TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN

HỆ ĐIỀU HÀNH

BÁO CÁO ĐỒ ÁN 3

GVHD: Lê Viết Long

MSSV	Họ và tên
19120472	Nguyễn Văn Tuấn Đạt
19120490	Phạm Hải Dương
19120493	Hồ Đắc Duy
19120590	Huỳnh Thanh Mỹ
19120677	Nguyễn Diệp Minh Tiến

I. Bảng phân công công việc:

Sinh viên	Nội dung
Phạm Hải Dương	- Tìm hiểu thuộc tính và phương thức của lớp PTable. - Viết báo cáo mô tả lớp PTable.
Nguyễn Văn Tuấn Đạt	- Tìm hiểu thuộc tính và phương thức của lớp Pcb. - Viết báo cáo mô tả lớp Pcb.
Huỳnh Thanh Mỹ	- Tìm hiểu thuộc tính và phương thức của lớp Thread. - Viết báo cáo mô tả lớp Thread.
Hồ Đắc Duy	- Code SC_Exec, addrspace.cpp, SC_Join. - Viết báo cáo mô tả quá trình trên.
Nguyễn Diệp Minh Tiến	- Tìm hiểu thuộc tính và phương thức của lớp Bitmap. - Viết báo cáo mô tả lớp Bitmap.

II. PTable: Là một lớp hoạt động như một bảng quản lý tiến trình.

A. Thuộc tính:

1. **BitMap *bm:** Đối tượng BitMap trong lớp PTable.
2. **PCB *pcb[MAXPROCESS] :**Mảng quản lý các tiến trình.
3. **int psize:** Kích thước size của BitMap.
4. **Semaphore *bmsem:** Đối tượng Semaphore quản lý đồng bộ hóa.

B. Phương thức:

1. **PTable(int size):**

   Chức năng: Khởi tạo bảng quản lý các tiến trình cho hệ thống.

Phương thức nhận vào size để tạo một BitMap bm mới, đồng thời khởi tạo một Semaphore bmsem. Sau đó lặp qua mảng pcb để khởi tạo NULL cho từng tiến trình.

2. **int ExecUpdate(char* filename):**

   Chức năng: Thực hiện cập nhật.

   Phương thức nhận vào tên file (filename), sau đó tạo một đối tượng file thực thi mới bằng tên file, nếu thành công thì tiếp tục, ngược lại báo không tìm thấy file và trả về -1.

   Phương thức tiếp tục kiểm tra chương trình được gọi có khác chính nó không, nếu là chính nó thì báo lỗi và trả về -1. Sau đó phương thức kiểm tra còn slot trống hay không, nếu không còn thì báo lỗi, trả về -1, ngược lại cấp phát một khối tiến trình mới và thực thi. Cuối cùng trả về pID nếu thực thi thành công.

3. **int ExitUpdate(int ec):**

   Chức năng: Dừng một tiến trình.

   Phương thức nhận vào exit code. Kiểm tra tiến trình có tồn tại hay không, nếu tồn tại thì tiếp tục, ngược lại báo lỗi và trả về -1. Sau khi biết tiến trình đang tồn tại, nếu đó là tiến trình chính (pID = 0) thì Halt hệ thống, trả về 0, ngược lại xóa tiến trình đó ra khỏi mảng quản lý.

4. **int JoinUpdate(int pID):**

   Chức năng: Hỗ trợ cho ExitUpdate() trong việc dừng một tiến trình.

   Phương thức nhận vào ID của tiến trình, tiến hành kiểm tra tiến trình cũng như ID có tồn tại hay không.

   Tiếp tục kiểm tra tiến trình đang chạy phải là cha của tiến trình cần join, nếu vi phạm thì báo lỗi và trả về -1.

   Sau đó gọi hàm JoinWait() đợi tiến trình con kết thúc, nếu kết quả trả về khác 0 thì gọi hàm ExitRelease() để cho phép tiến trình con kết thúc

5. **int GetFreeSlot():**

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

Phương thức này lấy kết quả từ phương thức FindFreeSlot() của lớp BitMap, thông qua thể hiện bm.

6. **void Remove(int pID):**

   Chức năng: Xóa một tiến trình ra khỏi mảng quản lý.

   Phương thức nhận vào ID của tiến trình, kiểm tra xem ID có hợp lệ hay không ($0 \leq ID \leq 9$), nếu hợp lệ thì xóa và giải phóng tiến trình có ID đó ra khỏi mảng quản lý.

III. Pcb: Là một lớp lưu thông tin để quản lí tiến trình.

A. Thuộc tính:

1. **Semaphore* joinsem:** quản lí quá trình join.
2. **Semaphore* exitsem:** quản lí quá trình exit.
3. **Semaphore* multex:** quản lí quá trình truy xuất độc quyền.
4. **int exitcode:** exit code của thread con được trả về.
5. **int numwait:** lưu số tiến trình đã join.
6. **char FileName[32]:** lưu tên của tiến trình.
7. **Thread* thread:** lưu tiến trình được nạp.
8. **int parentID:** id của tiến trình cha.

B. Phương thức:

1. **PCB(int id):**

   Chức năng: Khởi tạo các biến của lớp.

   Nhận vào 1 id kiểm tra id, gán pid / numwait / exitcode / thread, khởi tạo joinsem / exitsem / mutex

2. **~PCB():**

   Chức năng: Hàm hủy của lớp.

3. **int GetID()**:

Chức năng: Trả về id thread mà lớp PCB đang quản lí.

4. **int GetNumWait():**

   Chức năng: Trả về numwait.

5. **int GetExitCode():**

   Chức năng: Trả về exitcode.

6. **void SetExitCode(int ec):**

   Chức năng: cập nhật exitcode = ec.

7. **int Exec(char *filename, int pID):**

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

   Thực thi 1 thread với tên của thread đó và id của threa cha. Kiểm tra thread đã khởi tạo thành công chưa, thất bại trả về -1. Gọi hàm thực thi Fork() và trả về id của thread.

8. **void JoinWait():**

   Chức năng: Chuyển tiến trình sang block. Đợi JoinRelease tiếp tục thực thi.

   Gọi joinsem->P() để tiến trình chuyển sang trạng thái block và ngừng lại, chờ JoinRelease để thực hiện tiếp.

9. **void ExitWait():**

   Chức năng: Để quá trình dịch sang trạng thái block. Đợi Exit Release để tiếp tục thực thi.

   Gọi exitsem-->P() để tiến trình chuyển sang trạng thái block và ngừng lại, chờ ExitReleaseđể thực hiện tiếp.

10. **void JoinRelease():**

    Chức năng: Gọi quá trình JoinWait.

    Gọi joinsem->V() để giải phóng tiến trình gọi JoinWait().

11. **void ExitRelease():**

    Chức năng: Giải phóng tiến trình chờ.

    Gọi exitsem-->V() để giải phóng tiến trình đang chờ.

12. **void IncNumWait():**

    Chức năng: Tăng số thread con.

    Dùng multex->P() và multex->V() để thực hiện truy xuất độc quyền.

13. **void DecNumWait():**

    Chức năng: Giảm số thread con.

    Dùng multex->P() và multex->V() để thực hiện truy xuất độc quyền.

14. **void SetFileName(char* fn):**

    Chức năng: cập nhật lại FileName = fn.

15. **void GetFileName():**

    Chức năng: trả về chuỗi chứa tên thread.

IV. 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.

A. Thuộc tính:

1. **#define BitsInByte 8:** Biểu diễn bitmap dưới dạng 1 mảng 8 số nguyên
2. **#define BitsInWord 32:** Biểu diễn bitmap dưới dạng 1 mảng 32 số nguyên
3. **int numBits:** Số lượng bits trong bitmap
4. **int numWords:** Số lượng Words lưu trữ bitmap (Làm tròn nếu numBits không phải là bội số của số bits trong 1 word)
5. **unsigned int *map:** Lưu trữ bit

B. Phương thức:

1. **BitMap(int nitem):**

   Chức năng: Khởi tạo 1 bitmap với nitem bits rỗng.

2. **void Mark(int which):**

   Chức năng: Đánh dấu bit thứ which.

3. **void Clear(int which):**

   Chức năng: Xóa bit thứ which.

4. **void Test(int which):**

   Chức năng: Kiểm tra bit thứ which đã được đặt chưa.

5. **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 đều đã đánh dấu thì trả về -1.

6. **int NumClear():**

   Chức năng: Trả về số bit chưa đánh dấu (Tìm bao nhiêu bits chưa đánh dấu)

7. **void Print():**

Chức năng: In nội dung của bitmap, dễ gỡ lỗi - Chỉ in những bit thứ n mà đã được đánh dấu

8. **void FetchFrom(OpenFile *file):**

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

9. **void WriteBack(OpenFile *file):**

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

V. Thread:

A. Thuộc tính:

1. **int* stackTop:** lưu vị trí đầu tiên
2. **int machineState[MachineStateSize]:** khởi tạo mảng machineState với độ dài bằng MachineStateSize
3. **int processID:** lưu ID của Threads
4. **int exitStatus:** lưu trạng thái đã thoát hay chưa
5. **int * stack:** lưu vị trí cuối cùng
6. **ThreadStatus status:** trạng thái hiện tại của Thread bao gồm: ready, running, blocked, just_created
7. **char* name:**
8. **int userRegisters[NumTotalRegs]:** khởi tạo mảng userRegisters với độ dài bằng NumTotalRegs
9. **AddrSpace *space:** mã của user đang chạy

B. Phương thức:

1. **Thread(char* debugName):**

   Chức năng: tạo một thread mới

   Theard đặt tên theo biến debugName.

   Trạng thái lúc khởi tạo đặt là JUST_CREATED

   stackTop và stack được gán giá trị NULL

2. **void FreeSpace():**

   Chức năng: giải phóng vùng nhớ

3. **void Yield():**

   Chức năng: nhường CPU cho một trong readylist

   Khởi tạo một Thread mới dùng để lấy CPU hiện tại

   Tạm dừng Thread đang sử dụng và nhường CPU cho thread vừa tạo nếu không có Thread sẵn sàng thì tiếp tục chạy Thread hiện tại

4. **void Sleep():**

   Chức năng: Chuyển tiến trình sang trạng thái ngủ(BLOCKED)

   Hoãn Thread hiện tại, gán status của nó bằng BLOCKED và loại bỏ khỏi danh sách sẵn sàng

   Nếu danh sách trống, interrupt->Idle() gọi đến để chờ interrupt kế tiếp và các Thread bị khóa có thể dùng lại nếu cần.

5. **void Fork(VoidFunctionPtr func, int arg):**

   Chức năng: khởi tạo một tiểu tiến trình:

   Trong đó, func là địa chỉ của tiến trình khi Thread thực hiện, arg được gán vào thread mới.

   Phân bổ bộ nhớ stack cho func với arg bằng hàm StackAllocate(VoidFunctionPtr func, int arg)

   scheduler->ReadyToRun(this) sẽ giả định đang vô hiệu hóa.

6. **void Finish():**

Chức năng: kết thúc Thread đang chạy

Hàm Finish trỏ biến toàn cục threadToBeDestroyed đến Thread hiện tại

Sau đó gọi hàm Sleep để khóa Thread đó lại và khi một Thread mới khác chạy thì threadToBeDestroyed sẽ xóa đi.

7. **void CheckOverflow():**

   Chức năng: kiểm tra stack của Thread có vượt quá giới hạn hay không

8. **void setStatus(ThreadStatus st):**Chức năng: thay đổi status của Thread
9. **char* getName():**

   Chức năng: trả về tên của Thread

10. **void Print():**

    Chức năng: in ra màn hình tên của Thread

11. **void StackAllocate(VoidFunctionPtr func, int arg):**

    Chức năng: cấp phát stack và phân bổ ngăn xếp cho Thread, được sử dụng trong Fork()

    Cấp phát bộ nhớ cho stack với StackSize

    Đặt giá trị kiểm tra trên đầu stack. Nếu một Thread bị tràn stack, thì Scheduler sẽ kiểm tra giá trị có thay đổi hay không.

    Đưa stack trỏ đến ThreadRoot. Các công việc sẽ bắt đầu từ đây thay vì tại user-supplied

12. **void SaveUserState():**

Chức năng: lưu trạng thái thanh ghi

Sử dụng vòng lặp và gán từng phần tử của mảng userRegisters với trạng thái thanh ghi thu được từ machine->ReadRegister(i)

13. **void RestoreUserState():**

Chức năng: khôi phục trạng thái thanh ghi

Sử dụng vòng lặp và gọi hàm machine->WriteRegister để ghi lại dữ liệu từ mảng userRegisters

VI. Báo cáo code:

A. Các hàm viết thêm:

1. addrspace.cpp:

- Sửa pageTable[i].physicalPage = i; thành pageTable[i].physicalPage = gPhysPageBitMap->Find();
- Hàm tính số trang còn trống:

```
int numclear = gPhysPageBitMap->NumClear();
 if(numPages > numclear)
  {
        printf("\nAddrSpace::Load : not enough memory for new process");
        numPages = 0;
        delete executable;
        addrLock->V();
  }
```

2. SC_Exec:

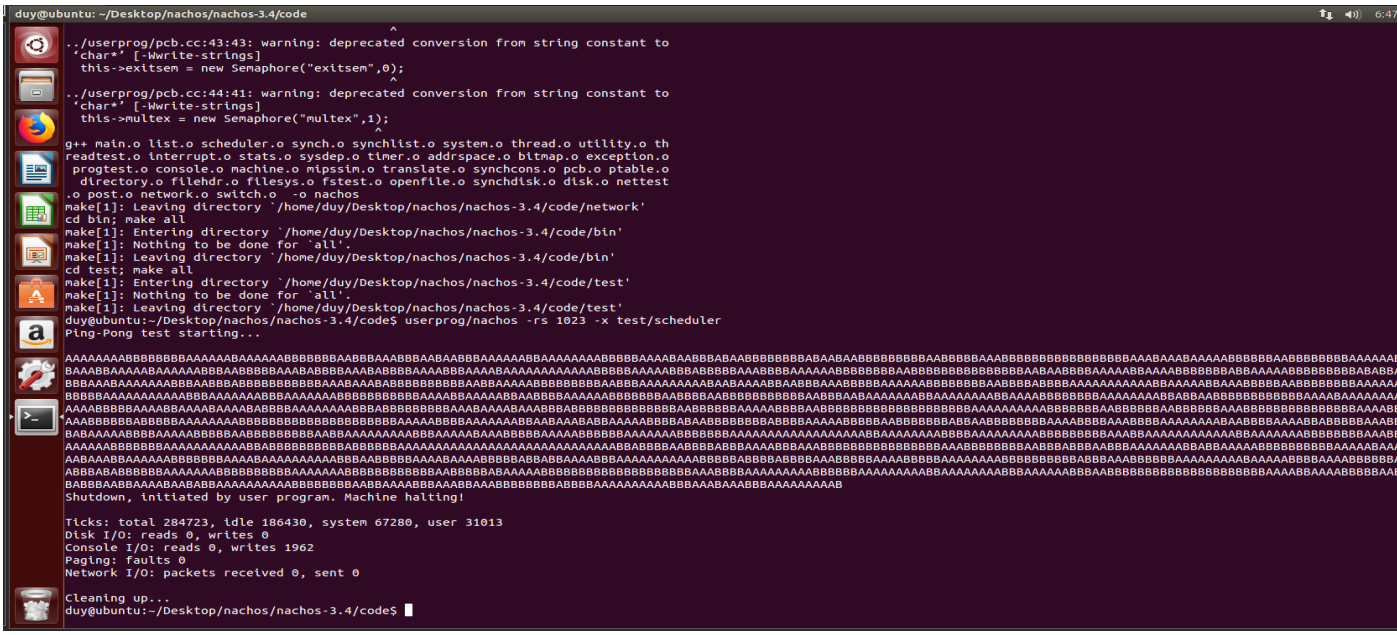
- Đọc địa chỉ tên chương trình “name” từ thanh ghi r4.
- Tên chương trình lúc này đang ở trong user space. Gọi hàm User2System đã được khai báo trong lớp machine để chuyển vùng nhớ user space tới vùng nhớ system space.
- Nếu bị lỗi không đủ bộ nhớ(name = null) thì báo “ Not

enough memory in System" và gán -1 vào thanh ghi 2 và kết thúc hàm.

- Nếu bị lỗi không mở được file thì báo "Exec:: Can't open this file." và gán -1 vào thanh ghi 2 và kết thúc hàm.
- Nếu không có lỗi thì gọi pTab.ExecUpdate(name), trả về và lưu kết quả thực thi phương thức này vào thanh ghi r2.

3. **SC_Join:**
   - Đọc địa chỉ id của thread từ thanh ghi số r4.
   - Gọi res = pTab->JoinUpdate(id) để tiến hành kiểm tra tiến trình cũng như ID có tồn tại hay không?
   - Sau đó lưu kết quả res vào thanh ghi số 2.

B. <u>Ảnh chụp kết quả:</u> Gọi lệnh userprog/nachos -rs 1023 -x test/scheduler



VII. <u>Tài liệu tham khảo:</u>

- Tài liệu thầy cung cấp trên moodle.
- https://www.youtube.com/playlist?list=PLRgTVtca98hUgCN2_2vzsAAXPiTFbvHpO

---------- Hết ----------