TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM

KHOA CÔNG NGHỆ THÔNG TIN

PROJECT 3

GIẢI QUYẾT ĐA CHƯƠNG TRONG NACHOS

Học phần: Hệ điều hành

Lớp: 19_21

Giáo viên hướng dẫn: Lê Viết Long

Sinh viên thực hiện:

- Lê Kiệt – 19120554
- Hồ Hữu Ngọc – 19120602
- Nguyễn Phát Minh – 19120586
- Hoàng Mạnh Khiêm – 19120543

MỤC LỤC

I. LỚP PTABLE

Bảng quản lí tiến trình, dùng lớp **PCB** để thực hiện tiến trình con.

- *PTable(int size)*: Khởi tạo size đối tượng pcb để lưu size process. Gán giá trị ban đầu là null. Khởi tạo *bm va *bmsem sử dụng

- *int ExecUpdate(char* filename)*: Thực hiện cập nhật
  - Chỉ nạp một tiến trình vào một thời điểm
  - Thực hiện mở file để xem mở được hay file có tồn tại hay không, nếu không trả '-1'
  - Kiểm tra chương trình có gọi lại chính nó không, nếu có trả về '-1'
  - Gọi hàm GetFreeSlot() để kiếm một ID mới, nếu ID là '-1' nghĩa là không đủ ổ chứa sau đó sẽ trả về '-1'
  - Nếu thực hiện thành công ID mới thì sẽ cấp phát ô nhớ cho **PCB** có input là ID, sau đó gán và trả ra process ID
  - Trước lúc trả về gọi semaphore bm

- *int ExitUpdate(int ec)*; Chức năng: Thực hiện đưa ra khỏi mảng
  - Kiểm tra tiến trình hiện tại có tồn tại hay không
  - Kiểm tra có phải main process, nếu phải thì thực hiện Halt()
  - Thiết lập exitcode
  - Kiểm tra có nằm trong tình trạng chờ không, nếu không thì báo cho tiến trình cha thực thi tiếp còn tiến trình con thì exit, sau đó remove process ID. Nếu nằm trong tình trạng chờ thì cũng remove process ID
  - Trả về exitcode

- *int JoinUpdate(int pID)*: Đưa vào mảng
  - Kiểm tra ID có tồn tại không
  - Kiểm tra tiến trình đang chạy có phải là cha của tiến trình này hay không
  - Đợi tiến trình con kết thúc
  - Lấy exitcode
  - Kiểm tra hợp lệ exitcode
  - Cho phép tiến trình con kết thúc

- *int GetFreeSlot()*: Tìm một slot trống để lưu thông tin cho tiến trình mới

- *void Remove(int pID)*: Xóa một processID ra khỏi mảng quản lý nó, khi mà tiến trình này đã kết thúc
  - Kiểm tra tính hợp lệ của process ID
  - Đánh dấu bitmap đã sử dụng

- Xóa PCB

II. LỚP PCB

Khối quản lý tiến trình

- *PCB(int id)*: constructor khởi tạo các biến: kiểm tra parentID, gán pid / numwait / exitcode / thread, khởi tạo joinsem / exitsem / mutex

- *int Exec(char *filename, int pID)*: nạp chương trình có tên lưu trong biến filename và processID – định danh tiến trình
  - Tạo địa chỉ thread mới cho file input
  - Gán proccess ID cho thread
  - Trả về proccess ID

- *void JoinWait()*: Tiến trình cha đợi tiến trình con kết thúc
  - Gán ID tiến trình cha vào tình trạng chờ
  - Gọi IncNumWait(); để tăng số tiến trình chờ
  - Gọi semaphore join

- *void ExitWait()* Tiến trình con kết thúc.
- Gọi P() để giảm biến semaphore exit

- *void JoinRelease()*: Báo cho tiến trình cha thực thi tiếp
  - Gọi DecNumWait() để giảm số tiến trình chờ
  - Gọi semaphore join

- *void ExitRelease()*: cho phép tiến trình con kết thúc. Gọi semaphore exit
- Gọi V() để tăng biến semaphore exit

- *void IncNumWait()*: Tăng số tiến trình chờ
- *void DecNumWait()*: Giảm số tiến trình chờ

III. LỚP 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. Lớp này để lưu vết các tiến trình hiện hành. Gồm 1 mảng cờ hiệu để đánh dấu các khung trang còn trống để nạp vào page tương ứng ở class AddrSpace.

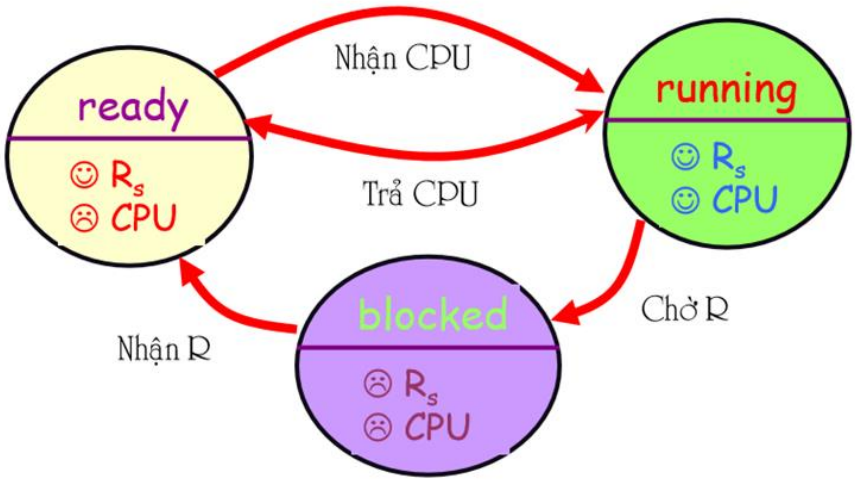
Các hàm quan trọng cần dùng:

- void Mark(int which): Đánh dấu khung trang này được sử dụng.
- int NumClear(): Trả về tổng số khung trang còn trống trên bộ nhớ
- BitMap(int nitem) : Khởi tạo với nitem bits rỗng.
- void Clear(int which) : Xóa which bits
- void Test(int which): Kiểm tra which bits đã được đặt chưa
- int Find(): 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.
- void FetchFrom(OpenFile *file) : Truy cập – đọc dãy bit từ một file
- void WriteBack(OpenFile *file): Lưu trữ – viết dãy bit vào một file

IV. LỚP 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().



- **Một vài hàm quan trọng:**

o *Thread(char *debugName) :* 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.

o *void Yield()* : 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.

o *void Sleep()* : 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.

o *void Fork(VoidFunctionPtr func, int arg) :* Khởi tạo các thông tin cần thiết cho một tiểu trình và đưa nó vào CPU để thực hiện.

Cấp phát vùng nhớ Stack cho tiến trình. Trong lúc cấp phát vùng nhớ Stack này, Fork gán con trỏ hàm VoidFunctionPtr (func), và số nguyên int (arg). Số nguyên này là tham số của hàm được trỏ bởi VoidFunctionPtr, ta cài số này là id của tiểu trình phụ chạy trong nachos

o *void Finish()* : 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. Dùng 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 kiểm tra lại biến threadToBeDestroyed để xóa nó đi.

- *void SaveUserState()* : lưu trạng thái các thanh ghi – trạng thái CPU ở mức độ người dùng – khi đang thực hiện phần việc ở mức người dùng
- *void ResoreUserState()* : 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
- *void StackAllocate(VoidFunctionPtr func, int arg)* : cấp phát stack và tạo nên các bản ghi việc kích hoạt ở đầu để phục vụ cho func
  - Cấp phát bộ nhớ cho stack với StackSize
  - 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.
  - 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.

V. XỬ LÝ ĐA CHƯƠNG NACHOS

1. Biến toàn cục

- Sử dụng biến toàn cục **Bitmap* gFrameControl** để quản lý các frames. Khởi tạo **gFrameControl** bằng đúng **NumPhysPages** = 128
- Sử dụng biến **Semaphore *sem** để quản lý độc quyền truy xuất cho việc cấp phát cho từng tiến trình trong constructor của file addrspace.h. Vì quản lý truy xuất độc quyền, mỗi lần chỉ cấp phát cho 1 tiến trình nên khởi tạo **sem** = 1
- Sử dụng biến **PTable *processTab** để quản lý bảng tiến trình, khởi tạo bảng có 10 đối tượng PCB để lưu kích thước tiến trình

2. Điều chỉnh file để hỗ trợ đa chương

- Chỉnh sửa class Thread trong file ./threads/thread.h:
  - Thêm **processID** để quản lý ID phân biệt giữa các tiến trình và **exitStatus** để kiểm tra exit code của tiến trình. Cả 2 biến này đều được khởi tạo là 0

- Hàm void **FreeSpace**() để giải phóng bộ nhớ khi tiến trình kết thúc

- Thêm constructor AddrSpace::AddrSpace(char* filename) trong file addrspace.h và addrspace.cc- constructor nhận vào tên 1 tiến trình (filename) và cấp phát cho tiến trình đó
  - Hàm đọc tên của tiến trình vào biến **executable** thuộc kiểu OpenFile*. Đoạn code sau đó được xử lý gần giống với constructor có sẵn của lớp AddrSpace, chỉ khác ở việc constructor mới thêm sẽ xử lý đa chương
  - Vì nhiều tiến trình có thể xảy ra cùng lúc với nhau nên constructor sẽ được gọi cùng lúc nhiều lần, điều này gây ảnh hưởng cho 1 tiến trình hiện đang xin cấp phát. Vì thế ràng buộc đoạn code cấp phát tiến trình giữa sem->P() và sem->V() để đảm bảo độc quyền truy xuất
  - Sau đó tính toán các kích thước cần thiết và kiểm tra số trang cần cho cấp phát (**numPages**) có lớn hơn số trang còn trống trên bộ nhớ không (**numclear**), nếu đúng thì không đủ bộ nhớ cấp phát à dừng việc cấp phát cho tiến trình đó. Số trang còn trống trên bộ nhớ (**numclear**) sẽ được lấy từ phương thức của lớp BitMap gFrameControl->NumClear().
  - Khi hỗ trợ đa chương, bộ nhớ sẽ không còn biểu diễn liên tiếp nhau nữa, vì thế khi cung cấp thông tin cho bảng trang pageTable[i].physicalPage, ta sẽ gán bằng phương thức gFrameControl->Find(). Phương thức Find() của lớp Bitmap tìm bit trống và đánh dấu nó đã sử dụng
  - Cuối cùng, nạp chương trình lên bộ nhớ chính

Minh họa constructor

```
OpenFile* executable = fileSystem->Open(filename);
//1 vài bước kiểm tra…
sem→P();
//tính toán kích thước…
int numclear = gFrameControl->NumClear(); //tính số trang trống
if(numPages > numclear) { // kiểm tra số trang cần cho cấp phát có lớn hơn số trang trống
      sem→V();
     //ngừng cấp phát
}

for (i = 0; i < numPages; i++) { // cấp phát không gian địa chỉ
     …
     pageTable[i].physicalPage = gFrameControl->Find();
     ….
}
sem->V();
// Nạp chương trình lên bộ nhớ chính
```

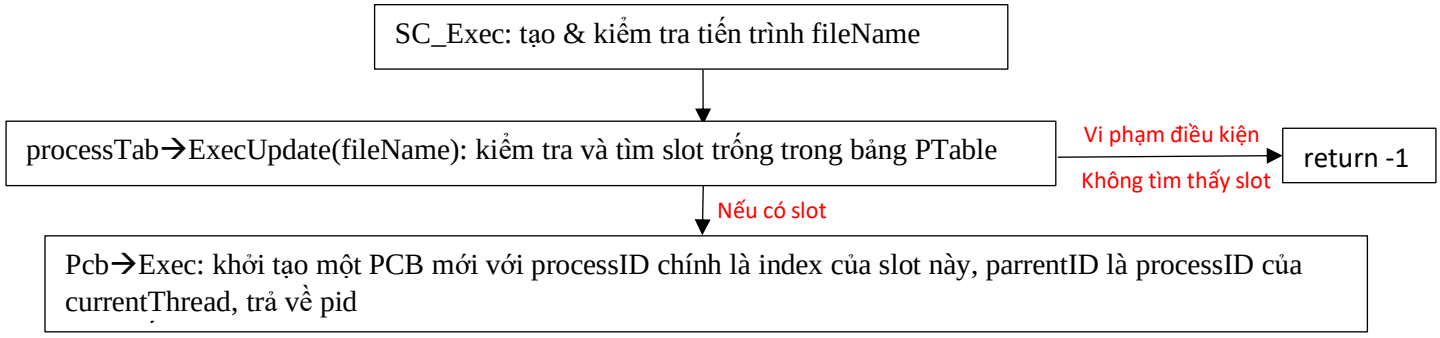
3. Thêm syscall

- Viết syscall cho 3 hàm Exec, Join, và Exit trong file syscall.h tương ứng với 3 case SC_Exec, SC_Join và SC_Exit trong file exception.cc

a) SC_Exec

Chức năng: Gọi thực thi một chương trình mới trong một system thread mới

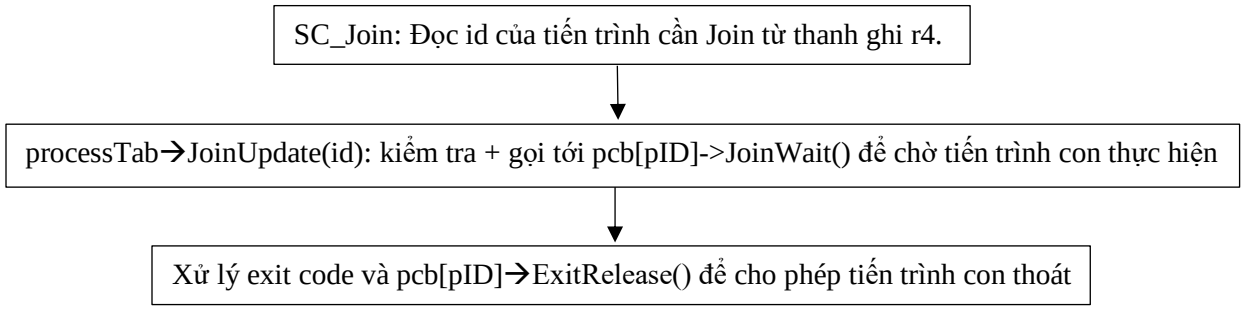
- Đầu tiên, đọc tên tiến trình truyền vào hàm thông qua thanh ghi r4, tên này được lưu tại địa chỉ **address**, sau đó truyền **address** vào User2System() để lấy được chuỗi tên nằm trong kernel space
- Kiểm tra có mở được tiến trình (hay tập tin) bằng phương thức Open() của lớp FileSystem, nếu không mở được thì in thông báo lỗi và kết thúc
- Cuối cùng, cấp phát block mới và id **pid** cho tiến trình thông qua hàm ExecUpdate() của lớp Ptable, trả về pid cho hàm Exec qua việc ghi **pid** vào thanh ghi r2
- Luồng chạy chính:



b) SC_Join

Chức năng: thực hiện đợi và block dựa trên tham số SpaceID **id**

- Đọc id của tiến trình cần Join từ thanh ghi r4
- Gọi thực hiện processTab->JoinUpdate(id) để join tiến trình con vào tiến trình cha → JoinWait() để chờ tới khi tiến trình con hoàn thành và ExitRelease() để tiến trình con kết thúc. Sau khi JoinUpdate(), lưu kết quả exit code vào thanh ghi r2
- Luồng chạy chính:



c) SC_Exit

Chức năng: thực hiện thoát tiến trình nó đã join

- Đọc exit status từ thanh ghi r4
- Nếu exit status = 0 (tức tiến trình exit 1 cách bình thường) thì gọi đến processTab→ExitUpdate(exitStatus) để ExitWait() - xin tiến trình cha cho thoát và JoinRelease() - giải phóng luôn tiến trình cha đang đợi nó (nếu có)
- Giải phóng bộ nhớ khi tiến trình kết thúc bằng phương thức currentThread→FreeSpace()
- Luồng chạy chính:

SC_Exit: Đọc exit status từ thanh ghi r4

↓

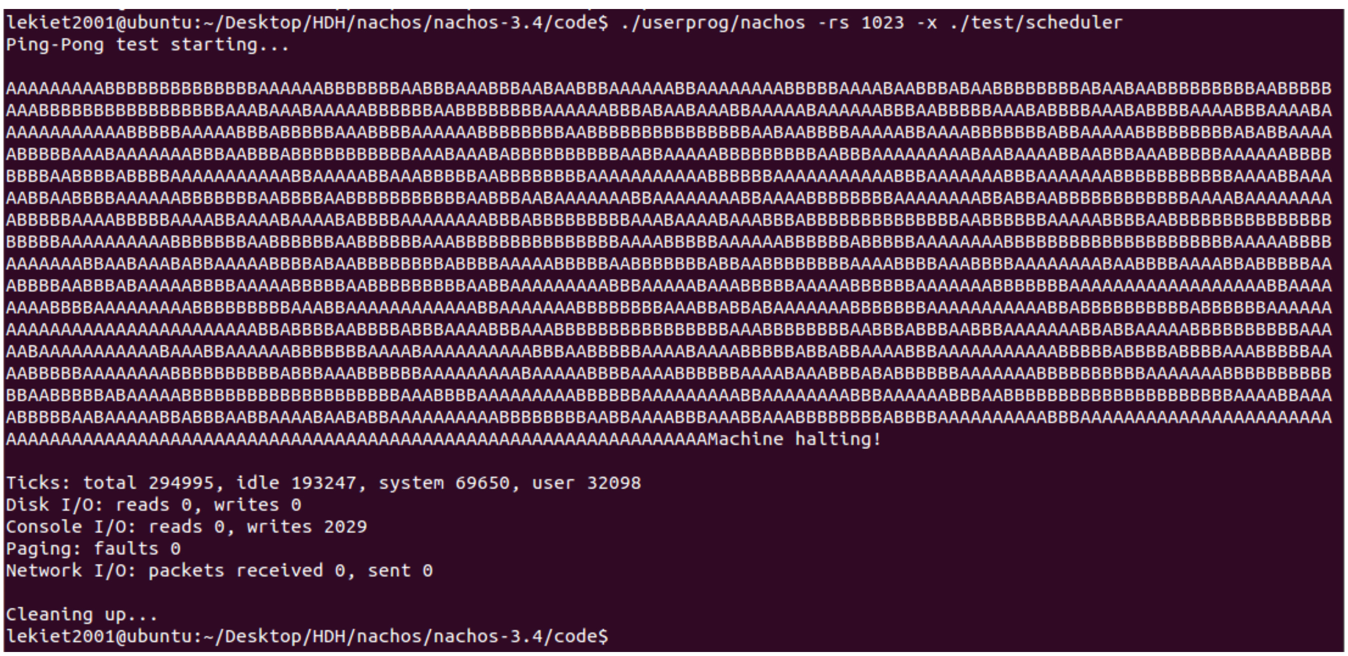
processTab->ExitUpdate(status):
- Nếu tiến trình là main process thì gọi Halt()
- Ngược lại gọi pcb[pID]→SetExitCode để đặt exitcode cho tiến trình gọi

↓

pcb[pID]→JoinRelease(): để giải phóng tiến trình cha đang đợi nó (nếu có)
pcb[pID]→ExitWait(): để xin tiến trình cha cho phép thoát

VI. DEMO

- cd vào thư mục nachos/nachos-3.4/code à make à ./userprog/nachos -rs 1023 -x ./test/scheduler



VII. THAM KHẢO

- Chapter 3, 4: https://www.studocu.com/en-au/document/university-of-southern-queensland/operating-systems/study-book/1570329