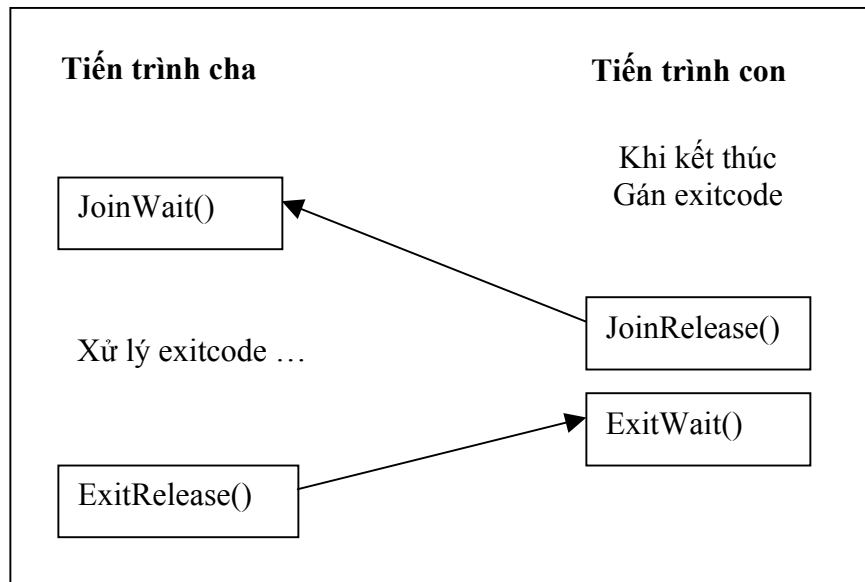


## ĐA CHƯƠNG VÀ ĐỒNG BỘ HOÁ

Chúng ta dùng lớp Ptable để quản lý thông tin các tiến trình. Khai báo lớp Ptable có thể như bên dưới. Lớp Ptable quản lý 10 tiến trình, mỗi tiến trình lại được mô tả chi tiết, và các hành động của nó được cài đặt trong lớp PCB. Và class BitMap để lưu vết số lượng process hiện hành

- Khai báo biến Ptable processTab toàn cục trong threads/system.h và khởi tạo trong threads/system.cc
- Sơ lược qua vòng đời của 1 process
  - Nếu là tiến trình đầu tiên thì process ID của nó là 0.
  - Nếu là tiến trình con A, sau lời gọi system call Exec() thì nó gọi Join() vào tiến trình cha. Thật sự là việc báo cho tiến trình cha phải đợi cho tiến trình con A kết thúc mới thực thi tiếp. Như vậy, khi mà gọi system call Join() thì HĐH sẽ gọi A->JoinWait() (semaphore đợi cho đến khi A kết thúc) và đến khi A kết thúc bằng system call Exit() thì HĐH sẽ gọi A->JoinRelease(), và gọi A->ExitWait() để xin phép exit, HĐH lúc đó sẽ gọi A->ExitRelease() để cho phép tiến trình con kết thúc. Kết luận, trong chương trình, khi mà xử lý cho system call Join() thì HĐH gọi pcb[x]->JoinWait(), kế nhận mã exitcode để xử lý cần thiết rồi gọi pcb[x]->ExitRelease() để cho phép tiến trình con này kết thúc (pcb[x] = tiến trình mới tạo). Và khi tiến trình mới tạo này thoát bằng system call Exit() thì ta sẽ viết mã: pcb[x]->JoinRelease() và pcb[x]->ExitWait(), nghĩa là tiến trình con signal cho tiến trình cha thực thi tiếp, và nó phải đợi cho tiến trình cha cho phép nó thoát (gọi pcb[x]->ExitWait()), khi đó HĐH sẽ gọi pcb[x]->ExitRelease() (hàm mà ta viết trong sysctem call SC\_Join)



```
#ifndef PTable_H
#define PTable_H
#include "bitmap.h"
#include "pcb.h"
#include "schandle.h"
#include "semaphore.h"

#define MAXPROCESS 10

class PTable {
private:
    BitMap *bm;
    PCB *pcb[MAXPROCESS];
    int psize;
    Semaphore *bmsem; // dùng để ngăn chặn trường hợp nạp 2 tiến trình cùng lúc
public:
    PTable(int size);
    // Khởi tạo size đối tượng pcb để lưu size process. Gán giá trị ban đầu là null. Nhớ
    khởi tạo *bm và *bmsem để sử dụng

```

```
~PTable();
```

```
// hủy các đối tượng đã tạo
```

```
int ExecUpdate();// return PID
```

//Thực thi cho system call SC\_EXEC, Kiểm tra chương trình được gọi có tồn tại trong máy không. Kiểm tra thử xem chương trình được gọi là chính nó không? Chúng ta không cho phép điều này. Kiểm tra còn slot trống để lưu tiến trình mới không (max là 10 process). Nếu thỏa các điều kiện trên thì ta lấy index của slot trống là ProcessID của tiến trình mới tạo này, giả sử là ID. Và gọi phương thức Exec của lớp PCB với đối tượng tương ứng quản lý process này, nghĩa là gọi pcb[ID]->Exec(...). Xem chi tiết mô tả lớp PCB ở bên dưới.

Phương thức này được gọi trong hàm xử lý system call SC\_Exec

```
int ExitUpdate(int ec);
```

//Được thủ tục xử lý cho system call SC\_Exit sử dụng. Trong system call này chúng ta kiểm tra thử PID có tồn tại không, sau đó chúng ta mới xử lý kết thúc tiến trình, phải gọi JoinRelease() để giải phóng sự chờ đợi của tiến trình cha và ExitWait() để xin phép tiến trình cha cho kết thúc.

Chú ý là sau khi gọi thủ tục này, chúng ta phải giải phóng tiến trình hiện tại bằng các dòng lệnh như sau, nếu là main process thì chúng ta gọi hàm Halt() luôn

```
int pid = currentThread->processID;
```

```
currentThread->FreeSpace();
```

```
if(pid == 0){//exit main process
```

```
interrupt->Halt();
```

```
}
```

```
currentThread->Finish();
```

```
int JoinUpdate(int id);
```

// Được sử dụng khi mà chúng ta viết thủ tục cho system call Join(), trong thủ tục này thì tiến trình cha gọi pcb[id]->JoinWait() để chờ cho tới khi tiến trình con kết thúc (trước khi tiến trình con kết thúc thì tiến trình con gọi JoinRelease()). Sau đó tiến trình cha lại gọi ExitRelease() để cho phép tiến trình con được kết thúc.

Chú ý là chúng ta không cho phép tiến trình join vào chính nó, hoặc là join vào tiến trình khác không phải là cha của nó.

```
int GetFreeSlot();
```

//Tìm free slot để lưu thông tin cho tiến trình mới

```
bool IsExist(int pid);
```

//kiểm tra có tồn tại process ID này không

```
void Remove(int pid);
```

//Xóa một processID ra khỏi mảng quản lý nó, khi mà tiến trình này đã kết thúc  
};

```
#endif
```

```
#ifndef PCB_H
```

```
#define PCB_H
```

```
#include "thread.h"
```

```
#include "synch.h"
```

```
class PCB{
```

```
private:
```

```
Semaphore *joinsem;//semaphore cho quá trình join
```

```

Semaphore *exitsem;//semaphore cho quá trình exit
Semaphore *mutex;
int    exitcode;
Thread  *thread;
int    pid;
int    numwait ;// số tiến trình đã join

public:
int    parentID; //ID của tiến trình cha
PCB(int id);
~PCB();
int Exec(char *filename,int pid);// nạp chương trình có tên lưu trong biến
filename và //processID sẽ là pid
int GetID() {return pid;}
int GetNumWait();
void JoinWait();
void ExitWait();
void JoinRelease();
void ExitRelease();
void IncNumWait();
void DecNumWait();
void SetExitCode(int ec){exitcode = ec;}
int  GetExitCode(){return exitcode;}
};
#endif

```