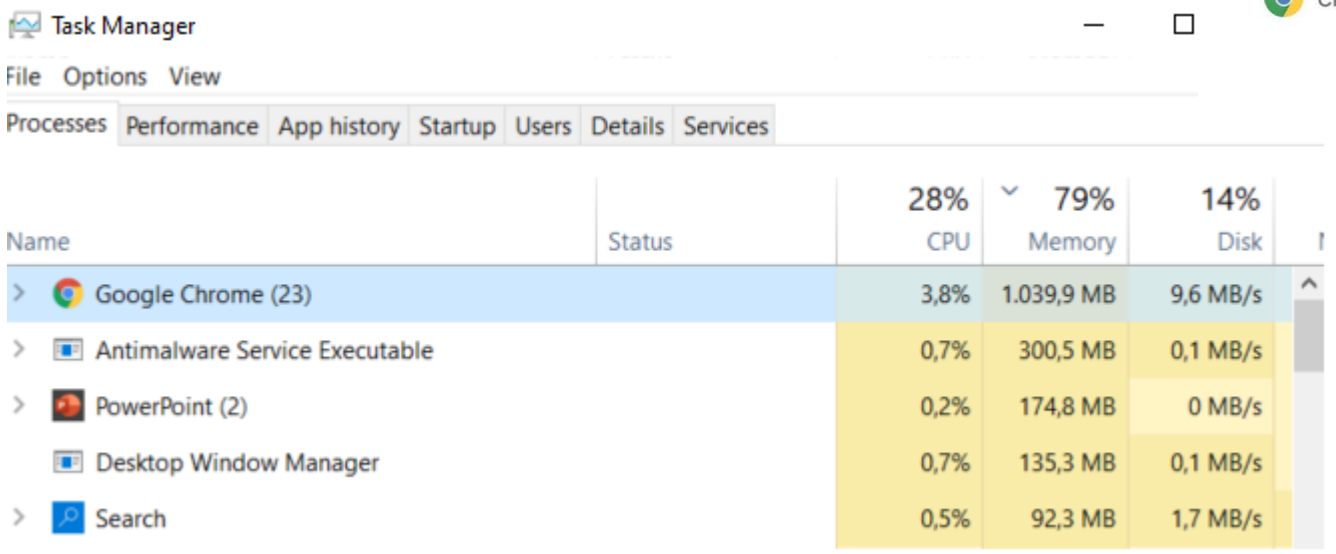
Tạo & kết thúc process

- Một số OS không cho phép child process tồn tại nếu parent process đã kết thúc → kết thúc process sẽ kết thúc tất cả các process "con, cháu,..."
- Parent process "đợi" child process bằng cách gọi system call wait().
- Nếu không có parent process nào đợi (không gọi wait()) → zombie process
- Nếu parent process bị hủy mà không gọi wait() → orphan process

Multiprocess Architecture

- Nhiều web browser hoạt động dưới dạng 1 process đơn → vấn đề?
- Google Chrome hoạt động theo kiến trúc multiprocess gồm 3 loại:
  - Browser: xử lý giao diện người dùng, disk, I/O.
  - Renderer: render trang web từ HTML, Javascript,… Một renderer process tương ứng với 1 website được mở. Thực thi trong sandbox để hạn chế tối đa sự ảnh hưởng nếu có lỗ hổng bảo mật.
  - Plug-in

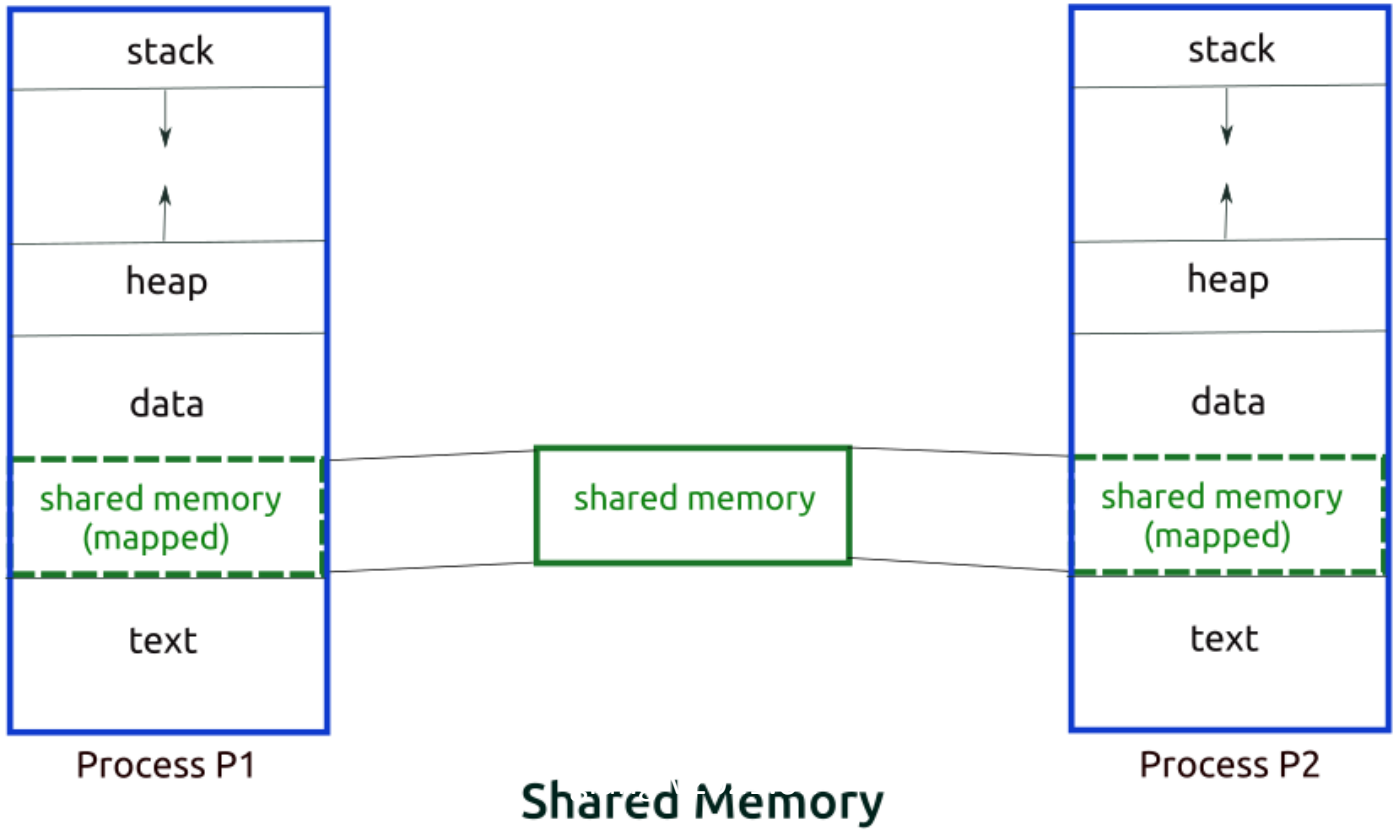




Interprocess communication (liên lạc liên tiến trình)

- Các process trong hệ thống có thể hoạt động: độc lập (independent) hoặc hợp tác (cooperating).
- Việc hợp tác có thể dẫn đến hoạt động&dữ liệu của các tiến trình ảnh hưởng nhau.
- Tại sao tiến trình phải hợp tác: chia sẻ thông tin, tăng tốc độ tính toán, module hóa.
- Các tiến trình muốn hợp tác cần **Interprocess Communication (IPC)**
- Hai mô hình IPC:
  - Shared memory
  - Message passing (Message queues)

- Là một vùng nhớ được chia sẻ giữa các process muốn giao tiếp → quyền điều khiển giao tiếp là của các process tham gia, không phải hệ điều hành.
- Các P tham gia có thể thay đổi cũng như nhận cập nhật về dữ liệu ở vùng nhớ chung.
- Thông tin được chia sẻ không đi qua kernel.



Interprocess communication → Shared memory

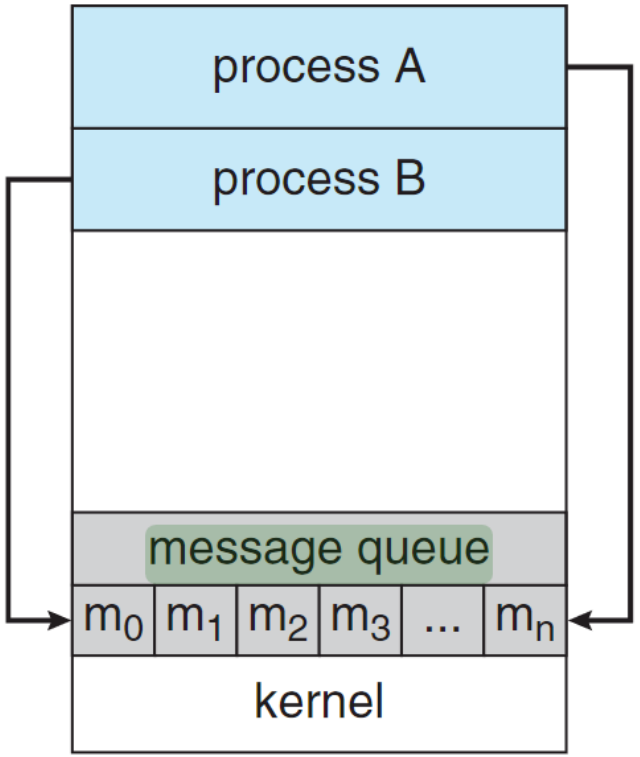
- Code ví dụ: https://www.geeksforgeeks.org/ipc-shared-memory/

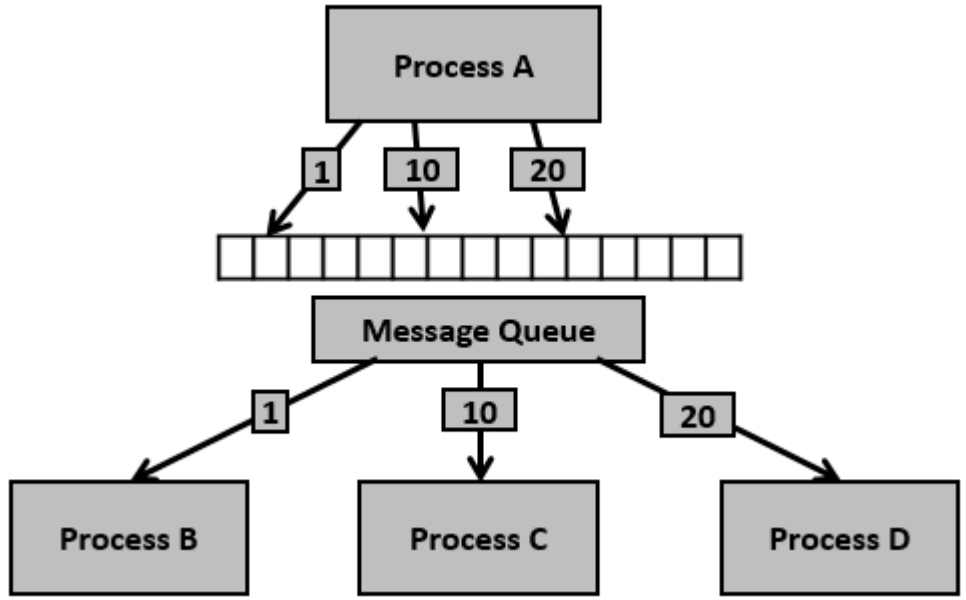
```cpp
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);

    cout<<"Write Data : ";
    gets(str);

    printf("Data written in memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

    return 0;
}
```

```cpp
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);

    printf("Data read from memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);

    return 0;
}
```

Interprocess communication → Message passing (Message queues)

- Một Message Queue (MQ) là một danh sách liên kết các message được lưu trữ tại kernel và nhận diện bởi MQ identifier.
- Process có thể tạo MQ mới hoặc truy cập vào MQ có sẵn.
- Message có thể lấy ra khỏi queue dựa trên type, không nhất thiết dùng FIFO.





Interprocess communication → Message passing (Message queues)

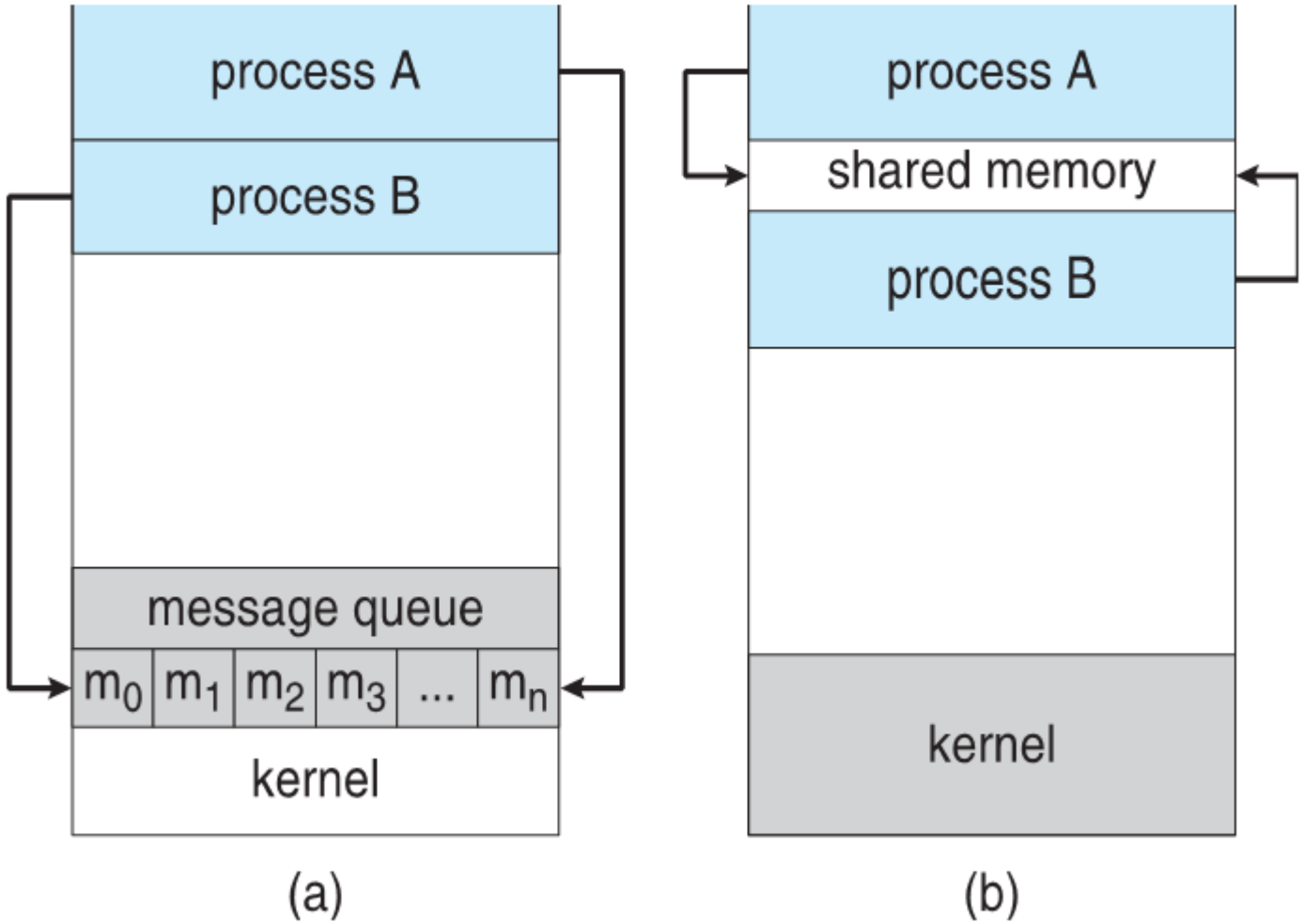
- Code ví dụ: https://www.geeksforgeeks.org/ipc-using-message-queues/

```c
// C Program for Message Queue (Writer Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    fgets(message.mesg_text,MAX,stdin);

    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);

    // display the message
    printf("Data send is : %s \n", message.mesg_text);

    return 0;
}
```

```c
// C Program for Message Queue (Reader Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n",
                    message.mesg_text);

    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

- Những điểm khác biệt giữa Shared Memory vs Message Queue

(a) Message passing. (b) shared memory.



Producer – Consumer problem

- Bài toán Sản xuất – Tiêu thụ:
  - Buffer (bộ đệm) dùng để chứa dữ liệu, có kích thước giới hạn/không giới hạn (bounded/unbouned buffer). Tiến trình Producer "sản xuất" dữ liệu và bỏ vào buffer. Tiến trình Consumer "tiêu thụ" dữ liệu trong buffer.
  - Producer không thể sản xuất nếu buffer đầy và ngược lại với Consumer.
- Cần giải quyết 2 vấn đề:
  - Giao tiếp liên tiến trình: cần có buffer chung.
  - Đồng bộ tiến trình: khi truy cập dữ liệu dùng chung.
- Ví dụ về buffer chung trong bài toán

Producer – Consumer problem

Shared data

```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int counter = 0;
```

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

        while (counter == BUFFER_SIZE)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
        counter++;
}
```

```
while (true) {
        while (counter == 0)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
         counter--;
        /* consume the item in next consumed */
}
```

- ✓ Giải quyết được vấn đề giao tiếp: cả producer và consumer cùng truy cập được dữ liệu dùng chung trên shared memory.
- ✓ Vấn đề đồng bộ: counter?

Producer – Consumer problem

- Race condition
  - **`counter++`** could be implemented as

```
register1 = counter
register1 = register1 + 1
counter = register1
```

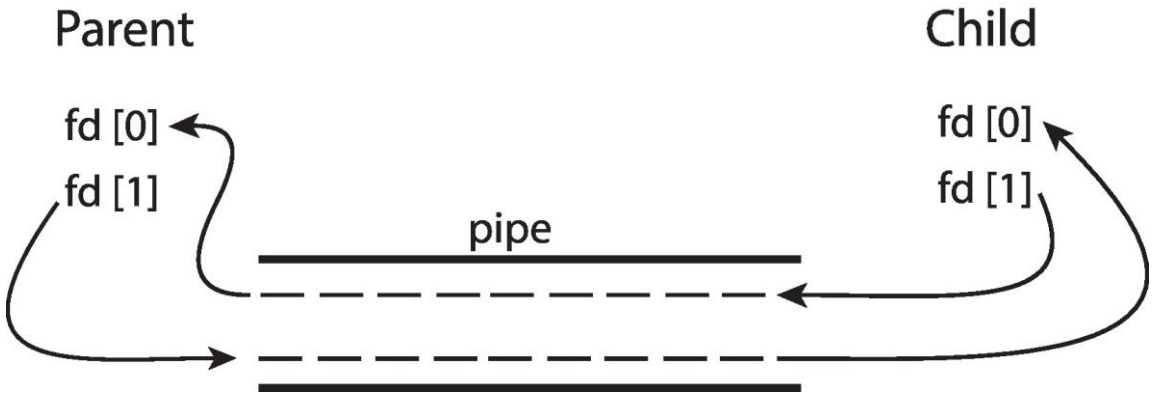
  - **`counter--`** could be implemented as

```
register2 = counter
register2 = register2 - 1
counter = register2
```

  - Consider this execution interleaving with "count = 5" initially:

S0: producer execute `register1 = counter` {register1 = 5}
S1: producer execute `register1 = register1 + 1` {register1 = 6}
S2: consumer execute `register2 = counter` {register2 = 5}
S3: consumer execute `register2 = register2 - 1` {register2 = 4}
S4: producer execute `counter = register1` {counter = 6 }
S5: consumer execute `counter = register2` {counter = 4}

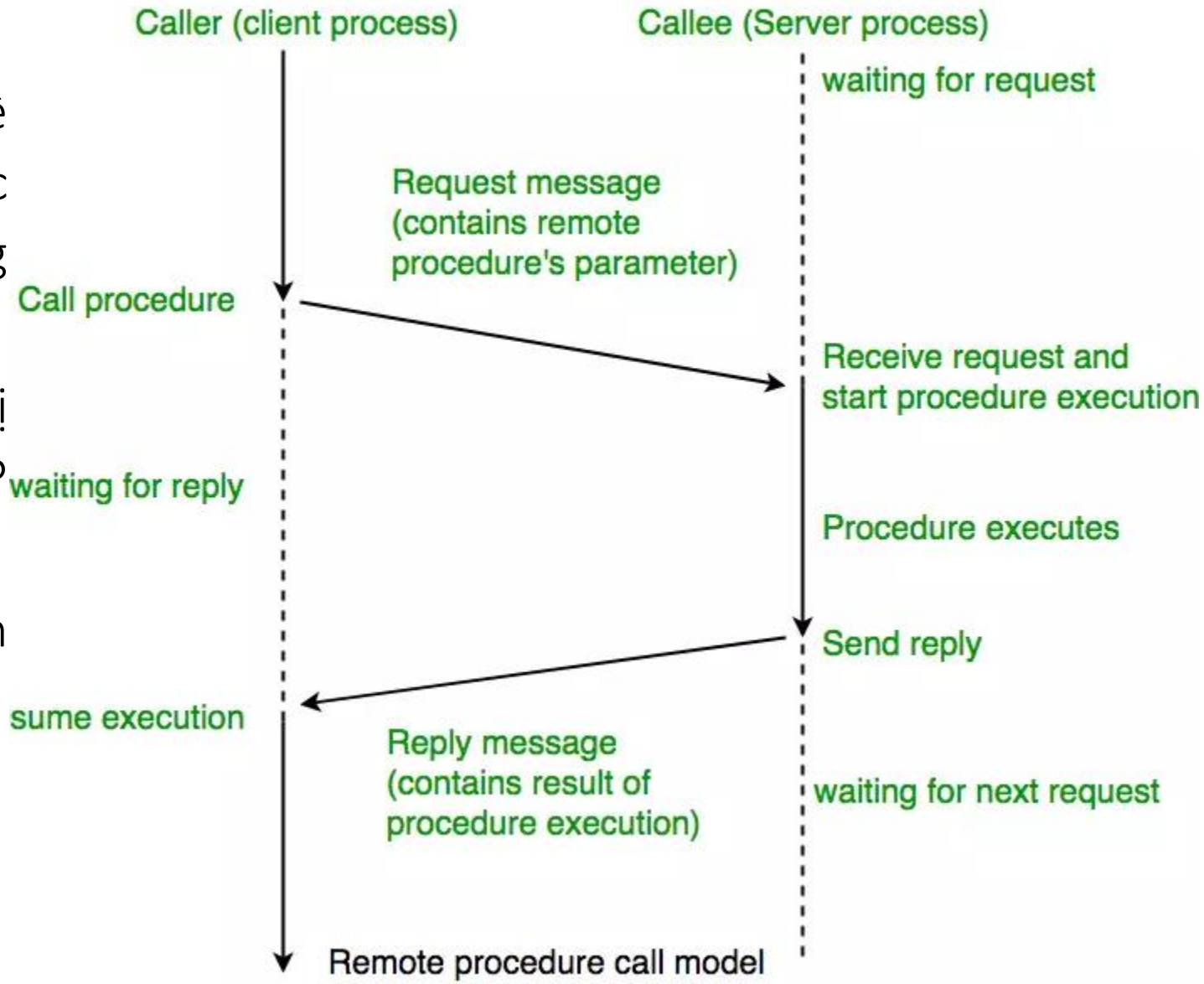
- Pipes:
  - Hoạt động như một "đường ống" kết nối hai process.
  - Gồm: Ordinary pipes (còn gọi là unnamed pipes hoặc Windows gọi là anonymous pipes) và Named pipes.
  - Ordinary pipes: một chiều, thường dùng đối với P có quan hệ cha con.



  - Named pipes: hai chiều, các P sử dụng không cần mối quan hệ cha con.

Một số cơ chế liên lạc khác

- Sockets:
  - Cho phép các P trên các hệ thống khác nhau có thể liên lạc với nhau, thường sử dụng trong mô hình client – server.
  - Sử dụng IP để xác định thiết bị và port number để xác định P trên thiết bị đó.
  - Điểm kết nối hai P ở hai đầu liên lạc gọi là socket.
  - Socket = IP + Port
- Remote Procedure Calls (RPC):



Tham khảo

- Operating System Concepts – 10th edition
- https://www.geeksforgeeks.org