Mục lục

[MÔ TẢ CHI TIẾT 2](#_Toc477549812)

[PTABLE 2](#_Toc477549813)

[PCB 2](#_Toc477549814)

[BITMAP 3](#_Toc477549815)

[THREAD 4](#_Toc477549816)

[NHỮNG VẤN ĐỀ 8](#_Toc477549817)

[HƯỚNG DẪN SỬ DỤNG 9](#_Toc477549818)

[TÀI LIỆU THAM KHẢO 10](#_Toc477549819)

## PTABLE

Bảng quản lý tiến trình

1. *PTable(int size);*

Chức năng: Khởi tạo size đối tượng pcb để lưu size process. Gán giá trị ban đầu là null. Khởi tạo \*bm va \*bmsem sử dụng

1. *int* *ExecUpdate(char\* filename);*

Chức năng: Thực hiện cập nhật

Sau khi đã kiểm tra tính đúng đắn của tình trạng hiện tại như còn đủ chổ chứa, tham số truyền vào tên dài, file đã tồn tại hay chưa, ta cấp phát block mới và thực hiện. cuối cùng trả về pid nếu thực hiện thành công.

1. *int ExitUpdate(int ec);*

Chức năng: Thực hiện đưa ra khỏi mảng

Sau khi kiểm tra tồn tại và có phải tiến trình chính, ta thực hiện “Remove()”

1. *int JoinUpdate(int pID);*

Chức năng: Đưa vào mảng

Sau khi ngắt và lưu tình trạng hiện tại, kiểm tra bộ nhớ, có trùng vị trí hiện tại, ta tiến hành đưa vào

1. *int GetFreeSlot();*

Chức năng: Tìm một slot trống để lưu thông tin cho tiến trình mới

1. *void Remove(int pID);*

Chức năng: Xóa một processID ra khỏi mảng quản lý nó, khi mà tiến trình này đã kết thúc

## PCB

Khối quản lý tiến trình

1. *PCB(int id);*

Chức năng: Khởi tạo các biến: kiểm tra parentID, gán pid / numwait / exitcode / thread, khởi tạo joinsem / exitsem / mutex

1. *int Exec(char \*filename, int pID);*

Chức năng: nạp chương trình có tên lưu trong biến filename va processID – định danh tiến trình

Joinsem và exitsem là 2 con trỏ tới semaphore đề dùng cho tiến trình join và exit. Trong class Semaphore này gồm 2 hàm quan trọng void P() – đưa vào hàng và void V() – đẩy ra.

Semaphore được nhiều cho các mục đích khác nhau, nhưng cụ thể ở đây nó giúp bảo vệ quá trình kiểm soát việc chia sẻ tài nguyên được chặt chẽ. Nó giúp cho việc đảm bảo truy cập sử dụng chỉ duy nhất một nguồn tài nguyên được chia sẻ trong một thời điểm nhất định. Hay một tiến trình muốn đợi cho tiến trình khác làm việc gì đó.

Hai hàm này để quản lý tiến trình Join

1. *void JoinWait();*
2. *void ExitWait();*

Hai hàm này để quản lý tiến trình Release

1. *void JoinRelease();*
2. *void ExitRelease();*

Tăng giảm số tiến trình đã Join – với việc sử dụng biến mutex

1. *void IncNumWait();*
2. *void DecNumWait();*

## BITMAP

Là một dãy bit dùng để quản lý các thành phần đã được khởi tạo hay chưa trong một mảng – đánh dấu các khung trang còn trống

1. *BitMap(int nitem)*

Chức năng: Khởi tạo với nitem bits rỗng.

1. *void Mark(int which)*

Chức năng: Đánh dấu bit thứ which

1. *void Clear(int which)*

Chức năng: Xóa which bits

1. *void Test(int which)*

Chức năng: Kiểm tra which bits đã được đặt chưa

1. *int Find()*

Chức năng: Trả về số bit đầu chưa đánh dấu và đặt nó đã sử dụng – tìm và cấp phát 1 bit. Nếu tất cả đề đã đánh dấu thì trả về -1.

1. *int NumClear()*

Chức năng: Trả về số bít chưa đánh dấu

1. *void FetchFrom(OpenFile \*file)*

Chức năng: Truy cập – đọc dãy bit từ một file

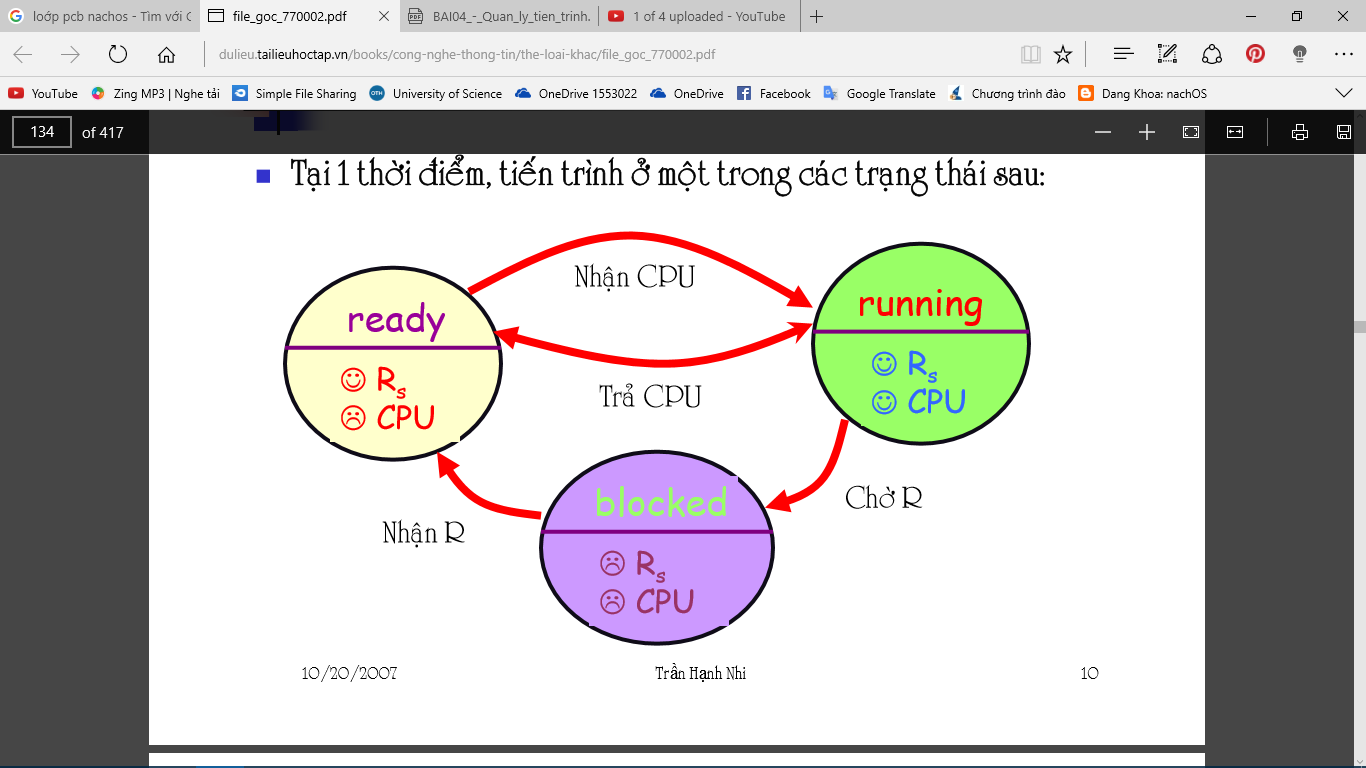
1. *void WriteBack(OpenFile \*file)*

Chức năng: Lưu trữ – viết dãy bit vào một file

## THREAD

Thread là một class hữu dụng để phục vụ việc quản lý đa chương với từng đơn tiến trình. Trong khi tiến trình chỉ cho phép thực hiện tuần tự thì tất cả các thread lại cho phép các tiến trình cụ thể chia sẻ cùng địa chỉ. Và do đó nó một thread sẽ có “local stack” nằm “private”. Tuy nhiên nó lại được chia sẻ với các thread khác để có thể truy cập lẫn nhau với các biến toàn cục. Để dễ dàng quản lý – theo dấu các threads, chúng được gán các trạng thái như sau:

* READY: trong trạng thái “sẵn sàng” dùng cho CPU, và cũng chỉ là sẵn sàng khi có thread khác đang chạy. Khi được chọn, chúng sẽ chuyển sang trạng thái RUNNUNG.
* RUNNING: trong trạng thái đang được sử dụng, và biến toàn cục currentThread sẽ trỏ đến nó.
* BLOCKED: trong trạng thái bị khóa – được đặt khi đưa vào hàm Thread::Sleep(). Khi đó có nghĩa là nó đang đợi một biến điều kiện hay báo hiệu nào đó và sẽ khi không nằm trong danh sách sẵn sang phục vụ.
* JUST\_CREATED: thread “đã được tạo” nhưng stack của nó vẫn còn trống. Trạng thái này được đặt khi nó được nằm trong hàm khởi tạo và sẽ được thay đổi khi vào hàm Thread::Fork().



1. *Thread(char \*debugName)*

Chức năng: Dùng để tạo ra một thread mới

Trạng thái của thread được đặt là JUST\_CREATED, stack của nó đặt là NULL, được đặt tên là debugName.

1. *void Yield()*

Chức năng: Nhường CPU cho một thread trong ready list

Hoãn thread đang sử dụng và chọn một thread mới. Nếu không có thread nào sẵn sang để thực hiện thì nó tiếp tục chạy thread hiện tại.

1. *void Sleep()*

Chức năng: Đưa tiến trình vào trạng thái BLOCKED.

Hoãn thread đang sử dụng, đổi trạng thái của nó sang BLOCKED và loại khỏi danh sách sẵn sàng. Nếu danh sách đó trống, interrupt->Idle() được gọi để đợi cho tới interrup kế tiếp. Hàm Sleep được sử dụng khi thread cần “khóa” lại đề đợi tới trường hợp cần thiết ở tương lai và tại đó nó sẽ không còn trạng thái BLOCKED cũng như được và danh sách sẵn sang để tiếp tục sử dụng.

1. *void Fork(VoidFunctionPtr func, int arg)*

Chức năng: Khởi tạo các thông tin cần thiết cho một tiểu trình và đưa nó vào CPU để thực hiện.

func chứa địa chỉ của tiến trình bắt đầu khi một thread bắt đầu thực hiện. Còn arg được gán vào thread mới. Fork sẽ cấp phát không gian chứa stack của thread mới, khởi tạo các thanh ghi. Khi func không được gọi như một tiến trình bình thường, thay vì trả về giá trị nào đó, thread đang chạy func đó sẽ kết thúc. Fork chỉ quan tâm đến chi tiết nào bằng cách tạo nên các bản ghi việc kích hoạt ở đầu.

1. *void Finish()*

Chức năng: kết thúc thread đang chạy

Vì ta không thể giải phóng stack khi thread với các dữ liệu đang sử dụng, nên ta sẽ giải phóng các dữ liệu bằng một thread khác. Dù hàm Finish trỏ biến toàn cục threadToBeDestroyed đến thread hiện tại nhưng không có nghĩa ta sẽ xóa bỏ nó. Sau đó nó sẽ gọi hàm Sleep đề khóa thread đó lại và đợi đến khi một thread mới khác thật sự chạy, nó sẽ kiếm tra lại biến threadToBeDestroyed để xóa nó đi.

1. *void StackAllocate(VoidFunctionPtr func, int arg)*

Chức năng: cấp phát stack và tạo nên các bản ghi việc kích hoạt ở đầu đề phục vụ cho func

1. Cấp phát bộ nhớ cho stack với StackSize
2. Đặt giá trị kiểm tra trên đầu stack. Bất cứ khi nào chuyển sang một thread mới, nếu một thread bị tràn stack trong quá trình hoạt động, thì scheduler sẽ kiểm tra giá trị có thay đổi hay không.
3. Đưa chương trình đếm PC trỏ đến ThreadRoot. Các công việc sẽ bắt đầu từ đây thay vì tại user-supplied, giúp cho việc đóng một thread nhanh chóng khi cần kết thúc. ThreadRoot được viết bằng hợp ngữ tại switch.s.
4. *void SaveUserState()*

Chức năng: lưu trạng thái các thanh ghi – trạng thái CPU ở mức độ người dùng – khi đang thực hiện phần việc ở mức người dùng

1. *void ResoreUserState()*

Chức năng: khôi phục trạng thái các thanh ghi – trạng thái CPU ở mức độ người dùng – khi đang thực hiện phần việc ở mức người dùng