TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQGHCM

KHOA CÔNG NGHỆ THÔNG TIN

MÔN HỆ ĐIỀU HÀNH

BÁO CÁO ĐỒ ÁN 03

Mục lục

THÔNG TIN THÀNH VIÊN

MSSV	HỌ TÊN	CÔNG VIỆC
19120257	Phạm Anh Khoa	Tìm hiểu lớp Bitmap
19120301	Võ Thành Nam	Viết code
19120331	Phạm Lưu Mỹ Phúc	Tìm hiểu lớp PCB
19120389	Tô Gia Thuận	Tìm hiểu lớp Ptable
19120454	Bùi Quang Bảo	Tìm hiểu lớp Thread

ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

Yêu cầu	Tỉ lệ hoàn thành
Giải quyết đa chương trong Nachos	100%
Chương trình minh họa đa chương	100%
Báo cáo	100%
Tổng	100%

BÁO CÁO ĐỒ ÁN 03

Lớp Thread:

Sử dụng để quản lý thread - luồng.

Luồng là gì?

- Luồng là một đơn vị độc lập thực thi một quy trình hay chương trình.
- Đơn tiến trình chỉ cho phép thực hiện tuần tự. Việc quản lý luồng cho phép các tiến trình có thể cùng chia sẻ địa chỉ.
- Khi nhiều luồng được thực thi đồng thời cùng lúc, thì chúng ta gọi đó là đa luồng (multithreading) hay đa chương (multiprogramming).

Luồng bao gồm:

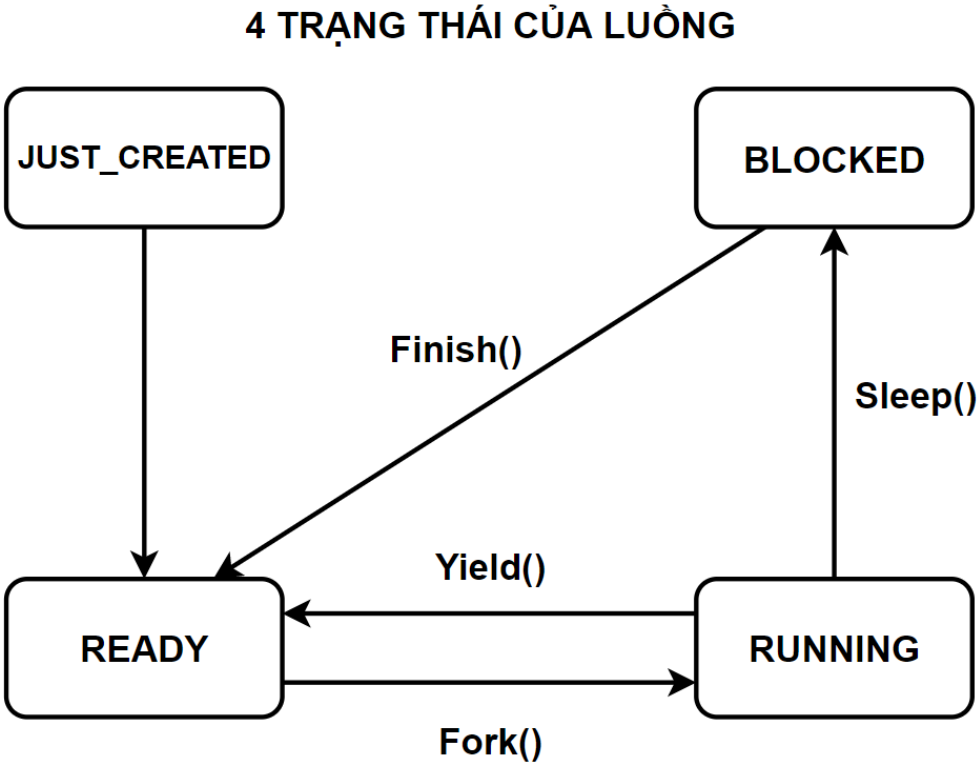
- Bộ đếm chương trình (program counter)
- Thanh ghi bộ xử lý (processor registers)
- Ngắn xếp thực thi (execution stack)

Lớp Thread:

- Một số thuộc tính:
  - int* stackTop: Con trỏ tới ngăn xếp hiện tại
  - int processID: ID dùng để phân biệt các tiến trình
  - ThreadStatus status: trạng thái của luồng, bao gồm:
    - **JUST_CREATED**: luồng đã được khởi tạo nhưng ngăn xếp đang trống
    - **RUNNING**: luồng đang được sử dụng
    - **READY**: luồng sẵn sàng để sử dụng
    - **BLOCKED**: luồng đã được đặt vào trạng thái ngủ
  - AddrSpace *space: Vùng nhớ của tiến trình trên RAM ảo
- **Quy trình quản lý luồng** phụ thuộc vào 4 phương thức chính:

- void **Fork**(VoidFunctionPtr func, int arg): Tạo một luồng, cấp phát vùng nhớ ngăn xếp cho tiến trình (phương thức StackAllocate), khởi tạo thanh ghi, đưa vào CPU để thực hiện, khiến cho luồng có thể chạy (*func)(arg)
- void **Yield**(): Nhường CPU cho một luồng khác nếu luồng đó nằm trong danh sách sẵn sàng để sử dụng, đưa luồng hiện tại vào danh sách sẵn sàng. Nếu không thì vẫn sử dụng luồng hiện tại.
- void **Sleep**(): Hoãn tiến trình và đặt luồng vào trạng thái ngủ, "BLOCKED", chờ đến thời điểm được gỡ bỏ trạng thái "BLOCKED" và được thêm vào danh sách sẵn sàng chạy.
- void **Finish**(): Luồng đang chạy hoàn tất.

- Một số phương thức khác:
  - void CheckOverflow(): Kiểm tra luồng có làm tràn ngăn xếp hay không.
  - void SaveUserState(): Lưu lại trạng thái của thanh ghi, mức người dùng
  - RestoreUserState(): Khôi phục trạng thái của thanh ghi, mức người dùng
  - void StackAllocate(VoidFunctionPtr func, int arg): Cấp phát ngăn xếp cho luồng.



Lớp bitmap

Bitmap là lớp có sẵn trong nachos bao gồm một mảng số nguyên có giá trị là 0 hay 1 dùng để đánh dấu các khung trang đã được sử dụng hay còn trống

Các thuộc tính bao gồm

- int numBits: Là số lượng số bit trong một bitmap
- int numWords: Là số lượng words trong một bitmap (1 word= 4 bytes= 32 bit)
- unsigned int *map: Là mảng số nguyên lưu các giá trị 0/1 của các bit

Trong đa chương lớp bitmap dùng để kiểm tra có còn đủ khung tranh trống cho tiến trình được nạp vào ở addrspace.cc hay không và để khởi tạo bảng khung trang pTable

Trong đó có các hàm quan trọng là

- int **Find**(): Trả về vị trí đầu tiên có khung trang trống đồng thời đánh dấu luôn khung trang thành đang sử dụng nếu không tìm thấy trang trống trả về -1

- void **Mark**(int which): Đánh dấu khung trang ở vị trí which thành 1 có nghĩa là chuyển thành đang sử dụng
- void **Clear**(int which): Đánh dấu khung trang ở vị trí which thành 0 có nghĩa là chuyển thành trang trống
- int **NumClear**(): Dùng để trả về số lượng khung trang còn trống

Ngoài ra còn có một số hàm khác là

- bool **Test**(int which): Kiểm tra khung trang ở vị trí which có đang được sử dụng hay không trả về True nếu đang sử dụng
- void **Print**(): Để in các giá trị bitmap trong dãy
- void **FetchFrom**(OpenFile *file): Là hàm để đọc giá trị dãy bitmap từ một file nachos
- void **WriteBack**(OpenFile *file): Là hàm để ghi giá trị dãy bitmap vào một file nachos

Lớp PCB:

Lớp PCB (Process Control Block) được dùng để lưu thông tin nhằm quản lý process.

Một số thuộc tính quan trọng:

- int parentID: ID của tiến trình cha.
- int JoinStatus: chứa ID của tiến trình đang join.
- int pid: định danh của tiến trình.
- thread* thread: lưu tiến trình được nạp.
- char FileName[32]: lưu tên của tiến trình.
- int numwait: số tiến trình đã join.

- 3 thuộc tính thuộc lớp Semaphore: *joinsem (semaphore cho quá trình join), *exit (semaphore cho quá trình exit), *mutex (semaphore cho quá trình nạp) có chức năng quản lý quá trình join, exit, và nạp thread.

Một số phương thức quan trọng:

- PCB (int id): hàm constructor khởi tạo joinsem/exitsem/mutex, pid, exitcode, numwait, parentID.
- ~PCB(): hàm destructor của lớp.
- Các hàm get dùng để trả về thông tin của các thuộc tính trong lớp.
- Các hàm set dùng để thiết lập thông tin thuộc tính trong lớp.
- Hàm IncNumWait(), DecNumWait() dùng để tăng, giảm tiến trình đã join bằng cách tăng giảm biến numwait.

  Joinsem và exitsem là 2 con trỏ tới semaphore để dùng cho tiến trình join và exit. Trong class Semaphore này gồm 2 hàm void P() – đưa vào hàng và void V() – đẩy ra. Semaphore được dùng để bảo vệ quá trình kiểm soát việc chia sẻ tài nguyên được chặt chẽ. Nó giúp việc đảm bảo truy cập sử dụng chỉ duy nhất một nguồn tài nguyên được chia sẻ trong một thời điểm nhất đinh hay một tiến trình muốn đợi cho tiến trình khác làm việc gì đó.

- Hàm dùng để quản lý tiến trình join:
  + void JoinWait(): JoinStatus = parentID, tăng số tiến trình join bằng hàm IncNumWait(). Sau đó, gọi joinsem->P() để tiến trình chuyển sang trạng thái block và ngừng lại chờ JoinRelease để thực hiện tiếp
  + void ExitWait(): gọi exitsem->P() để tiến trình chuyển sang trạng thái block và ngừng lại chờ ExitRelease để thực hiện tiếp.
- Hàm dùng quản lý tiến trình release:

+ void JoinRelease(): giảm số tiến trình, gọi joinsem->V() để giải phóng tiến trình và cho tiến trình cha tiếp tục thực thi

+ void ExitRelease(): gọi exitsem->V() để giải phóng tiến trình đang chờ, cho phép tiến trình con kết thúc.

- Void Exec(char*filename, int pID): được dùng để nạp chương trình. Phương thức thực hiện gọi mutex->P() để tránh nạp 2 tiến trình con cùng lúc. Nếu thread khởi tạo không thành công thì báo lỗi và gọi mutex->V() và trả về -1. Nếu khởi tạo thành công, đặt processID của thread này là pID, đặt parentID của thread là processID của thread gọi Exec, thực thi Fork(MyStartProcess, pID), gọi mutex->V() và trả về pID.
- Void MyStartProcess(int pID): dùng để khởi tạo và thực thi chương trình người dung, cấp vùng nhớ cho tiến trình, tạo thanh ghi, nếu không đủ vùng nhớ thì thông báo lỗi và return.

Lớp Ptable:

Lớp Ptable được dùng để quản lý các tiến trình đang được chạy trong hệ thống. Do đó, bên trong lớp Ptable sẽ sử dụng các lớp Bitmap và PCB.

Một số thuộc tính quan trọng:

- PCB ***pcb[MAXPROCESS]:** là một mảng dùng để mô tả tiến trình, trong đó có tối đa MAXPROCESS tiến trình.
- Bitmap ***bm**: dùng để quản lí các khung trang bằng cách đánh dấu các ô đã được sử dụng trong PCB.
- Semaphore ***bmsem**: ngăn chặn các trường hợp xảy ra đụng độ (do nạp 2 tiến trình cùng một lúc).
- int **psize**: kích thước bảng Ptable

Một số phương thức bên trong lớp:

- **PTable** (int size): phương thức khởi tạo Ptable. Bên trong phương thức, khởi tạo đối tượng pcb để lưu trữ số lượng size tiến trình.

- **~PTable** (): phương thức hủy của lớp.
- bool **IsExist** (int pID): kiểm tra xem tiến trình còn tồn tại hay không.
- int **GetFreeSlot** (): tìm vị trí trống trong khung trang.
- char* **GetName** (int pID): Từ giá trị định danh của tiến trình trả về tên của tiến trình.
- int **ExecUpdate** (char* filename): xử lý cho system call SC_Exec (thực thi chương trình)

  Bao gồm các bước:
  - Kiểm tra tính hợp lệ của tên chương trình, kiểm tra xem chương trình cần thực có đang được thực thi hay không (tự thực thi chính nó).
  - Tìm vị trí trống trong bảng Ptable, nếu không còn chỗ trống sẽ xuất ra thông báo "Đã vượt quá 10 tiến trình". Nếu còn chỗ thì sẽ nạp tiến trình vào bảng Ptable (thêm vào mảng pcb), đồng thời thêm tên tiến trình vào bảng pcb và tiến hành thực thi. Trả về giá trị pid (đã đề cập ở lớp PCB).
- int **JoinUpdate** (int pID): xử lý cho system call SC_Join (thêm tiến trình)

  Bao gồm các bước:
  - Kiểm tra tính hợp lệ của tiến trình
  - Kiểm tra tiến trình đang chạy có là cha của tiến trình cần tham gia hay không.
  - Cho phép tiến trình con thực thi và tiến trình cha đợi.
  - Tiến trình con kết thúc và thoát.
- int **ExitUpdate** (int ec): xử lý cho system call SC_Exit (thoát tiến trình)

  Bao gồm các bước:
  - Kiểm tra pid có tồn tại hay không, nếu có thì kiểm tra tiếp xem tiến trình đó có phải là main process thì gọi Halt() để chạy chương trình.
  - Gọi SetExitCode cho tiến trình, giảm số tiến trình chờ của tiến trình cha, sau đó xóa tiến trình khỏi bảng.
- void **Remove** (int pID) : xóa tiến trình có giá trị định danh pID ra khỏi bảng

Các bước chỉnh sửa và xây dựng chương trình:

- **Bước 1**: Khai báo các biến toàn cục gPhysPageBitMap để kiểm soát trang, processTab để quản lí tiến trình và addrLock cho semaphore trong system.h và system.cc.

```
extern Machine* machine;    // user program memory and registers
extern SynchConsole* gSynchConsole; // ket noi voi console
extern PTable* processTab; // Bien quan li cac tien trinh
extern Semaphore* addrLock; //semaphore
extern BitMap* gPhysPageBitMap; // Bien quan li cac trang
#endif
```

```
#ifdef USER_PROGRAM
    machine = new Machine(debugUserProg);   // this must come first
    gSynchConsole = new SynchConsole();
    addrLock = new Semaphore("addrLock", 1);
    gPhysPageBitMap = new BitMap(256);
    processTab = new PTable(MAXPROCESS);
#endif
```

- **Bước 2**: Thay đổi số khung trang và kích thước sector.

  Trong machine.h: #define NumPhysPages 128

  Trong disk.h: #define SectorSize 512

- **Bước 3**: Trong class Thread của thread.h, thêm biến processID để kiểm soát id của tiến trình và hàm FreeSpace để giải phóng vùng nhớ của thread.

- **Bước 4**: Trong file addrspace.cc:
  - Constructor AddrSpace: Sử dụng addrLock để tạo miền găng, thêm đoạn code kiểm tra số trang trống và cách cấp phát trang trống.
  - Vì hàm MyStartProcess trong file pcb được cấp sử dụng constructor AddrSpace với tham số truyền vào là filename, nên ta cần tạo thêm một constructor AddrSpace tương tự với constructor đã có nhưng với tham số truyền vào là filename.
  - Thay đổi đoạn lệnh thêm data vào vùng nhớ.

```
    // then, copy in the code and data segments into memory
    if (noffH.code.size > 0){
    for(i = 0; i < numPages ; i++)
        executable->ReadAt(&(machine->mainMemory[noffH.code.virtualAddr]) +
        (pageTable[i].physicalPage*PageSize),PageSize,noffH.code.inFileAddr + (i*PageSize));
    }

    if (noffH.initData.size > 0){
    for(i = 0 ; i < numPages ; i++)
        executable->ReadAt(&(machine->mainMemory[noffH.initData.virtualAddr]) +
        (pageTable[i].physicalPage*PageSize),PageSize, noffH.initData.inFileAddr+(i*PageSize));
    }
```

```
    // Mien gang
    addrLock->P();

    // how big is address space?
    size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
            + UserStackSize;    // we need to increase the size
                                // to leave room for the stack

    // Number page process need
    numPages = divRoundUp(size, PageSize);
    size = numPages * PageSize;
    // Kiem tra xem con trang trong hay khong
    if(numPages > gPhysPageBitMap->NumClear())
    {
        printf("\nAddrSpace::Load : not enough memory for new process");
        numPages = 0;
        delete executable;
        addrLock->V();
        return;
    }

    DEBUG('a', "Initializing address space, num pages %d, size %d\n",
                numPages, size);
    // first, set up the translation
    pageTable = new TranslationEntry[numPages];

    for (i = 0; i < numPages; i++)
    {
        pageTable[i].virtualPage = i;   // for now, virtual page # = phys page #
        pageTable[i].physicalPage = gPhysPageBitMap->Find(); //tim trang trong
        pageTable[i].valid = TRUE;
        pageTable[i].use = FALSE;
        pageTable[i].dirty = FALSE;
        pageTable[i].readOnly = FALSE;  // if the code segment was entirely on
                                // a separate page, we could set its
                                // pages to be read-only
    }

    addrLock->V();
    // Ket thuc mien gang
```

- **Bước 5**: Trong class PCB của pcb.h và pcb.cc, thêm thuộc tính FileName để lưu tên tiến trình và các setter, getter tương ứng.

- **Bước 6**: Cài đặt các system call Exec, Join, Exit, thủ tục User2System và System2User theo hướng dẫn trong slide seminar và tham khảo từ file shandle.cc.

Demo chương trình:

Link demo: https://youtu.be/DZI9t8kCF80

- Chương trình ping:

```
1  #include "syscall.h"
2
3  int main(){
4      int i;
5      for(i=0;i<1000;i++)
6          PrintChar('A');
7  }
```

- Chương trình pong:

```
1  #include "syscall.h"
2
3  int main(){
4      int i;
5      for(i=0;i<1000;i++)
6          PrintChar('B');
7  }
```

- Chương trình scheduler:

```c
#include "syscall.h"

void main(){
    int pingPID,pongPID;
    PrintString("Ping Pong test starting...\n");
    pingPID=Exec("./test/ping");
    pongPID=Exec("./test/pong");
    Join(pingPID);
    Join(pongPID);
}
```

- Kết quả thực thi chương trình:

```
nam@ubuntu: ~/Desktop/hdh/Nachos_Project1/nachos/nachos-3.4/code
cd bin; make all
make[1]: Entering directory `/home/nam/Desktop/hdh/Nachos_Project1/nachos/nachos-3.4/code/bin'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/nam/Desktop/hdh/Nachos_Project1/nachos/nachos-3.4/code/bin'
cd test; make all
make[1]: Entering directory `/home/nam/Desktop/hdh/Nachos_Project1/nachos/nachos-3.4/code/test'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/nam/Desktop/hdh/Nachos_Project1/nachos/nachos-3.4/code/test'
nam@ubuntu:~/Desktop/hdh/Nachos_Project1/nachos/nachos-3.4/code$ ./userprog/nachos -rs 1023 -x ./test/schedu
ler
Ping Pong test starting...
BBBBAAABBBBBAAAAABAAAAAABAAAAAABBBBBBBAABBBAAABBBAABAABBBAAAAAAABBAAAAAAAABBBBBAAAABAABBBABAABBBBBBBBABAABAAB
BBBBBBBAABBBBBAAABBBBBBBBBBBBBBBBBAAABAAABAAAAABBBBBAABBBBBBBBBAAAAAABBBABAABAAABBAAAAABAAAAAABBBAABBBBBAAA
BABBBBAAABABBBBAAAABBBAAAABAAAAAAAAAAAAABBBBBAAAAABBBABBBBBAAABBBBAAAAAAABBBBBBBBAABBBBBBBBBBBBBBBBAABAABBBBAAA
AABBAAAABBBBBBBABBAAAAABBBBBBBBABABBAAAAABBBBBAAABAAAAAAABBBAABBBABBBBBBBBBBBBAAABAAABABBBBBBBBBBAABBAAAAABB
BBBBBBBAABBBAAAAAAAAABAABAAAABBAABBBAAABBBBBAAAAAABBBBBBBBAABBBBABBBBAAAAAAAAAAAABBAAAAABBAAABBBBBAABBBBBBBA
AAAAAAAAAABBBBBBAAAAAAAAAAAABBBAAAAAAAABBBAAAAAAAABBBBBBBBBBBAAAABBAAAAABBAABBBBAAAAAAABBBBBBBAABBBBAABBBBBBBBB
BAABBBAABAAAAAAAABBAAAAAAAABBAAAABBBBBBBBAAAAAAAABBABBAABBBBBBBBBBBBAAAABAAAAAAAAAAABBBBBAAABBBBBAAAABBAAAABAA
AABABBBBAAAAAAAABBBABBBBBBBBBAAABAAAABAAABBBABBBBBBBBBBBBBBBBAABBBBBBAAAAABBBBAABBBBBBBBBBBBBBBBBBBBAAAAAAAAAA
BBBBBBBAABBBBBBBAABBBBBBBAAABBBBBBBBBBBBBBBAAAABBBBBAAAAAABBBBBBABBBBBAAAAAAAABBBBBBBBBBBBBBBBBBBBAAAAABBBBAA
AAAAABBAABAAABABBAAAAABBBBABAABBBBBBBBBABBBBAAAAABBBBBAABBBBBBBABBAABBBBBBBBAAAABBBBAAABBBBAAAAAAAAABAABBBBAAA
ABBABBBBBAAABBBBAABBBABAAAAABBBBAAAAABBBBBAABBBBBBBBBBAABBAAAAAAAAAABBBAAAAABAAABBBBBAAAAABBBBBBAAAAAAAABBBBBBB
AAAAAAAAAAAAAAAAAABBAAAAAAAAABBBBAAAAAAAAAABBBBBBBBBAAABBAAAAAAAAAAAAAABBAAAAAAAABBBBBBBAAABBABBABAAAAAAAABBBBBBB
AAAAAAAAAAABBABBBBBBBBBBABBBBBBAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBABBBBAABBBBABBBAAAABBAAAABBBBBBBBBBBBBBBBBBBAAABB
BBBBBBAABBBABBBAABBBAAAAAAABBABBAAAAABBBBBBBBBBAAAAABAAAAAAAAAAABAAABBAAAAAAABBBBBBAAAABAAAAAAAAAAABBBAABBBBB
AAAABAAAABBBBBABBBABBAAAAABBBAAAAAAAAAAAABBBBBABBBBABBBBAAABBBBBAAAABBBBBAAAAAAAAABBBBBBBBBABBBAAABBBBBBAAAAAAA
AABAAAAAABBBBAAAABBBBBBAAAABAAABBBABABBBBBBAAAAAAAABBBBBBBBBBAAAAAAAABBBBBBBBBBBAABBBBABAAAAAABBBBBBBBBBBBBB
BBBBAAABBBBAAAAAAAAAABBBBBBAAAAAAAAAABBAAAAAAAAABBBAAAAAAABBBAABBBBBBBBBBBBBBBBBBBBBBBBBAAAABBAAAABBBBBAABAAAAAABBABB
BAABBAAAABAABABBAAAAAAAAAAABBBBBBBBAABBAAAABBBAAABBAAABBBBBBBBABBBBAAAAAAAAAAABBBAAABAAABBMachine halting!

Ticks: total 283093, idle 185260, system 66970, user 30863
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 1951
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
nam@ubuntu:~/Desktop/hdh/Nachos_Project1/nachos/nachos-3.4/code$
```

THAM KHẢO

- Slide seminar project 3.

- Các lớp được cung cấp sẵn trong shandle.

- Video hướng dẫn đồ án của anh Nguyễn Thành Chung.