

HỆ ĐIỀU HÀNH

Đồ án 2

System Call về File, Đa chương, Lập lịch và Đồng bộ

19120036 Nguyễn Đăng Tiến Thành

19120176 Lê Công Bình

19120200 Nguyễn Tam Dũng



Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên TP HCM
Tháng 1/2022

MỤC LỤC

A.	Thông tin nhóm.....	3
B.	Thiết kế	3
1.	Lớp AddrSpace	4
2.	Lớp PCB	4
3.	Lớp PTable	5
4.	Lớp Sem.....	7
5.	Lớp STable	7
C.	Cài đặt	8
1.	System call nhập xuất file.....	8
1.1.	int CreateFile(char *name).....	8
1.2.	OpenFileID Open(char *name, int type).....	8
1.3.	int Close(OpenFileID id).....	9
1.4.	int Read(char *buffer, int charcount, OpenFileID id)	10
1.5.	int Write(char *buffer, int charcount, OpenFileID id)	10
2.	Đa chương, lập lịch và đồng bộ hóa	11
2.1.	SpaceID Exec(char* name).....	11
2.2.	int Join(SpaceID id).....	12
2.3.	void Exit(int exitCode)	12
2.4.	int CreateSemaphore(char* name, int semVal)	13
2.5.	int Wait(char* name).....	13
2.6.	int Signal(char* name).....	13
D.	Hướng dẫn sử dụng chương trình	15
1.	Chương trình vòi nước	15
2.	Chương trình hai vòi nước	16
E.	Tài liệu tham khảo	18

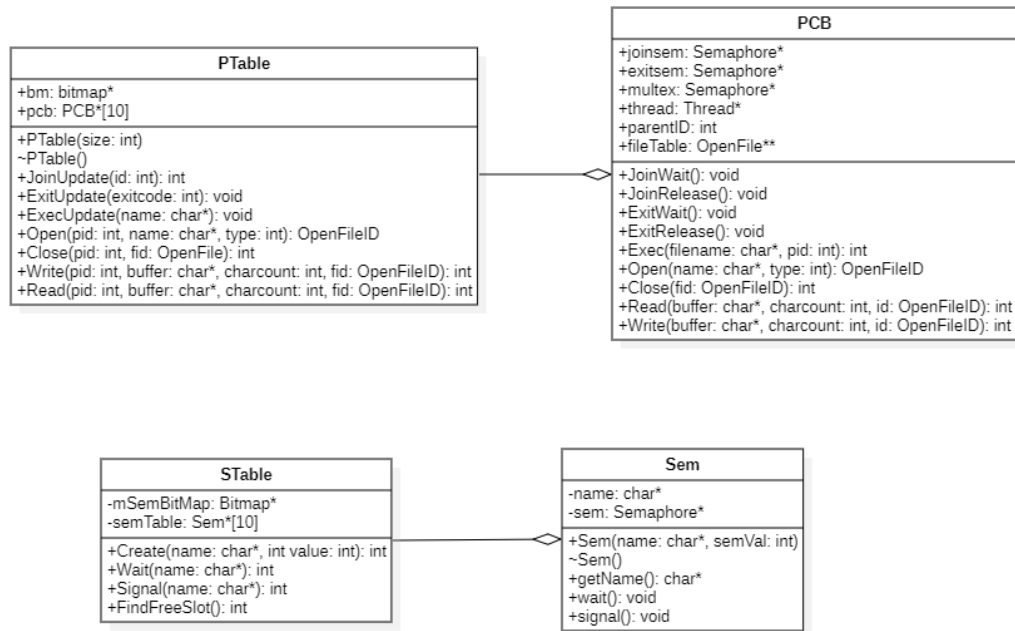
A. Thông tin nhóm

MSSV	Họ và tên	Mức độ đóng góp
19120036	Nguyễn Đăng Tiến Thành	100%
19120176	Lê Công Bình	100%
19120200	Nguyễn Tam Dũng	100%

Nhóm đã hoàn thành tất cả các yêu cầu của đồ án.

B. Thiết kế

Dưới đây là mô tả về thiết kế của các lớp, thuộc tính và hàm quan trọng đã được cài đặt phục vụ cho đọc ghi file và đa chương, lập lịch, đồng bộ.



(Sơ đồ lớp của các lớp quan trọng)

1. Lớp AddrSpace

Lớp AddrSpace (*thread/addrspace.cc*) dùng để cấp vùng nhớ và nạp chương trình người dùng, để hỗ trợ đa chương, ta cần có những thay đổi sau trong hàm *AddrSpace::Load(char *filename)*.

- Vì các chương trình không còn được nạp lên các frame liên tục trên bộ nhớ, nên phải thay đổi *physicalPage* lấy địa chỉ frame còn trống từ *Bitmap* toàn cục *gPhysPageBitMap*.
- Nạp chương trình người dùng, bao gồm code, init data và read only data, lên *mainMemory* dựa trên *physicalPage* của *pageTable*.
- Giải phóng bộ nhớ, trả lại frame đã đánh dấu cho *gPhysPageBitMap* khi chương trình người dùng kết thúc.

2. Lớp PCB

Lớp PCB (Process Control Block) được cài đặt trong file *thread/pcb.h* và *thread/pcb.cc*. Dùng để quản lý một tiến trình.

	Tên	Ý nghĩa
Thuộc tính	Semaphore* joinsem	Quản lý quá trình Join của process.
	Semaphore* exitsem	Quản lý quá trình Exit của process.
	Semaphore* mutex	Quản lý truy xuất số lượng tiến trình chờ đang chờ.
	Thread* thread	Quản lý thực thi tiến trình.
	int parentID	Lưu process id của tiến trình cha.
	OpenFile** fileTable	Quản lý các file đang được mở trong tiến trình.
Phương thức	void JoinWait()	Chuyển tiến trình sang trạng thái chờ cho đến khi được giải phóng bởi JoinRelease().
	void JoinRelease()	Giải phóng tiến trình khỏi trạng thái chờ bởi JoinWait().

	void ExitWait()	Chuyển tiến trình sang trạng thái chờ đến khi được gọi ExitRelease().
	void ExitRelease()	Giải phóng tiến trình khỏi trạng thái chờ bởi ExitWait().
	int Exec(char *filename, pid)	Thực thi tiến trình tên là filename với process id là <i>pid</i> .
	OpenFileID Open(char*name, int type)	Mở một file với type 0: đọc và ghi hoặc type 1: chỉ đọc.
	int Close(OpenFileID fid)	Đóng một file có file id là <i>fid</i> .
	int Read(char* buffer,int charcount , OpenFileID id)	Đọc chuỗi buffer có charcount kí tự từ file có file id là <i>id</i> .
	int Write(char* buffer,int charcount, OpenFileID id)	Ghi chuỗi buffer có charcount kí tự từ file có file id là <i>id</i> .
	int Seek(int position, OpenFileID id)	Di chuyển vị trí con trỏ đọc ghi file có id là <i>id</i> một khoảng position kí tự

3. Lớp PTable

Lớp PTable được cài đặt trong file thread/ptable.h và thread/ptable.cc. Dùng để quản lý các tiến trình được chạy của chương trình người dùng.

	Tên	Ý nghĩa
Thuộc tính	Bitmap *bm	Đánh dấu các vị trí đã được sử dụng.
	PCB *pcb[MAX_PROCESS]	Mảng gồm MAX_PROCESS = 10 PCB cấp cho người dùng.
Phương thức	int JoinUpdate(int id)	Xử lý cho syscall SC_Join, kiểm tra pid hợp lệ, gọi pcb[id]->JoinWait() để tiến trình cha đợi đến khi tiến trình con này kết thúc, sau đó gọi

		pcb[id]->ExitRelease() để cho phép tiến trình con này được kết thúc.
	void ExitUpdate(int exitcode)	Xử lý cho syscall SC_Exit, lấy process id của tiến trình hiện hành, gọi pcb[id]->JoinRelease() để tiến trình cha có thể tiếp tục, sau đó pcb[id]->ExitWait() để xin tiến trình cha cho phép dừng.
	void ExecUpdate(char *name)	Xử lý cho syscall SC_Exec. Trường hợp tên không hợp lệ hoặc không còn chỗ trống thì trả về -1. Nếu hợp lệ thì thực thi tiến trình mới bằng cách gọi thread->Fork((VoidFunctionPtr) &StartProcess, (void*)pid).
	OpenFileID Open(int pid, char*name, int type)	Mở một file của tiến trình có id là pid
	int Close(int pid, OpenFileID fid)	Đóng một file có id fid của tiến trình có id là pid.
	int Read(int pid, char* buffer,int charcount, OpenFileID fid)	Đọc chuỗi buffer có charcount kí tự từ file có file id là <i>fid</i> của tiến trình có id là <i>pid</i> .
	int Write(int pid, char* buffer, int charcount, OpenFileID fid)	Ghi chuỗi buffer có charcount kí tự từ file có file id là <i>fid</i> trong tiến trình có id là <i>pid</i> .
	int Seek(int pid, int position, OpenFileID fid)	Di chuyển con trỏ đến vị trí position của file có id là <i>fid</i> trong tiến trình có id là <i>pid</i> .

- Hàm StartProcess(int id) sau khi Fork, lúc scheduler cho phép thực thi, sẽ tạo một AddrSpace để nạp chương trình người dùng và bắt đầu thực thi tiến trình này.

4. Lớp Sem

Lớp Sem được cài đặt trong file thread/stable.h và thread/stable.cc. Dùng để quản lý một Semaphore.

	Tên	Ý nghĩa
Thuộc tính	char* name	Tên của semaphore.
	Semaphore *sem	Semaphore mà Sem quản lý.
Phương thức	wait()	Thực hiện sem->P() để chờ.
	signal()	Thực hiện sem->V() để giải phóng.
	getName()	Lấy tên của Semaphore.

5. Lớp STable

Lớp Stable được cài đặt trong file thread/stable.h và thread/stable.cc chung với lớp Sem. Dùng để quản lý các Semaphore mà người dùng sử dụng, cho phép người dùng tạo tối đa MAX_SEMAPHORE = 10.

	Tên	Ý nghĩa
Thuộc tính	Bitmap *mSemBitMap	Bitmap quản lý mảng cho biết Semaphore nào đã được dùng.
	Sem *semTable[MAX_SEMAPHORE]	Mảng Semaphore được cấp cho người dùng.
Phương thức	int Create(char *name, int value)	Cấp phát một semaphore tên name với giá trị khởi tạo là value. Nếu name đã tồn tại hoặc hết ô trống thì sẽ trả về -1.
	int Wait(char *name)	Semaphore name thực hiện chờ, trả về -1 nếu name không hợp lệ ngược lại trả về 1.

	int Signal(char *name)	Giải phóng semaphore name, trả về -1 nếu name không hợp lệ.
	int FindFreeSlot()	Tìm slot trống trên mảng semTable, trả về -1 nếu không tìm thấy.

C. Cài đặt

1. System call nhập xuất file

1.1. int CreateFile(char *name)

- **Input:**
 - name: chuỗi kí tự tên file cần tạo.
- **Output:**
 - Trả về 0 nếu tạo file thành công, -1 nếu tạo file thất bại.
- **Cài đặt:**
 - Đọc địa chỉ file cần tạo từ thanh ghi R4, chuyển dữ liệu vừa đọc từ Userspace về kernelspace bằng hàm User2system().
 - Gọi hàm SysCreateFile() để xử lí:
 - Kiểm tra nếu tên file không hợp lệ (không có kí tự nào hoặc NULL) thì trả về -1 (lỗi không thể tạo file).
 - Gọi hàm Create(filename) của lớp FileSystem để tạo file. Nếu thành công trả về 0, nếu thất bại trả về -1.
 - Giải phóng vùng nhớ
 - Tăng thanh ghi PC.

1.2. OpenFileID Open(char *name, int type)

- **Input:**
 - name: chuỗi kí tự tên file cần mở.
 - Type: type = 0 cho mở file đọc và ghi, type = 1 cho mở file chỉ đọc.

- **Output:**
 - Trả về id của file nếu mở thành công, trả về -1 nếu mở file thất bại.
- **Cài đặt:**
 - Đọc địa chỉ file cần mở từ thanh ghi R4, đọc type từ thanh ghi R5, chuyển dữ liệu vừa đọc từ Userspace về kernelspace bằng hàm User2system().
 - Gọi hàm SysOpen() để xử lí:
 - Kiểm tra nếu file đang được mở thì trả về -1.
 - Dùng hàm FindFreeSlot() để tìm vị trí trống trong mảng fileTable[] (mảng quản lí file đang được mở).
 - Nếu không tìm được vị trí trống thì trả về -1 (lỗi không thể mở file).
 - Nếu tìm được vị trí trống trong mảng fileTable[] thì gọi hàm Open của lớp FileSystem để mở file.
 - Trả về 0 nếu mở file đọc và ghi, trả về 1 nếu mở file chỉ đọc.
 - Giải phóng vùng nhớ
 - Tăng thanh ghi PC.

1.3. int Close(OpenFileID id)

- **Input:**
 - id: id của file cần đóng.
- **Output:**
 - Trả về 0 nếu đóng file thành công, trả về -1 nếu đóng file thất bại.
- **Cài đặt:**
 - Đọc OpenIdFile cần mở từ thanh ghi R4.
 - Gọi hàm SysClose() để xử lí:
 - Kiểm tra nếu id file không nằm trong đoạn từ 0 đến 9 thì báo lỗi và trả về -1.
 - Nếu id file nằm trong đoạn từ 0 đến 9 thì xóa vùng nhớ lưu trữ file, gán vùng nhớ này bằng NULL, trả về kết quả 0.
 - Giải phóng vùng nhớ, tăng thanh ghi PC.

1.4. `int Read(char *buffer, int charcount, OpenFileID id)`

- **Input:**
 - `buffer`: Chuỗi kí tự lưu kết quả đọc từ file hoặc console.
 - `charcount`: Số lượng kí tự muốn đọc.
 - `id`: Id của file cần đọc, 0 là `stdin`, 1 là `stdout`, từ 2 tới 9 là các file khác.
- **Output:**
 - Số byte thực sự đọc được nếu đọc thành công:
 - -1 nếu bị lỗi .
 - -2 nếu chạm tới cuối file.
- **Cài đặt:**
 - Lấy các tham số địa chỉ `buffer` lưu string cần đọc từ thanh ghi số 4, số byte yêu cầu từ thanh ghi 5 và `id` của file từ thanh ghi số 6.
 - Chuyển dữ liệu của `buffer` từ User space vào Kernel space thông qua hàm `User2System()`.
 - Gọi hàm `int SysRead(char* buffer, int charcount, OpenFileID id)`:
 - Nếu `id < 0`, `id > 9`, `id = 1` (`id` của console write) hoặc file không tồn tại thì `SysRead` trả về -1.
 - Nếu mở được file hợp lệ thì dữ liệu được ghi vào `buffer` và trả về số byte đọc được.
 - Nếu đang ở cuối file thì trả về -2.
 - Nếu kết quả trả về khác -1 và -2 (đọc được dữ liệu). Sau khi đọc xong, chuỗi đang nằm ở Kernel space do đó, cần chuyển dữ liệu chuỗi `buffer` vừa đọc được từ Kernel space về User space thông qua hàm `System2User()`.
 - Giải phóng bộ nhớ của `buffer`
 - Tăng thanh ghi PC

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

- **Input:**
 - `buffer`: Chuỗi kí tự cần ghi vào file hoặc in ra console.

- charcount: Số lượng kí tự muốn in.
- id: Id của file cần ghi, 0 là stdin, 1 là stdout, từ 2 tới 9 là các file khác.
- **Output:**
 - Số byte thực sự ghi được nếu ghi thành công.
 - -1 nếu bị lỗi.
- **Cài đặt:**
 - Lấy các tham số địa chỉ buffer lưu string cần ghi từ thanh ghi số 4, số byte yêu cầu từ thanh ghi 5 và id của file từ thanh ghi số 6.
 - Chuyển dữ liệu của buffer từ User space vào Kernel space thông qua hàm User2System().
 - Gọi hàm `int SysWrite(char* buffer, int charcount, OpenFileID id)`:
 - Nếu $id < 0$, $id > 9$, $id = 0$ (id của console read), type của file là dạng chỉ đọc hoặc file không tồn tại thì SysWrite trả về -1.
 - Nếu mở được file hợp lệ thì dữ liệu được ghi vào buffer và trả về số byte đã ghi.
 - Giải phóng bộ nhớ của buffer
 - Tăng thanh ghi PC

2. Đa chương, lập lịch và đồng bộ hóa

2.1. SpaceID Exec(char* name)

- **Input:**
 - name: tên của file cần chạy.
- **Output:**
 - -1 nếu bị lỗi và thành công thì trả về.
 - SpaceID của chương trình người dùng vừa được tạo nếu thành công.
- **Cài đặt:**
 - Lấy các tham số địa chỉ buffer lưu string cần ghi từ thanh ghi số 4.
 - Chuyển dữ liệu của buffer từ User space vào Kernel space thông qua hàm User2System().

- Gọi hàm void SysExec(cchar *name):
 - Ghi -1 vào thanh ghi số 2 nếu name = NULL hoặc khi quá trình mở một file bị lỗi.
 - Ghi id của tiến trình mới vào thanh ghi số 2 nếu chạy thành công.
- Giải phóng bộ nhớ của name
- Tăng thanh ghi PC

2.2. int Join(SpaceID id)

- **Input:**
 - id: process id của tiến trình muốn join vào tiến trình cha.
- **Output:**
 - Trả về -1 nếu id không hợp lệ hoặc tiến trình đang cố join vào tiến trình không phải tiến trình cha. Ngược lại trả về exit code của tiến trình.
- **Cài đặt:**
 - Nhận tham số từ register.
 - Gọi kernel->pTab->joinUpdate(id) để join vào tiến trình cha, chỉ khi tiến trình con này kết thúc thì tiến trình cha mới được tiếp tục thực thi.
 - Ghi kết quả vào register 2
 - Tăng thanh ghi PC

2.3. void Exit(int exitCode)

- **Input:**
 - exitCode: exit code trả về cho tiến trình mà nó đã chạy.
- **Output:**
 - Không có.
- **Cài đặt:**
 - Lấy tham số exitCode từ register 4.
 - Gọi kernel->pTab->ExitUpdate(exitCode) để giải phóng tiến trình và xin tiến trình cha để kết thúc.
 - Gọi kernel->currentThread->FreeSpace() để thu hồi bộ nhớ.
 - Gọi kernel->currentThread->Finish() để báo scheduler kết thúc tiến trình.
 - Tăng thanh ghi PC

2.4. `int CreateSemaphore(char* name, int semVal)`

- **Input:**
 - name: tên của semaphore cần tạo.
 - semVal: giá trị khởi tạo của semaphore.
- **Output:**
 - Trả về chỉ số của semaphore, nếu semaphore đã tồn tại hoặc đã hết ô trống thì trả về -1.
- **Cài đặt:**
 - Đọc tham số name và semVal từ register 4 và 5.
 - Gọi `kernel->semTab->Create(name, semVal)` đã được cài đặt trong lớp Stable để tạo semaphore mới.
 - Ghi kết quả trả về vào thanh ghi số 2
 - Tăng thanh ghi PC

2.5. `int Wait(char* name)`

- **Input:**
 - name: tên của semaphore cần chờ.
- **Output:**
 - Trả về -1 nếu không thành công, ngược lại trả về 1.
- **Cài đặt:**
 - Đọc tham số từ register 4.
 - Gọi `kernel->semTab->Wait(name)` được cài đặt ở lớp Stable để thực hiện chờ cho semaphore name.
 - Ghi kết quả trả về vào thanh ghi số 2
 - Tăng thanh ghi PC

2.6. `int Signal(char* name)`

- **Input:**
 - name: tên của semaphore cần giải phóng.

- **Output:**
 - Trả về -1 nếu không thành công, ngược lại trả về 1.
- **Cài đặt:**
 - Đọc tham số từ register 4.
 - Gọi kernel->semTab->Signal(name) được cài đặt ở lớp Stable để thực hiện giải phóng cho semaphore name.
 - Ghi kết quả trả về vào thanh ghi số 2.
 - Tăng thanh ghi PC

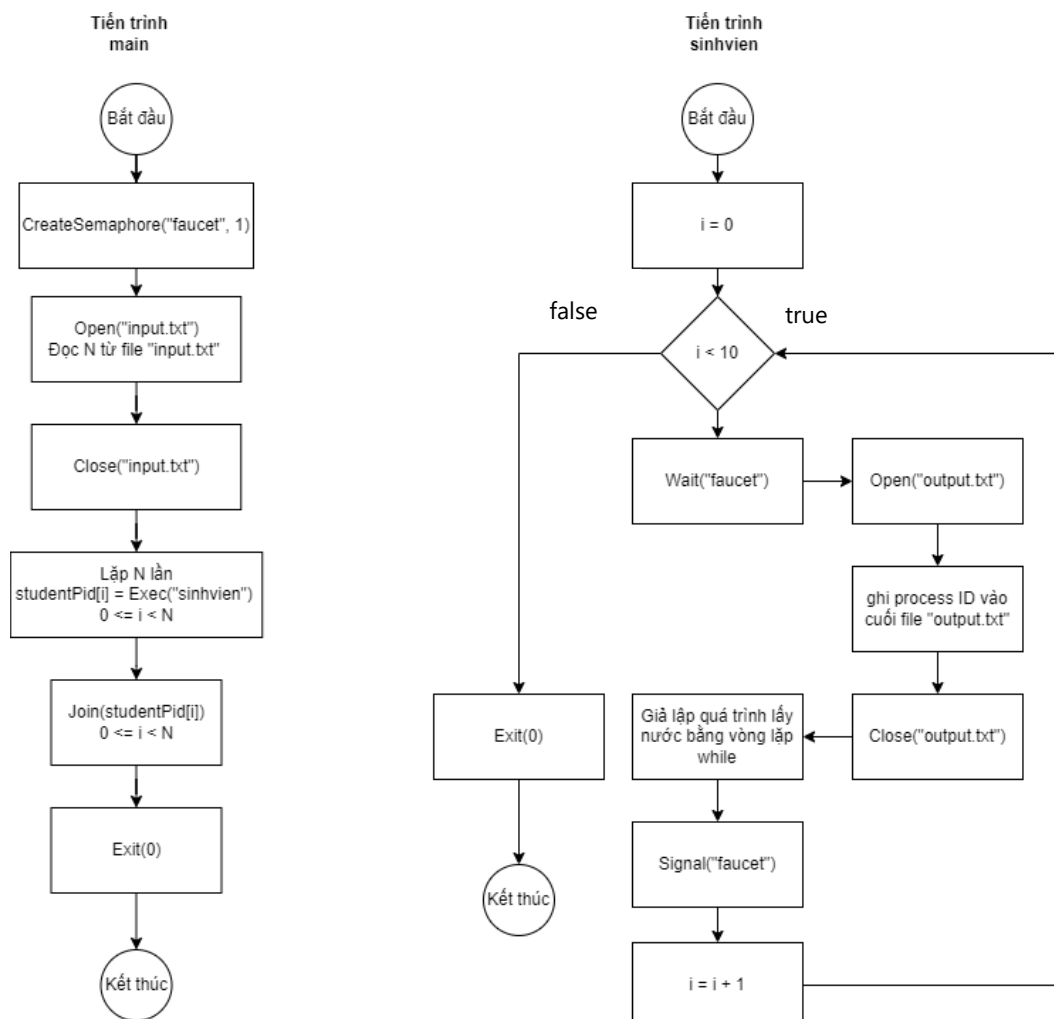
D. Hướng dẫn sử dụng chương trình

Chương trình minh họa cho các system call đã cài đặt bao gồm: CreateFile, Open, Close, Read, Write, Exec, Join, Exit, CreateSemaphore, Wait, Signal.

1. Chương trình vòi nước

Thực hiện đồng bộ giải quyết bài toán có N sinh viên, mỗi sinh viên muốn lấy 10 lít nước, nhưng mỗi lượt chỉ được lấy 1 lít và phải trả lại vòi nước sau khi lấy xong.

Vòi nước là tài nguyên chung, mỗi sinh viên là một tiến trình tham gia tranh chấp tài nguyên. Hình bên dưới mô tả lưu đồ thuật toán cho hai chương trình.



Chương trình chính cài đặt trong file test/main.c, chương trình mô phỏng quá trình tranh chấp và lấy nước của mỗi sinh viên được cài đặt trong file test/student.c, đọc đầu vào từ test/input.txt và xuất kết quả vào test/output.txt.

Để chỉ cho một sinh viên vào lấy nước tại một thời điểm, sử dụng một Semaphore “faucet” với giá trị khởi tạo là 1, Wait(“faucet”) khi có sinh viên vào lấy nước và Signal(“faucet”) khi sinh viên đã lấy nước xong.

Để chạy chương trình, ta vào thư mục *test* và chạy lệnh.

../build.linux/nachos -rs 1023 -x main

```
jug@LAPTOP-DKU7IQF9:~/NachOS-Project/project/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -rs 1023 -x main
Machine halting!

Ticks: total 1083859, idle 0, system 108250, user 975609
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Dữ liệu vào nằm ở file *test/input.txt*.

```
input.txt U test/input.txt
1 3
```

Dữ liệu đầu ra nằm ở file *test/output.txt*.

```
output.txt U test/output.txt
1 1 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1 3 3 2 2
```

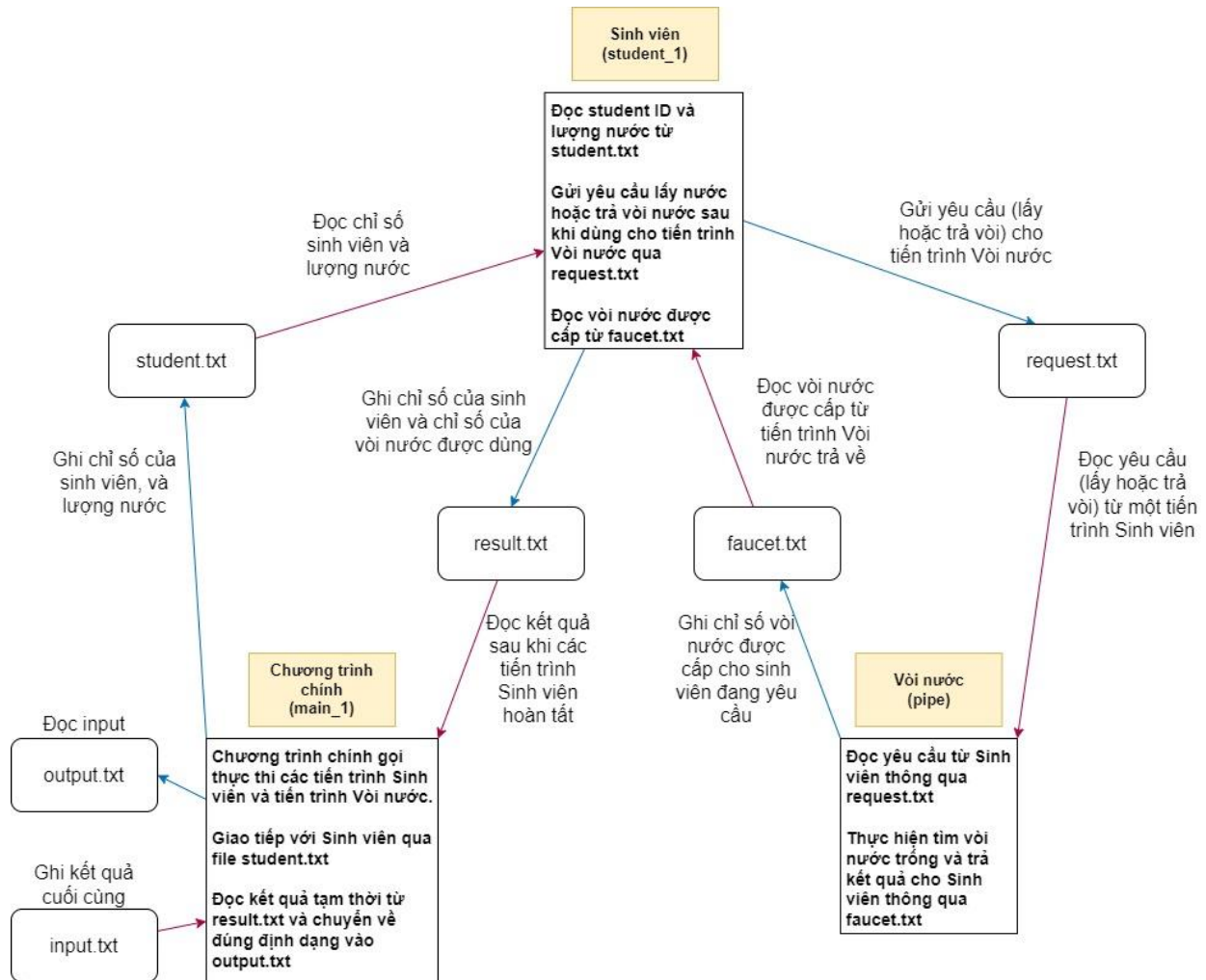
2. Chương trình hai vòi nước

Để hiểu hơn về đa chương, lập lịch và đồng bộ, nhóm cũng đã cài đặt thêm chương trình “Thống kê sử dụng máy lạnh”. Bài toán yêu cầu đồng bộ cho hai vòi nước phục vụ cho các sinh viên, mỗi vòi nước chỉ phục vụ một sinh viên tại một thời điểm.

Chương trình chính cài đặt trong file *test/main_1.c*, chương trình mô phỏng sinh viên trong file *test/student_1.c* và chương trình mô phỏng vòi nước trong file *test/pipe.c*.

Tiến trình chính gọi thực thi các tiến trình Sinh viên và tiến trình Vòi nước. Tiến trình chính tạo các tiến trình Sinh viên và chờ tất cả hoàn thành việc lấy nước. Các tiến trình Sinh viên sẽ tranh chấp tài nguyên chung là 2 vòi nước. Tiến trình Vòi nước sẽ cung cấp vòi nước cho Sinh viên giành được ưu thế.

Mô hình cài đặt được mô tả như hình bên dưới.




Để chạy chương trình, ta vào thư mục *test* và chạy lệnh.

../build.linux/nachos -rs 1023 -x main_1

```
jug@LAPTOP-DKU7IQF9:~/NachOS-Project/project/nachos/NachOS-4.0/code/test$ ../build.linux/nachos -rs 1023 -x main_1
Machine halting!


Ticks: total 1175495, idle 0, system 122950, user 1052545
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Dữ liệu vào nằm ở file *test/input.txt*.

 `input.txt` U `test/input.txt`

```
1 4
2 4 2 1
3 5 3 4 1
4 10 5 4 1
5 1 2 3 4 5
```

Dữ liệu đầu ra nằm ở file `test/output.txt`.

 `output.txt` U `test/output.txt`

```
1 1 2 2
2 1 2 2 1
3 1 2 2 2
4 1 2 1 2 1
```

E. Tài liệu tham khảo

- Các tài liệu được giáo viên thực hành cung cấp:
 - `Huong dan cac syscall va da chuong.pdf`
 - `Constructor_Cua_AddrSpace.pdf`
 - `[5] Da Chuong Dong Bo Hoa.doc`