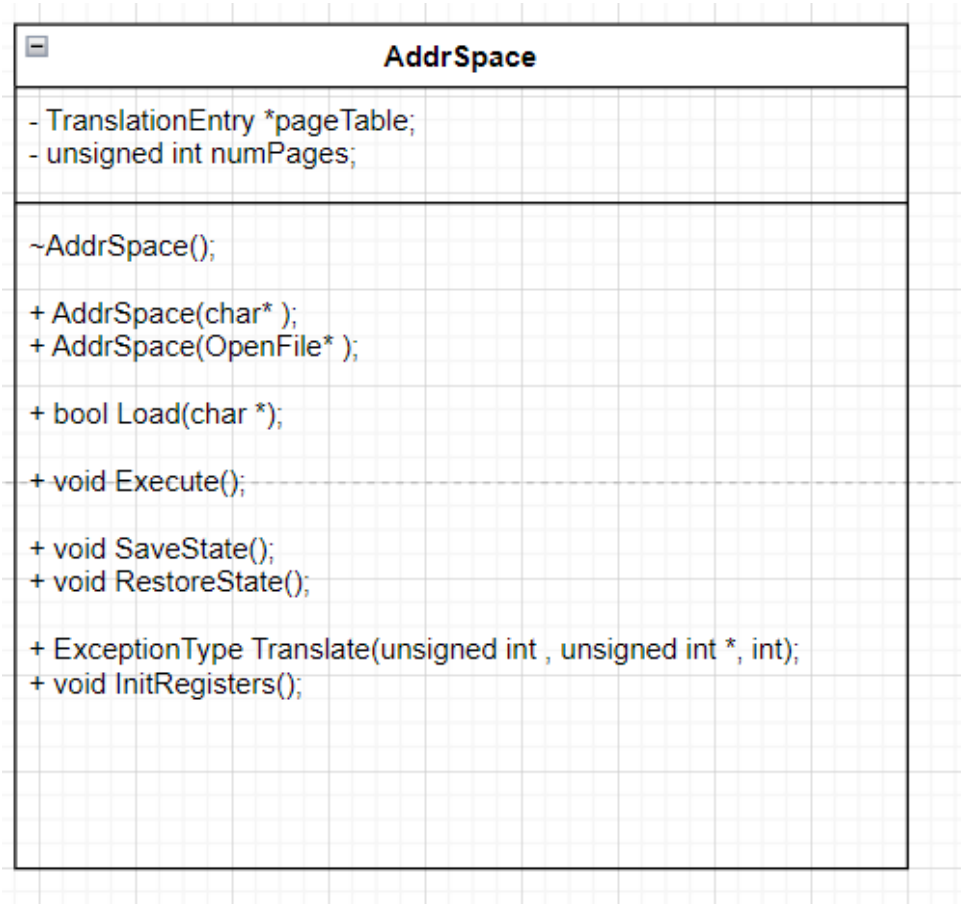
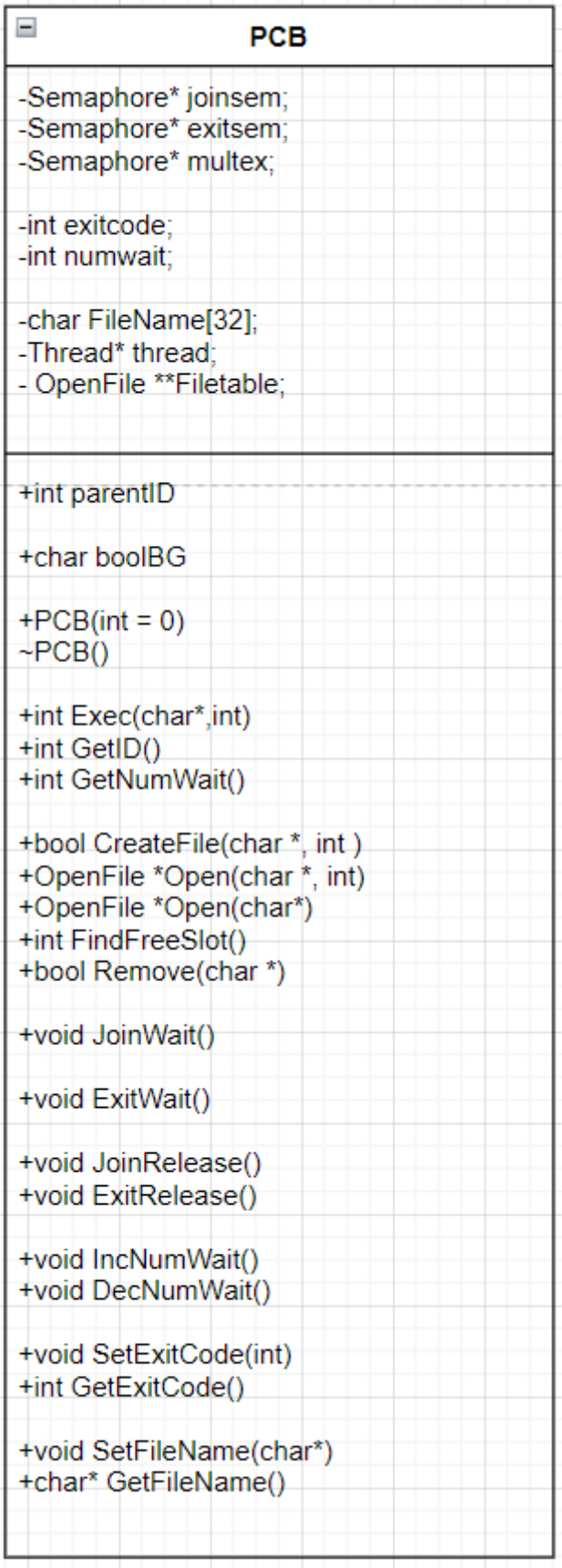
Báo cáo đồ án 2

1. Mô tả các thiết kế

- Sửa đổi file addrspace.cc trong thread



- Tạo class PCB trong pcb.cc và pcb.h ở thread. Class này lưu các thông tin để quản lý một process. Thiết kế lớp PCB như sau:


PCB
-Semaphore* joinsem;
-Semaphore* exitsem;
-Semaphore* multex;
-int exitcode;
-int numwait;
-char FileName[32];
-Thread* thread;
- OpenFile **Filetable;
+int parentID
+char boolBG
+PCB(int = 0)
~PCB()
+int Exec(char*,int)
+int GetID()
+int GetNumWait()
+bool CreateFile(char *, int)
+OpenFile *Open(char *, int)
+OpenFile *Open(char*)
+int FindFreeSlot()
+bool Remove(char *)
+void JoinWait()
+void ExitWait()
+void JoinRelease()
+void ExitRelease()
+void IncNumWait()
+void DecNumWait()
+void SetExitCode(int)
+int GetExitCode()
+void SetFileName(char*)
+char* GetFileName()

Trong đó:

joinsem: semaphore cho quá trình join

exitsem: semaphore cho quá trình join

multex: semaphore cho quá trình truy xuất độc quyền

numwait: số tiến trình đã join

parentID: ID của tiến trình cha

fileTable: quản lý các file đang mở (Số lượng tối đa là 10)

Exec: Tạo 1 thread mới

GetID: Trả về process ID của tiến trình đang thực hiện

GetNumwait: Trả về số lượng tiến trình chờ

CreateFile: Tạo 1 file mới

Open: Mở file

JoinWait: Tiến trình cha đợi tiến trình con kết thúc

ExitWait: Tiến trình con kết thúc

JoinRelease: Báo cho tiến trình cha khi thực thi tiếp

ExitRelease: Cho phép tiến trình con kết thúc

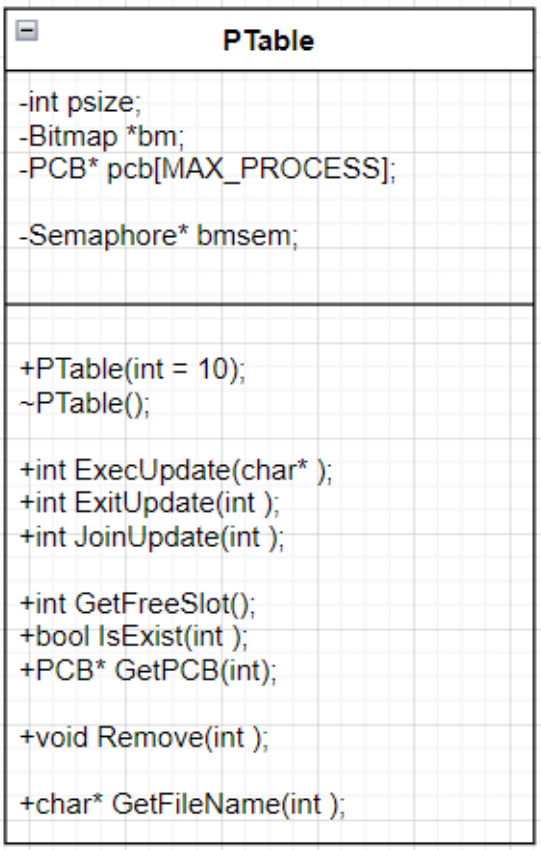
SetExitCode: Đặt exit code của tiến trình

GetExitCode: Trả về exit code

SetFileName: Set tên tiến trình

GetFileName: Get tên tiến trình

- Tạo class PTable trong ptable.cc và ptable.h ở thư mục thread. Class này để quản lý các tiến trình đang chạy. Thiết kế lớp PTable như sau:



Trong đó:

pcb: Là mảng mô tả các tiến trình. Trong đồ án này, mảng quản lý 10 tiến trình, tiến trình đầu tiên (tiến trình cha) ở vị trí thứ 0

bm: Đánh dấu các vị trí đã được sử dụng trong pcb.

bmsem: dùng để ngăn chặn tình trạng nạp 2 tiến trình cùng lúc

ExecUpdate: Xử lý cho system call SC_Exit

JoinUpdate: Xử lý cho system call SC_Join

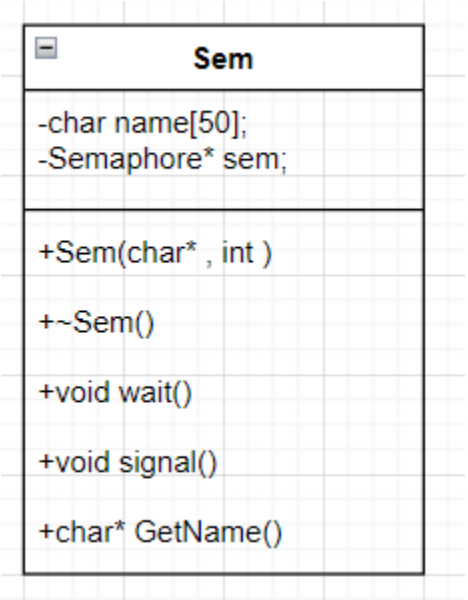
GetFreeSlot: Tìm slot trống để lưu thông tin cho tiến trình mới

IsExit: Kiểm tra sự tồn tại của 1 process bằng ID

Remove: Khi kết thúc tiến trình, delete processID ra khỏi mang đang quản lý

GetFileName: Trả về tên của tiến trình

- Tạo class Sem và class STable trong stable.cc và stable.h ở thư mục thread. Class Sem quản lý một semaphore, class STable quản lý các semaphore. Thiết kế lớp Sem như sau



Trong đó:

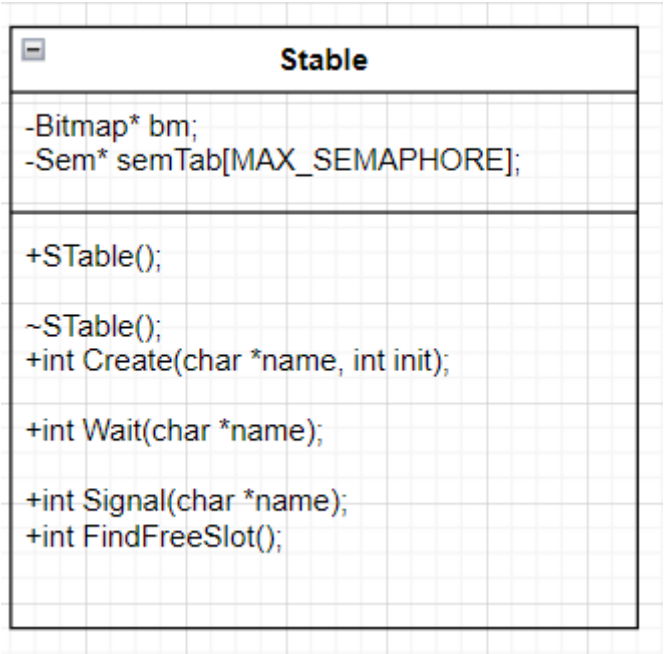
sem: Tạo semaphore để quản lý

wait: Thực hiện thao tác chờ

signal: Thực hiện thao tác giải phóng semaphore

GetName: Trả về tên của semaphore

- Thiết kế lớp STable như sau:



Trong đó:

bm: quản lý slot trống

semTab: mảng quản lý tối đa 10 đối tượng Sem

Create: Kiểm tra Semaphore “name” nếu chưa tồn tại thì tạo, ngược lại thì báo lỗi

Wait: Nếu tồn tại Semaphore “name” thì gọi hàm thực thi, ngược lại thì báo lỗi

Signal: Nếu tồn tại Semaphore “name” thì gọi hàm thực thi, ngược lại thì báo lỗi

FindFreeSlot: Tìm slot trống

2. Cài đặt các system call nhập xuất file

2.1. Cài đặt System Call: OpenFileID Open(char *name, int type) và int Close(OpenFileID id)

- User program có thể mở 2 loại file, file chỉ đọc và file đọc và ghi. Mỗi tiến trình sẽ được cấp một bảng mô tả file với kích thước cố định. Kích thước của bảng mô tả file có thể lưu được đặc tả của 10 files. Trong đó, 2 phần tử đầu, ô 0 và ô 1 để dành cho console input và console output.
- System call mở file phải làm nhiệm vụ chuyển đổi địa chỉ buffer trong user space khi cần thiết và viết hàm xử lý phù hợp trong kernel. Dùng class PTable để gọi xuống class PCB, System call Open sẽ trả về id của file (OpenFileID = một số nguyên), hoặc là -1 nếu quá trình mở file bị lỗi
- Mở file có thể bị lỗi như trường hợp là không tồn tại tên file hay không đủ ô nhớ trong bảng mô tả file. Tham số type = 0: mở file đọc và ghi, type = 1: chỉ đọc, type = 2: chạy stdin, type = 3: chạy stdout. Nếu tham số truyền bị sai thì system call phải báo lỗi. System call sẽ trả về -1 nếu bị lỗi và 0 nếu thành công.
- **Open(char *name, int type).**

  • Quy ước giá trị của type:

  ▪ Type =0: đọc và ghi.

  ▪ Type =1: chỉ đọc.

  ▪ Type =2: stdin.

▪ Type =3: stdout.

- Mô tả cài đặt **SC_Open**:

▪ Input: Địa chỉ của tên file trên user space, chế độ mở (type).

▪ Output: id file nếu thành công, -1 nếu lỗi

▪ Mục đích: mở một file với tham số số truyền vào gồm tên file và cách mở file.

Ta đọc địa chỉ của tham số name từ thanh ghi r4 và tham số type từ thanh ghi r5 sau đó kiểm tra type có hợp lệ hay không (0<= type <=3), kiểm tra index của file trong lớp PCB của có nằm trong bảng mô tả file của tiến trình hiện tại hay không. Nếu kiểm tra hai điều kiện trên hợp lệ thì thực hiện chép giá trị ở r4 từ phía User sang System bằng hàm `User2System()`. Giá trị chép được thực sự chính là tên file. Ta tiếp tục kiểm tra tên file, nếu là stdin và type truyền vào là 2 thì trả về cho thanh ghi r2 id của file là 0, nếu là stdout và type truyền vào là 3 thì trả về cho thanh ghi r2 id của file là 1, nếu là file bình thường thì type phải khác 2 và 3 và trả về cho thanh ghi r2 đúng id của file bằng (index – 1) của lớp PCB của tiến trình hiện tại. Trường hợp mở file không tồn tại hoặc ngược lại với tất cả điều kiện trên thì trả về -1 cho thanh ghi r2

- **int Close(OpenFileID id).**
- Mô tả cài đặt SC_Close:

▪ Input: ID file.

▪ Output: NULL.

▪ Mục đích: Đóng file với tham số truyền vào là ID của file

Đọc tham số id của file từ thanh ghi r4, sau đó kiểm tra xem file cần đóng có tồn tại không bằng fileTable trong PCB và kiểm tra id của file có nằm ngoài bảng mô tả file không. Nếu có một trong hai lỗi trên thì xuất ra thông báo lỗi và gọi syscall `Halt()`

để tắt hệ thống, nếu thành công thì xóa đi dữ liệu `fileTable[id]` và gán lại bằng NULL

2.2. Cài đặt System Call: int Read (char* buffer, int charcount, OpenFileIDid) và int Write (char* buffer, int charcount, OpenFileID id)

Các System call đọc và ghi vào file với id cho trước. Phải chuyển vùng nhớ giữa user space và system space, cần phân biệt giữa Console IO (OpenFileID 0, 1) và File. Lệnh Read và Write sẽ làm việc như sau: Phần console read và write sẽ sử dụng lớp `SynchConsoleInput` và `SynchConsoleOutput`. Sử dụng các hàm mặc định của `SynchConsoleInput` và `SynchConsoleOutput` để đọc và ghi. Đọc và ghi với Console sẽ trả về số bytes đọc và ghi thật sự, chứ không phải số bytes được yêu cầu. Trong trường hợp đọc hay ghi vào console bị lỗi thì trả về -1. Nếu đang đọc từ console và chạm tới cuối file thì trả về -2. Đọc và ghi vào console sẽ sử dụng dữ liệu ASCII để cho input và output, (ASCII dùng kết thúc chuỗi là NULL (‘\0’)). Phần đọc, ghi vào file sẽ sử dụng các hàm được cài đặt trong lớp PCB và quản lý bằng fileTable được xử lý trong class PCB. Cả read và write trả số kí tự đọc, ghi thật sự. Cả Read và Write trả về -1 nếu bị lỗi và -2 nếu cuối file. Cả Read và Write sử dụng dữ liệu binary.

2.3. int Read (char* buffer, int charcount, OpenFile id).

Mô tả cài đặt SC_Read:

- Input: Buffer, số ký tự cho phép, id của file.
- Ouput: -1 nếu lỗi, -2 nếu thành công với số byte được đọc.
- Mục đích: Đọc file với tham số là buffer, số ký tự cho phép (charcount) và id của file.

Ta đọc địa chỉ của tham số buffer từ thanh ghi r4, tham số charcount từ thanh ghi r5 và id của file từ thanh ghi r6, sau đó ta tiến hành kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không, file cần đọc có tồn tại không và file cần đọc có phải là stdout với type = 3 không. Nếu vi phạm các điều kiện trên thì trả về -1 cho thanh ghi r2 ngược lại là hợp lệ thì lấy vị trí con trỏ ban đầu trong file bằng phương thức

`GetCurrentPos()` của lớp OpenFile gọi là OldPos và thực hiện chép giá trị ở r4 từ phía User sang System bằng hàm `User2System()`. Giá trị chép được là buffer chứa chuỗi kí tự. Xét trường hợp đọc file stdin với type = 2, ta sử dụng phương thức getChar của SynchConsoleInput để xử lý để đọc buffer với độ dài charcount, trả về số byte thực sự đọc được cho thanh ghi r2 và chép buffer từ phía System sang User bằng hàm `System2User()`. Xét trường hợp đọc file bình thường, thì ta lấy vị trí con trỏ hiện tại trong file bằng phương thức `GetCurrentPos()` của file hiện tại trong bảng FileTable gọi là NewPos, trả về số byte thực sự đọc được cho thanh ghi r2 bằng công thức: NewPos – OldPos và cũng chép buffer từ phía System sang User bằng hàm `System2User()`. Trường hợp còn lại là đọc file rỗng thì trả về -2 cho thanh ghi r2

2.4. int Write (char* buffer, int charcount, OpenFileID id).

Mô tả cài đặt SC_Write:

▪ Input: Buffer, số ký tự cho phép, id của file.

▪ Ouput: -1 nếu lỗi, Số byte thực sự ghi được nếu thành công.

▪ Mục đích: Ghi file với tham số là buffer, số ký tự cho phép (charcount) và id của file.

Ta đọc địa chỉ của tham số buffer từ thanh ghi r4, tham số charcount từ thanh ghi r5 và id của file từ thanh ghi r6, sau đó ta tiến hành kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không, file cần ghi có tồn tại không và file cần ghi có phải là stdin với type = 2 hay là file chỉ đọc với type = 1. Nếu vi phạm các điều kiện trên thì trả về -1 cho thanh ghi r2 ngược lại là hợp lệ thì lấy vị trí con trỏ ban đầu trong file bằng phương thức `GetCurrentPos()` của lớp FileSystem gọi là OldPos và thực hiện chép giá trị ở r4 từ phía User sang System bằng hàm `User2System()`. Giá trị chép được là buffer chứa chuỗi kí tự. Xét trường hợp ghi file đọc và ghi với type = 0, thì ta lấy vị trí con trỏ hiện tại trong file bằng phương thức `GetCurrentPos()` của lớp FileSystem gọi là NewPos, trả về số byte thực sự ghi được cho thanh ghi r2 bằng công thức: NewPos – OldPos. Xét trường hợp ghi file stdout với type = 3, ta gọi phương thức Write của lớp SynchConsole để ghi từng kí tự trong buffer và kết thúc là kí tự xuống dòng ‘\n’, trả về số byte thực sự ghi được cho thanh ghi r2.

2.5. Cài đặt System Call: int Seek (int pos, OpenFileID id)

Seek sẽ phải chuyển con trỏ tới vị trí thích hợp. Pos lưu vị trí cần chuyển tới, nếu pos = -1 thì di chuyển đến cuối file. Trả về vị trí thực sự trong file nếu thành công và -1 nếu bị lỗi. Gọi Seek trên console phải báo lỗi.

Mô tả cài đặt SC_Seek:

▪ Input: Vị trí cần chuyển tới, id của file.

▪ Output: -1: Lỗi, Vị trí thực sự trong file: Thành công.

▪ Mục đích: Di chuyển con trỏ đến vị trí thích hợp trong file với tham số là vị trí cần dịch chuyển và id của file.

Ta đọc tham số pos từ thanh ghi r4 và id của file từ thanh ghi r5, sau đó ta tiến hành kiểm tra id của file truyền vào có nằm ngoài bảng mô tả file không, file cần di chuyển con trỏ có tồn tại không và kiểm tra người dùng có gọi Seek trên console không. Nếu vi phạm các điều kiện trên thì trả về -1 cho thanh ghi r2 ngược lại là hợp lệ thì kiểm tra nếu pos = -1 thì gán pos bằng độ dài của file bằng phương thức `Length()` của lớp FileSystem. Gọi phương thức Seek của lớp FileSystem với tham số truyền vào là pos để dịch chuyển con trỏ đến vị trí mong muốn và trả về vị trí dịch chuyển cho r2

3. Đa chương, lập lịch và đồng bộ hóa trong NachOS

Chương trình hiện tại giới hạn chỉ thực thi 1 chương trình, cần có vài thay đổi trong file addrspace.h và addrspace.cc để chuyển hệ thống từ đơn chương thành đa chương:

- Giải quyết vấn đề cấp phát các frames bộ nhớ vật lý, sao cho nhiều chương trình có thể nạp lên bộ nhớ cùng một lúc: sử dụng biến toàn cục **`Bitmap *gPhysPageBitMap`** để quản lý các frame
- Xử lý giải phóng bộ nhớ khi user program kết thúc.
- Thay đổi đoạn lệnh nạp user program lên bộ nhớ. Hiện tại, việc cấp phát không gian địa chỉ giả thiết rằng một tiến trình được nạp vào các đoạn liên tiếp nhau trong bộ nhớ. Một khi hỗ trợ đa chương trình, bộ nhớ sẽ không còn biểu diễn liên

tiếp nhau nữa: tạo **`pageTable = new TranslationEntry[numPages]`**, tìm trang trống và sau đó nạp lên bộ nhớ chính

3.1. System call SpaceID Exec(char* name)

System call này gọi thực thi một chương trình mới trong một system thread mới.
Ta cần thực hiện các bước sau:

- Cài đặt phương thức **`Exec(char* name, int id)`** của lớp PCB:
- Gọi `multex->P()`: tránh tính trạng nạp 2 tiến trình cùng lúc
- Kiểm tra thread mới tạo có thành công không, nếu không thì thông báo lỗi không đủ bộ nhớ, gọi `multex->V()` và trả về -1.
- Ngược lại, nếu khởi tạo thành công thì đặt processId của thread này là id
- Đặt parrentId của thread này là processId của thread gọi thực thi hàm Exec
- Gọi `Fork(StartProcess_2, id)`. Hàm `StartProcess_2` được cài đặt trong pcb.cc
- Gọi `multex->V()` và trả về id.
- Cài đặt phương thức **`ExecUpdate (char* name)`** của lớp PTable:
- Gọi `bmsem->P()`: tránh tính trạng nạp 2 tiến trình cùng lúc
- Nếu chương trình "name" không hợp lệ, hoặc không tồn tại, gọi `bmsem->V()` và trả về -1
- Nếu tên chương trình trùng với tên của currentThread (gọi thực thi chính nó) hoặc trùng với tên của tiến trình ban đầu, gọi `bmsem->V()` và trả về -1
- Ngược lại, nếu không xảy ra hai trường hợp trên thì tìm chỗ trống trong bảng pTab.
- Nếu không tìm thấy, gọi `bmsem->V()` và trả về -1
- Nếu tìm thấy (idx), đánh dấu vị trí đó đã sử dụng. Khởi tạo PCB mới có id là idx, chỗ trống tìm thấy, parentID là processId của currentThread
- Gọi `bmsem->V()` và trả về kết quả của `Exec(name, idx)`

Nếu có lỗi, system call trả về -1. Ngược lại, trả về processId của chương trình mới được tạo

Mô tả cài đặt System call:

- Đọc địa chỉ tên chương trình "name" trong thanh ghi r4

- Gọi thực thi hàm `User2System` để chuyển vùng nhớ chứa tên chương trình này (user space) sang kernel space
- Nếu bị lỗi (tên chương trình không tồn tại hoặc không mở được file) thì thông báo lỗi cho người dùng, ghi kết quả -1 vào thanh ghi r2 và kết thúc
- Ngược lại, gọi thực thi `pTab->ExecUpdate(name),` ghi kết quả vào thanh ghi r2 và trả về kết quả này

3.2. System call int Join(SpaceID id)

System call này sẽ đợi và block tiến trình dựa trên tham số id của tiến trình đó

Ta cần thực hiện các bước sau:

- Cài đặt phương thức **`JoinWait()`** ở lớp PCB: gọi `joinsem->P()`
- Cài đặt phương thức **`ExitRelease()`** ở lớp PCB: gọi `exitsem->V()`
- Cài đặt phương thức **`JoinUpdate(int id)`** ở lớp PTable

- Kiểm tra tiến trình đang gọi có trùng với cha của tiến trình có processId là id hay không. Nếu không, trả về -1 (vì không thể join vô tiến trình mà không phải cha của nó).
- Tăng số tiến trình chờ, tiến trình cha thực hiện `JoinWait()` để chờ tiến trình con thực hiện
- Xử lý exitcode
- Tiến trình cha thực hiện `ExitRelease()` cho phép tiến trình con thoát

Mô tả cài đặt System call:

- Đọc id của tiến trình cần Join từ thanh ghi r4
- Gọi `pTab -> JoinUpdate(id)`
- Lưu kết quả vào thanh ghi r2

3.3. System call void Exit(int exitcode)

System call thực hiện thoát một tiến trình nó đã join

Ta cần thực hiện các bước sau:

- Cài đặt phương thức **`JoinRelease()`** ở lớp PCB: Gọi `joinsem->V()`
- Cài đặt phương thức **`ExitWait()`** ở lớp PCB: Gọi `exitsem->P()`
- Cài đặt phương thức **`ExitUpdate(int exitcode)`** ở lớp PTable:

- Nếu tiến trình gọi là tiến trình main thì thực thi `Halt()`
- Ngược lại thì đặt exitcode cho tiến trình gọi

- Tiến trình con gọi `JoinRelease()` để giải phóng tiến trình cha đang đợi, báo cho tiến trình cha tiếp tục thực hiện
- Tiến trình con gọi `ExitWait()` kết thúc, xin phép tiến trình cha cho thoát

Mô tả cài đặt System call:

- Đọc exitcode từ thanh ghi r4
- Gọi `pTab -> JoinUpdate(id)`
- Lưu kết quả vào thanh ghi r2

Exitcode là 0 nếu quá trình thoát thực hiện thành công, mã lỗi trong các trường hợp còn lại.

Join system call trả về -1 nếu bị lỗi, ngược lại trả về exit code cho tiến trình đang bị block

3.4. System call CreateSemaphore

System call này được dùng để khởi tạo một Semaphore với đầu vào là tên và giá trị khởi tạo của Semaphore cần tạo. (trả về 0 nếu thành công, và -1 nếu thất bại)

- Khai báo prototype `CreateSemaphore(char* name, int semval)` trong ./userprog/syscall.h
- Cài đặt hàm `Create(char *name, int init)` ở lớp STable.

  Ở hàm này ta sẽ đi kiểm tra semaphore "name" có trong semTab hay không semTab còn vị trí trống không và khởi tạo Semaphore bằng `new sem(name, init)` ngược lại báo lỗi. (trả về 0 nếu thành công, và -1 nếu thất bại)

- Cài đặt hàm `SysCreateSemaphore(int virtAddr, int semval)` trong ./userprog/ksyscall.h.

  Hàm này sẽ kiểm tra và báo lỗi khi gọi hàm Create bên trên. Được gọi tại ./userprog/exception.cc

3.5. System call Wait

System call này giảm giá trị của một Semaphore đi một với đầu vào là tên Semaphore cần giảm. (trả về 0 nếu thành công, và -1 nếu thất bại)

- Khai báo prototype `Wait(char* name)` trong ./userprog/syscall.h
- Cài đặt hàm `Wait(char *name)` ở lớp STable.

  Ở hàm này ta sẽ đi tìm semaphore "name" trong semTab và gọi hàm `this->P()` ngược lại báo lỗi. (trả về 0 nếu thành công, và -1 nếu thất bại)

- Cài đặt hàm `SysWait(int virtAddr)` trong ./userprog/ksyscall.h.

  Hàm này sẽ kiểm tra và báo lỗi khi gọi hàm Wait bên trên. Được gọi tại ./userprog/exception.cc

3.6. System call Signal

System call này tăng giá trị của một Semaphore đi một với đầu vào là tên Semaphore cần tăng. (trả về 0 nếu thành công, và -1 nếu thất bại)

- Khai báo prototype `Signal(char* name)` trong ./userprog/syscall.h
- Cài đặt hàm `Signal(char *name)` ở lớp STable.

  Ở hàm này ta sẽ đi tìm semaphore "name" trong semTab và gọi hàm `this->V()` ngược lại báo lỗi. (trả về 0 nếu thành công, và -1 nếu thất bại)

- Cài đặt hàm `SysSignal(int virtAddr)` trong ./userprog/ksyscall.h.

  Hàm này sẽ kiểm tra và báo lỗi khi gọi hàm Signal bên trên. Được gọi tại ./userprog/exception.cc

4. Chương trình người dùng

Áp dụng các system call đã cài để giải quyết bài toán sau:

Trong một ký túc xá, có một vòi nước và n sinh viên, n sẽ input từ người dùng (n<5). Mỗi sinh viên sẽ cần lấy 10 lít nước từ vòi, nhưng ở mỗi lần lấy sinh viên chỉ có thể lấy tối đa 1 lít nước và phải trả vòi nước lại (sau khi trả vòi nước sinh viên có quyền yêu cầu sử dụng vòi nước ngay lập tức). Thời gian lấy một lít nước của mỗi sinh viên là ngẫu nhiên(có thể dùng vòng lặp for để mô phỏng thời gian đợi). Hãy viết một hệ thống đáp ứng được các yêu cầu trên.

Input: số lượng sinh viên (có thể đọc từ file hoặc nhập từ bàn phím)

Output: Thứ tự lấy nước của sinh viên cho đến khi sinh viên cuối cùng lấy được 10 lít nước

4.1. Thiết kế các file:

File sinhvien.c: Mô tả quá trình lấy nước của một sinh viên:

File voinuoc.c: Mô tả hệ thống

4.2. Cài đặt

File voinuoc.c: Mô phỏng hệ thống lấy nước, gồm các bước sau:

- Tạo file output.txt để lưu lại thứ tự các sinh viên lấy nước. Mỗi khi lấy nước, sinh viên sẽ thực hiện mở file này và ghi lại id của mình vào file.
- Tạo một vòi nước (là một semaphore) với giá trị khởi tạo ban đầu là 1: `Semaphore("voinuoc",1)` để cho biết rằng chỉ có duy nhất 1 sinh viên được lấy nước tại một thời điểm xác định
- Yêu cầu người dùng nhập số lượng sinh viên n
- Lặp qua n sinh viên, mỗi lần lặp gọi thực thi `Exec("./test/sinhvien")` để sinh viên lấy nước và lưu kết quả thực thi (id của sinh viên) lại vào file output.txt
- Gọi hàm `Join(id)` để các sinh viên cùng tham gia lấy vòi nước và đợi đến lượt mình lấy vòi nước

File sinhvien.c: Mô tả việc lấy nước của một sinh viên, xem sinh viên như một tiến trình, gồm các bước sau:

- Thực hiện mở file output.txt để ghi lại id của mình vào kết quả
- Lấy id của mình bằng cách gọi system call GetProcessId, trả về id của tiến trình hiện tại
- Theo bài toán, sinh viên i sẽ phải lấy nước 10 lần. Tiến hành một vòng lặp chạy 10 lần. Ở mỗi lần lặp đó:

- Gọi `Wait("voinuoc")` để giảm giá trị của semaphore. Nếu ban đầu giá trị của semaphore này bằng 1, tức là lúc đó vòi nước đang trống và sinh viên i có thể lấy nước. Ngược lại, vòi nước đang có sinh viên j nào đó khác xài và sinh viên i phải chờ
- Nếu sinh viên i có thể lấy được nước, tiến hành ghi lại id vào file output.txt và gọi `Signal("voinuoc")` để báo rằng mình đã dùng xong và trả lại vòi nước cho sinh viên khác xài (nếu có)

4.3. Hướng dẫn chạy chương trình

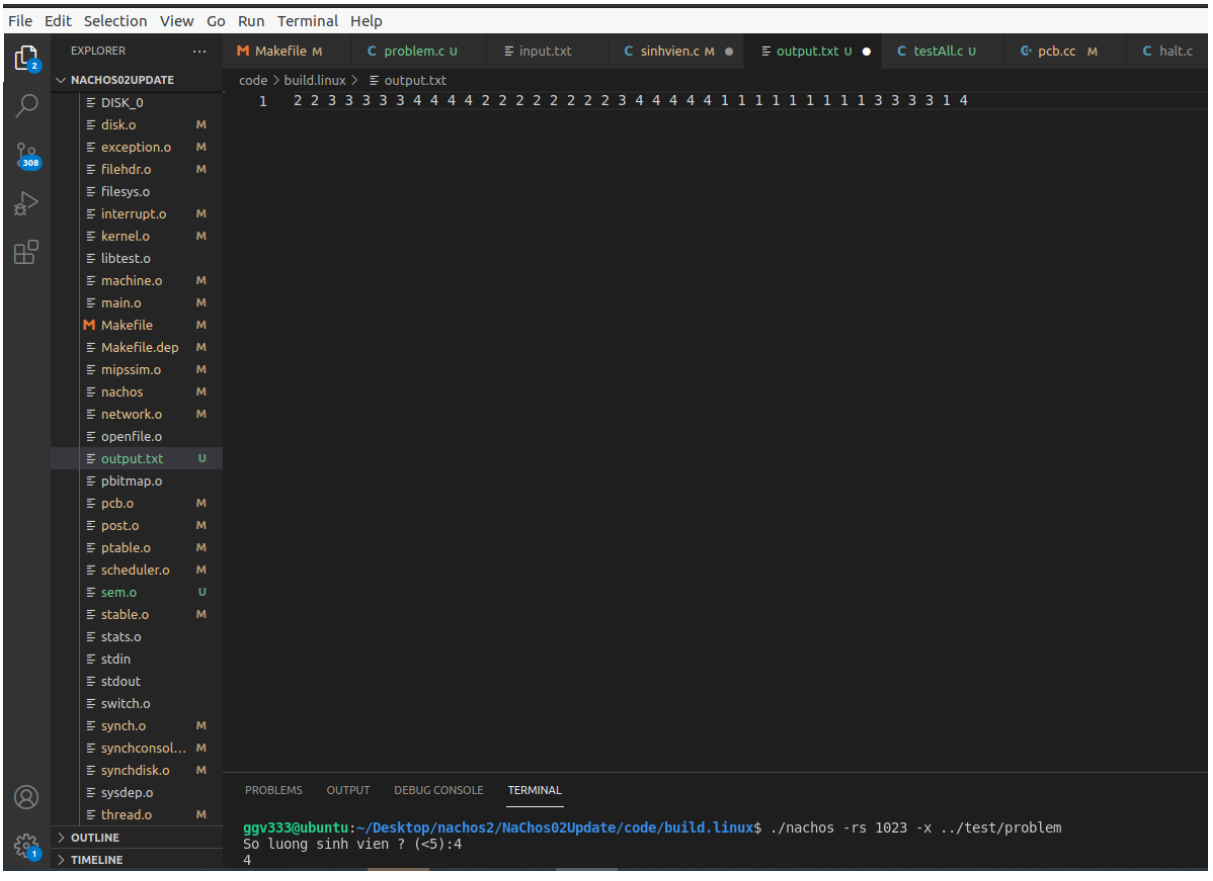
Giả sử đường dẫn hiện tại đang ở thư mục build.linux

Sau khi thực hiện make thành công, chạy lệnh

```
./nachos -rs 1023 -x ../test/problem
```

Nhập số lượng sinh viên vào từ bàn phím

Kết quả ở file output:



5. Phân công

Task	Tên	Tỉ lệ
System call nhập xuất file	Chung Hoàng Tuấn Kiệt	100%
Exec, Join, Exit	Vũ Hữu Nghĩa	100%
CreateSemaphore, Wait, Signal	Trần Huy Vũ	100%
Chương trình người dùng	Tất cả các thành viên	100%

6. Tài liệu tham khảo

- https://github.com/nguyenthanhchungfit/Nachos-Programing-HCMUS
- Tài liệu giảng viên cung cấp cho đồ án 2