

**VNUHCM-UNIVERSITY OF SCIENCE**

**FACULTY OF INFORMATION TECHNOLOGY**

**Artificial Intelligence**

**Midterm Examination**

**Lớp:** 23CLC06

**Họ tên:** Đào Trọng Nam

**MSSV:** 20127244

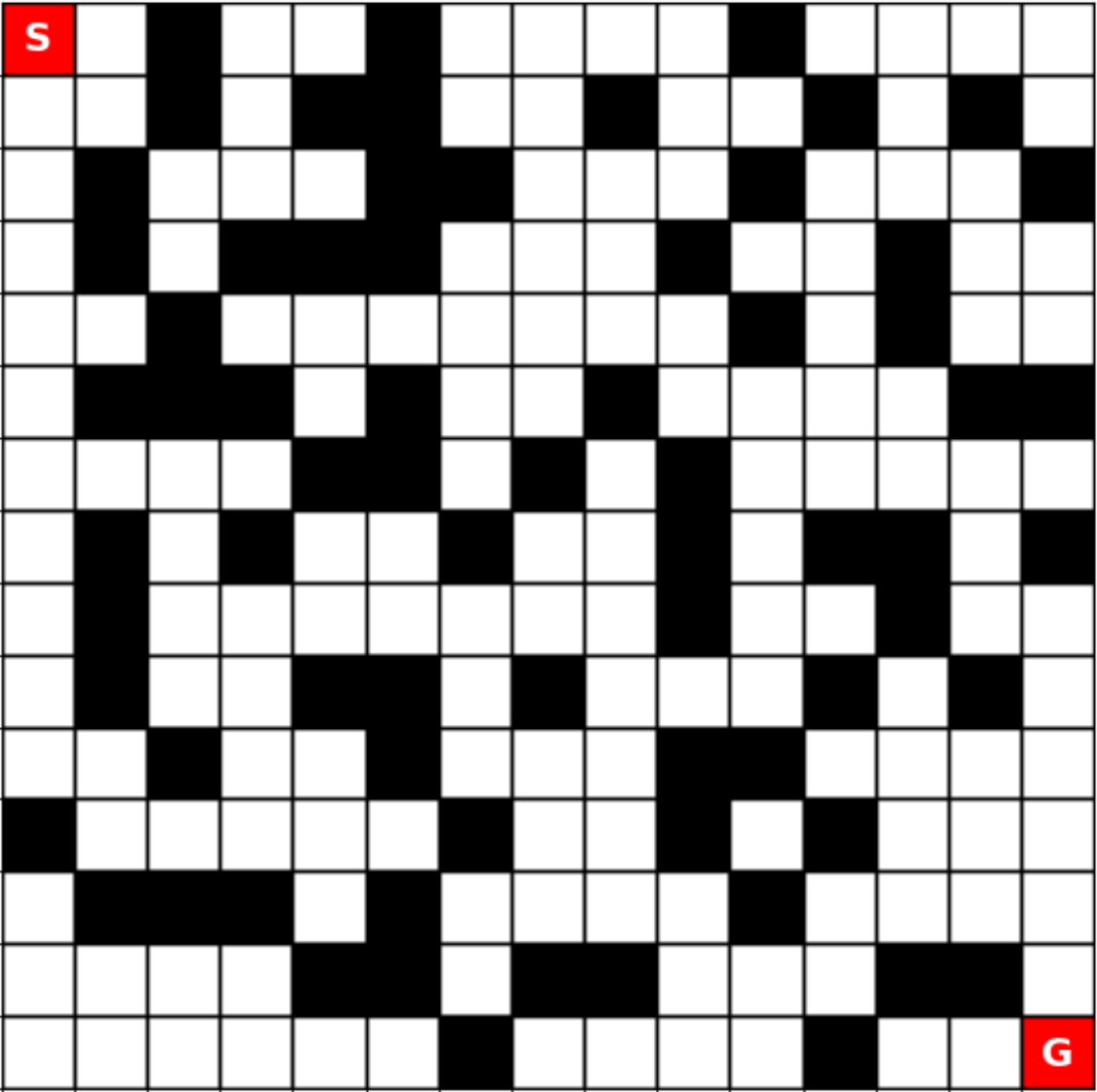
**Môn học:** Artificial Intelligence

**Năm học**: 2024-2025

**Giáo viên giảng dạy:** Bùi Tiến Lên

**TABLE OF CONTENTS**

1. **Instructions**
2. **Q1: Uninformed Search**
   1. A Solve the maze using BFS, visualize the paths found, and highlight the cells that were visited.
   2. Solve the maze using DFS (assuming that the agent moves in the order north, south, east, west), visualize the paths found, and highlight the failed paths.
   3. Compare BFS and DFS in terms of time and space complexity.
3. **Q2: Informed Search**
4. Use the A\* algorithm to find an optimal path.
5. Define a heuristic function (e.g., Manhattan distance).
6. Compare A with BFS and DFS in terms of efficiency.
7. Visualize the results
8. **Q3: Local Search**
9. Use the 15 × 15 grid maze below to solve this question.
10. Implement the Hill-Climbing algorithm to find a low-cost path.
11. Compare the results with A\* in terms of solution quality and efficiency.Visualize the result
12. **INSTRUCTIONS**

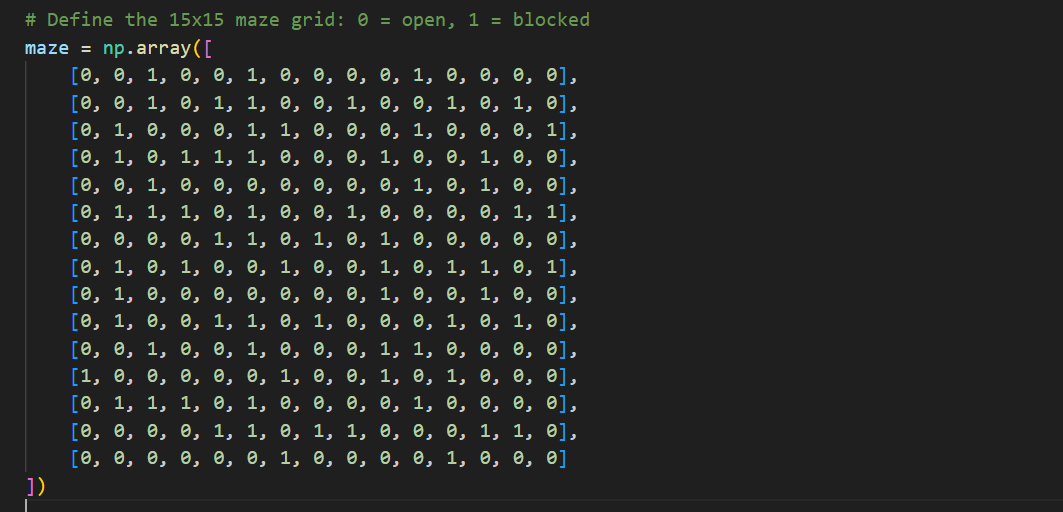
****

* Use the 15 × 15 grid matrix provided for all questions.
* Present the calculation clearly and concisely.
* Use appropriate symbols and diagrams where necessary.
* Assume all moves (North, South, East, West) have a cost of 1, except in Question 3 (Local Search), where each cell has its own reward.
* The agent (S) always starts in the upper left cell.
* The goal (G) is in the lower left cell.
* The agent can move to the adjacent cell.

1. **Q1: Uniformed Search**

**Define the 15x15 maze grid:**

Because all 3 questions use the same maze as the topic, I will declare the maze in this form at the beginning of the question. The cells with the number 0 are open cells, and the cells with the number 1 are blocked cells.



**Definition of initial position state:**

****

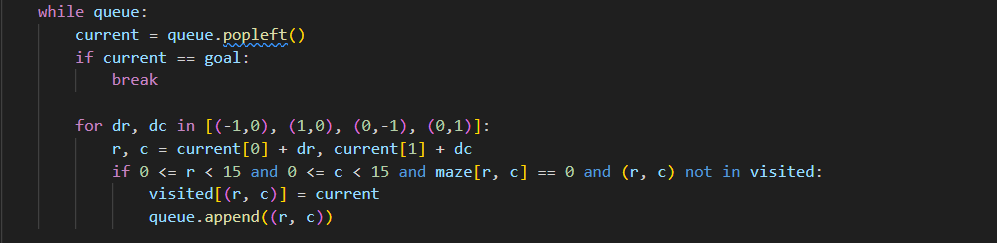
* 1. **A Solve the maze using BFS, visualize the paths found, and highlight the cells that were visited.**

**Initialize BFS:**

****

* **Queue:** manages cells waiting to be visited.
* **Visited:** keeps track of visited cells and their parent cells (previously visited cells).

**Main loop (BFS traversal):**

****

* Each time we examine a cell **(current),** we will expand to the surrounding cells that have not been visited.
* Ensure layered/layered search (BFS - Breadth-First Search).

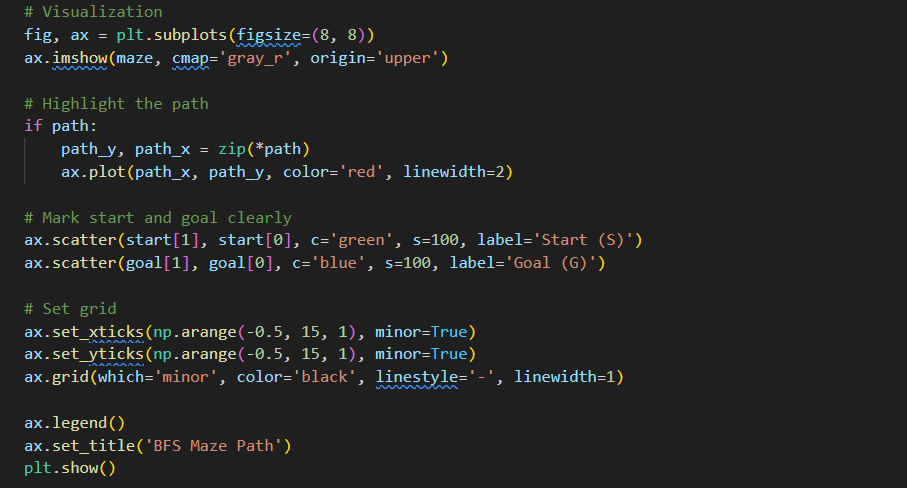
**Trace the path (when the destination is found) and display the result:**

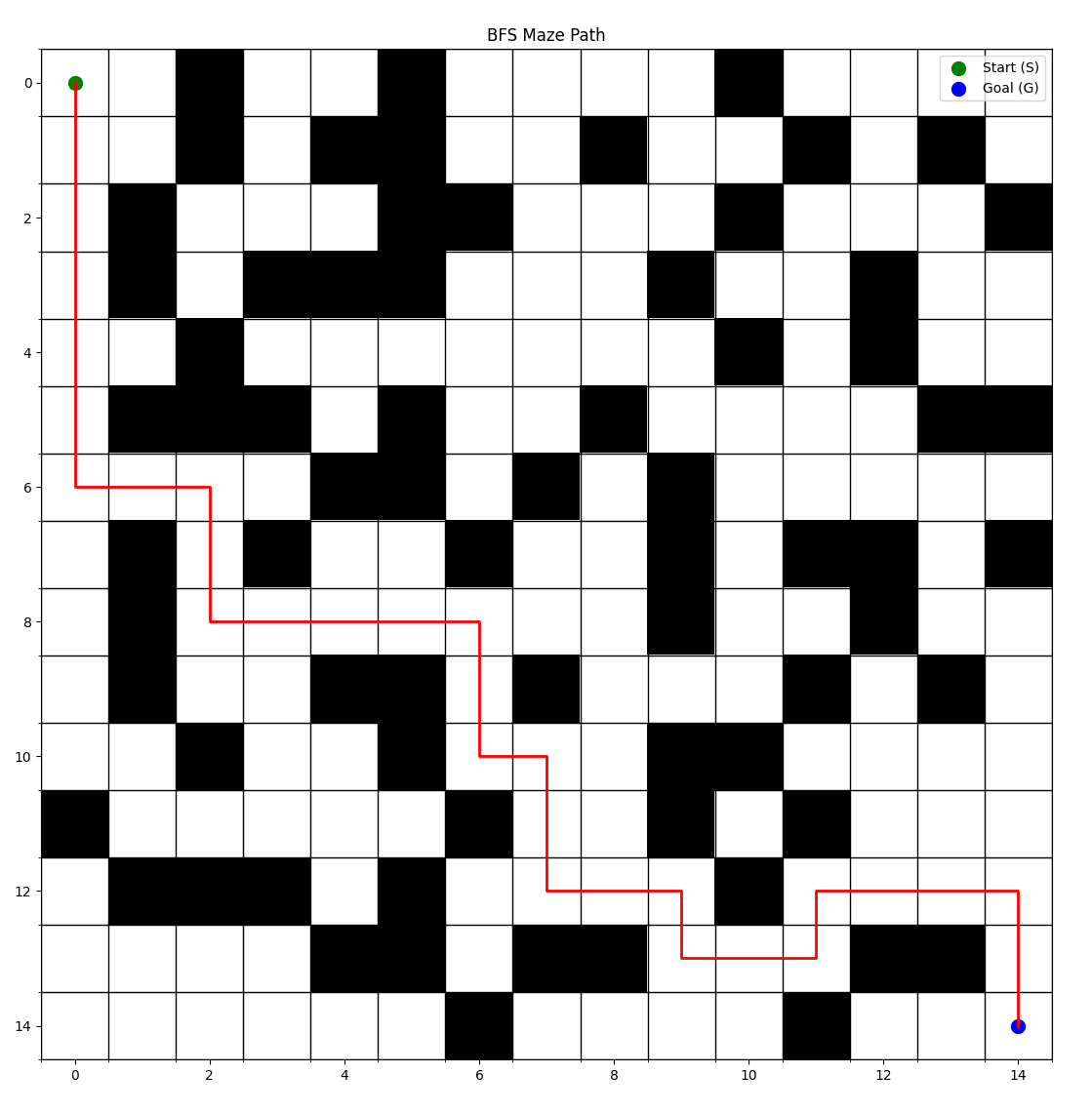


* Trace back from goal to start through visited parent cells.
* Reverse to show the correct path from start to goal.
* Call a function that returns a path from start to goal if available.

**Visualize the maze and path:**

Use ***Matplotlib*** library to visualize maze as matrix (2 dimensional array).

****

****

* 1. THIẾT KẾ GIẢI PHÁP

➢Cơ sở đồng bộ hóa:

Cơ sở của đồng bộ hóa chính là đối tượng Semaphore. (./threads/synch.h)

Semaphore(char \* debugName, int initialValue): Phương thức khởi tạo mặc định có tham số truyền vào là initialValue với ý nghĩa là Semaphore này sẽ có tối đa initialValue tiến trình được phép thực thi cùng lúc.

* void P(): Giảm biến đếm semaphore xuống, block tiến trình nếu như biến đếm này bằng 0.
* void V(): Tăng biến đếm semaphore lên và gọi một tiến trình thực thi nếu tiến trình này đang chờ thực thi từ hàng đợi queue.

➢ Lớp Thread: (./threads/thread.h)

Lớp thread tạo ra các tiểu trình bao gồm việc nạp và cấp phát vùng nhớ Stack,

quản lý trạng thái của tiến trình.

Một vài thuộc tính quan trọng:

* char\* name: Lưu tên của tiến trình, đường dẫn tương đối tới chương trình thực thi.
* AddrSpace \* space: Vùng nhớ của tiến trình trên ram ảo. Một vài hàm quan trọng:
* void Thread(char \* debugName): Khởi tạo một tiến trình với debugName là đường dẫn tương đối tới file chương trình thực thi.
* void Fork(VoidFunctionPtr func, int arg): 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.

➢ Lớp BitMap: (./userprog/bitmap.h)

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 Find(): Tìm một khung trang trống và đánh dấu nó đã được sử dụng.
* int NumClear(): Trả về tổng số khung trang còn trống trên bộ nhớ.

➢ Lớp PCB: (./userprog/userprog.h)

* PCB (Process Control Block): Lưu thông tin để quản lý process.
* Một số thuộc tính quan trọng:
* int pid: Định danh của tiến trình để phân biệt các tiến trình.
* Thread\* thread; Lưu tiến trình được nạp.
* int parentID: id của tiến trìn cha.
* FileName: Lưu tên của tiến trình.
* 3 thuộc tính Semaphore: Để quản lý quá trình Join, Exit và nạp chương trình.

➢ Lớp PTable:

Dùng để quản lý các tiến trình được chạy trong hệ thống.

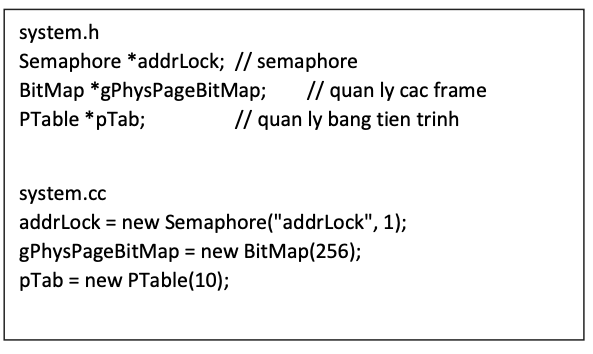
PCB\* pcb[MAX\_PROCESSES] là một bảng mô tả tiến trình có cấu trúc mảng một chiều có số phần tử tối đa MAX\_PROCESSES = 10 theo yêu cầu của đồ án.

Mỗi phần tử là một con trỏ lưu trữ đối tượng của lớp PCB.

Hàm constructor của lớp sẽ khởi tạo tiến trình cha (là tiến trình đầu tiên) ở vị trí 0 tương đương với phần tử đầu tiên của mảng. Từ tiến trình này, chúng ta sẽ tạo ra các tiến trình con thông qua system call Exec().

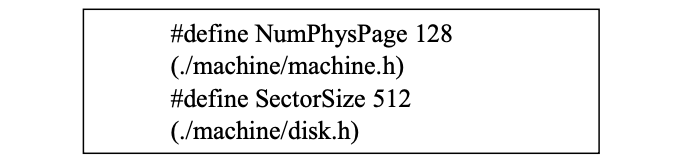
* 1. CÀI ĐẶT CHI TIẾT

➢ Bước 1: Khai báo các biến toàn cục trong ./threads/system.h và tạo đối tượng trong system.cc.



➢ Bước 2: Cài đặt 2 lớp PCB và PTable và tiến hành khai báo trong file để quản lý tiến trình “Makefile.common” 2 lớp vừa thêm.

➢ Bước 3: Resize lại số khung trang và kích thước của Sector



➢ Bước 4: Chỉnh sửa lại class Thread trong ./threads/thread.h

* Thêm int processID để quản lý ID phân biệt giữa các tiến trình.
* Thêm int exitStatus để kiểm tra exit code của tiến trình.
* Cài đặt hàm void FreeSpace() để giải phóng vùng nhớ trên bộ nhớ mà tiến

trình đang dùng.

➢ Bước 5: Cài đặt hàm StartProcess\_2(int id) (./userprog/progtest.cc); Là hàm dùng để hàm Fork trỏ hàm này đến vùng nhớ của tiến trình con.

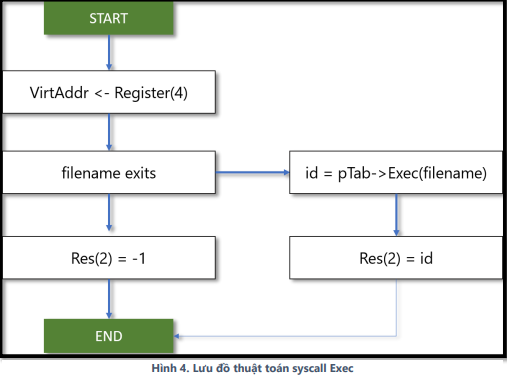
➢ Bước 6: Cài đặt lớp AddressSpace (./userprog/addrspace.cc và addrespace.h).

* Chương trình hiện tại giới hạn chỉ thực thi 1 chương trình -> cần phải có vài thay đổi trong file addrspace.h và addrspace.cc để chuyển từ hệ thống đơn chương thành đa chương. Cụ thể:
* 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 frames.
* 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 một pageTable = new
* TranslationEntry[numPages], tìm trang còn trống bằng phương thức Find() của lớp Bitmap, sau đó nạp chương trình lên bộ nhớ chính.
* pageTable[i].physicalPage = gPhysPageBitMap->Find();

➢ Bước 7: Cài đặt system call

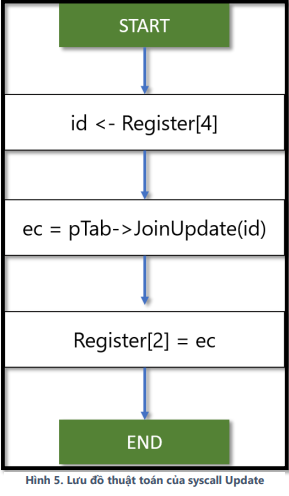
➢ System call Exec:

* Khai báo trong ./userprog/syscall.h SpaceId Exec(char \*name);
* Cài đặt hàm Exec(char \*name, int pid) ở lớp PCB
* Cài đặt hàm ExecUpdate(char\* name) ở lớp PTable.
* Lưu đồ thuật toán:



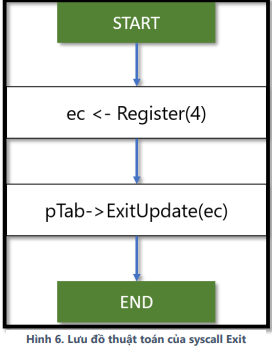
➢ System call Join:

* Khai báo ở ./userprog/syscall.h: int Join(SpaceID id).
* Cài đặt: JointWait(); ExitRelease() ở lớp PCB
* Cài đặt: JoinUpdate(int id) ở lớp PTable
* Lưu đồ:



➢ System call Exit:

* Khai báo ở ./userprog/syscall.h: void Exit(int exitCode).
* Cài đặt hàm: JoinRelease(), ExitWait() ở lớp PCB.
* Cài đặt: ExitUpdate(int exitcode) ở lớp Ptable
* Lưu đồ



* 1. LỚP SEM (./USERPROG/STABLE.H)

Dùng để quản lý Semaphore. Các thuộc tính và phương thức

|  |  |  |
| --- | --- | --- |
|  | Tên | Ý nghĩa |
| Thuộc tính | char name[50] | Lưu tên semaphore. |
| Semaphore\* sem | Dùng đối tượng semaphore đã cài đặt trong hệ thống để quản lý. |
| Phương thức | Sem() | Constructor của lớp |
| ~Sem() | Destructor của lớp |
| wait() | Thực hiện thao tác chờ |
| signal() | Thực hiện thao tác giải phóng Semaphore |
| GetName | Trả về tên của semaphore |

* 1. LỚP STABLE (./USERPRO/STABLE.H)

|  |  |  |
| --- | --- | --- |
|  | Tên | Ý nghĩa |
| Thuộc tính | BitMap \*bm | Quản lý slot trống của bảng. |
| Sem\* semTab [MAX\_SEMAPH ORE] | Quản lý các đối tượng lớp Sem. Số lượng tối đa là MAX\_SEMAPHORE. |
| Phương thức | STable() | Constructor của lớp STable |
| ~Stable | Destructor của lớp STable |
| int Create(char \*name, int init) | Hàm tạo một semaphore mới có tên name và giá trị khởi tạo init và chèn vào bảng mô tả semaphore nếu thành công |
| int Wait(char \*name) | Nếu tồn tại semaphore trong bảng thì thực hiện gọi semTab[i]->wait(); |
| int Signal(char \* name) | Nếu tồn tại semaphore trong bảng thì thực hiện gọi semTab[i]->signal(); |
| int FindFreeSlot(int id) | Hàm tìm 1 slot trống trên bảng semTab chưa được dùng bởi semaphore nào cả |

* 1. VIẾT CÁC SYSTEM CALL

Các system call về OpenFile đã báo cáo và cài đặt ở Project 1; các system call về đa

chương đã báo cáo và cài đặt ở Project 2.

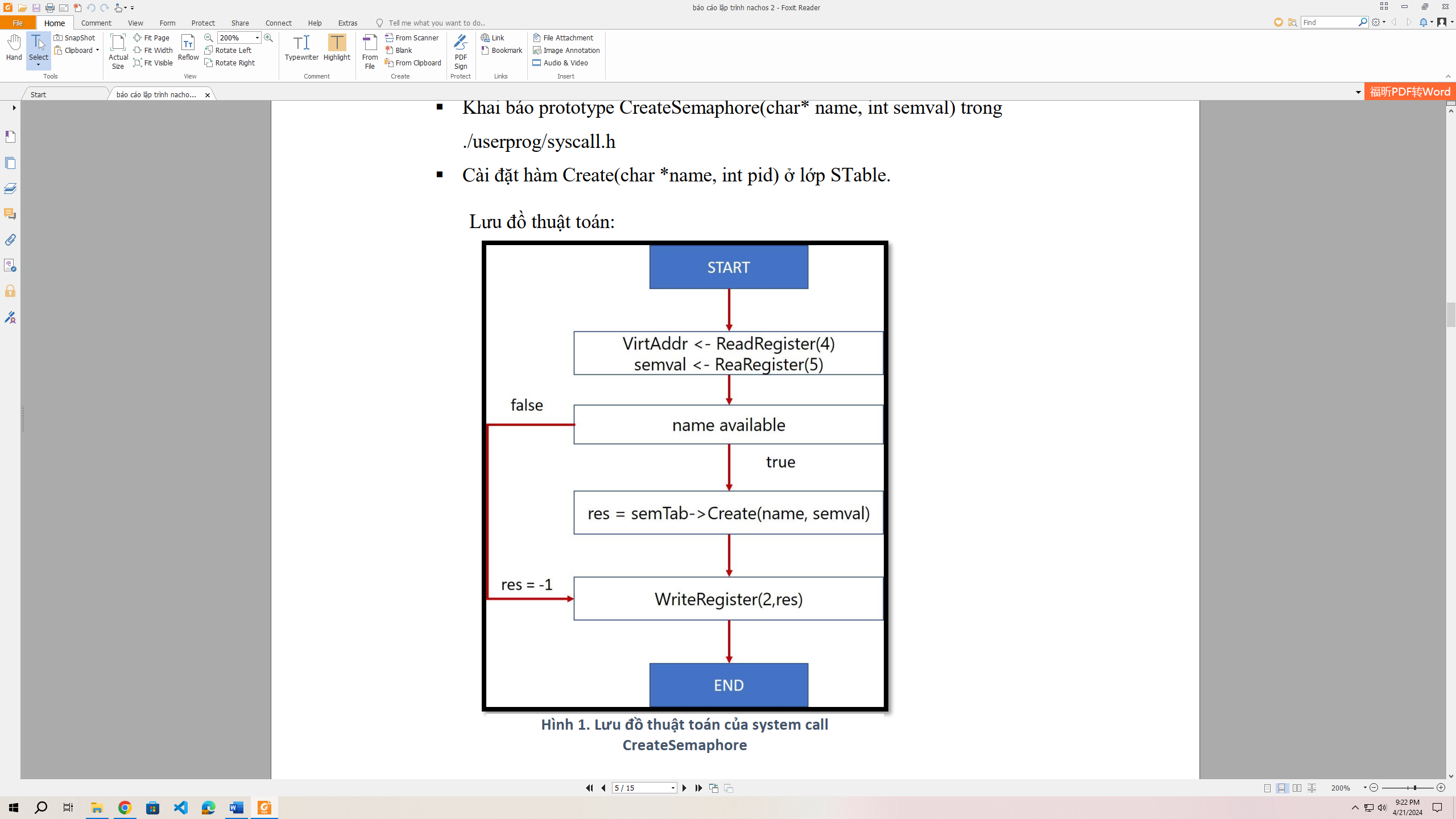
Khai báo biến toàn cục STable\* semTab trong ./thread/system.h và khởi tạo bên

./thread/system.cc.

***Stable\* semTab;***

* + Syscal CreateSemaphore
* Khai báo prototype CreateSemaphore(char\* name, int semval) trong ./userprog/syscall.h
* Cài đặt hàm Create(char \*name, int pid) ở lớp STable.

Lưu đồ thuật toán



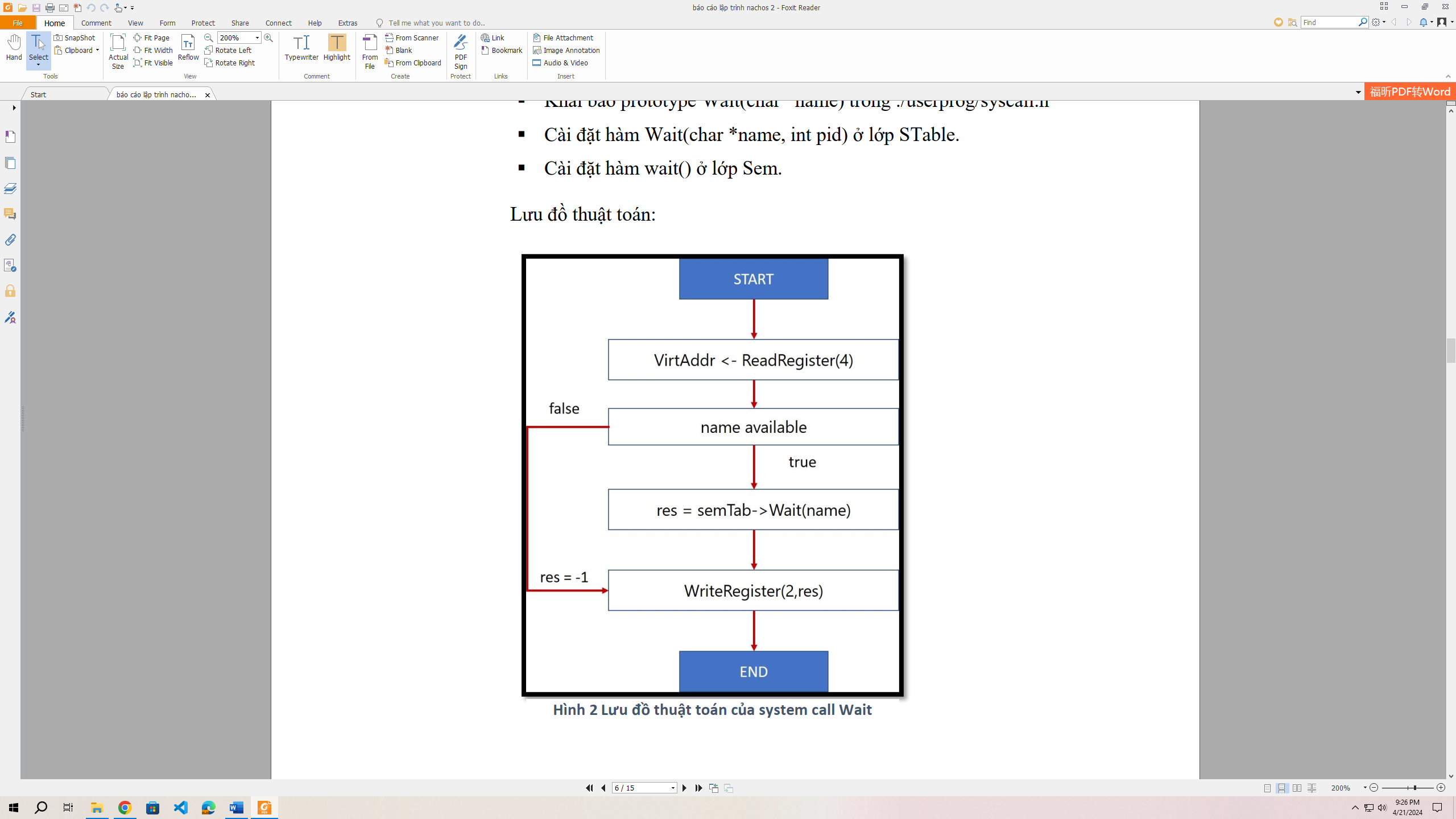
* + Syscall Wait

▪ Khai báo prototype Wait(char\* name) trong ./userprog/syscall.h

▪ Cài đặt hàm Wait(char \*name, int pid) ở lớp STable.

▪ Cài đặt hàm wait() ở lớp Sem.

Lưu đồ thuật toán



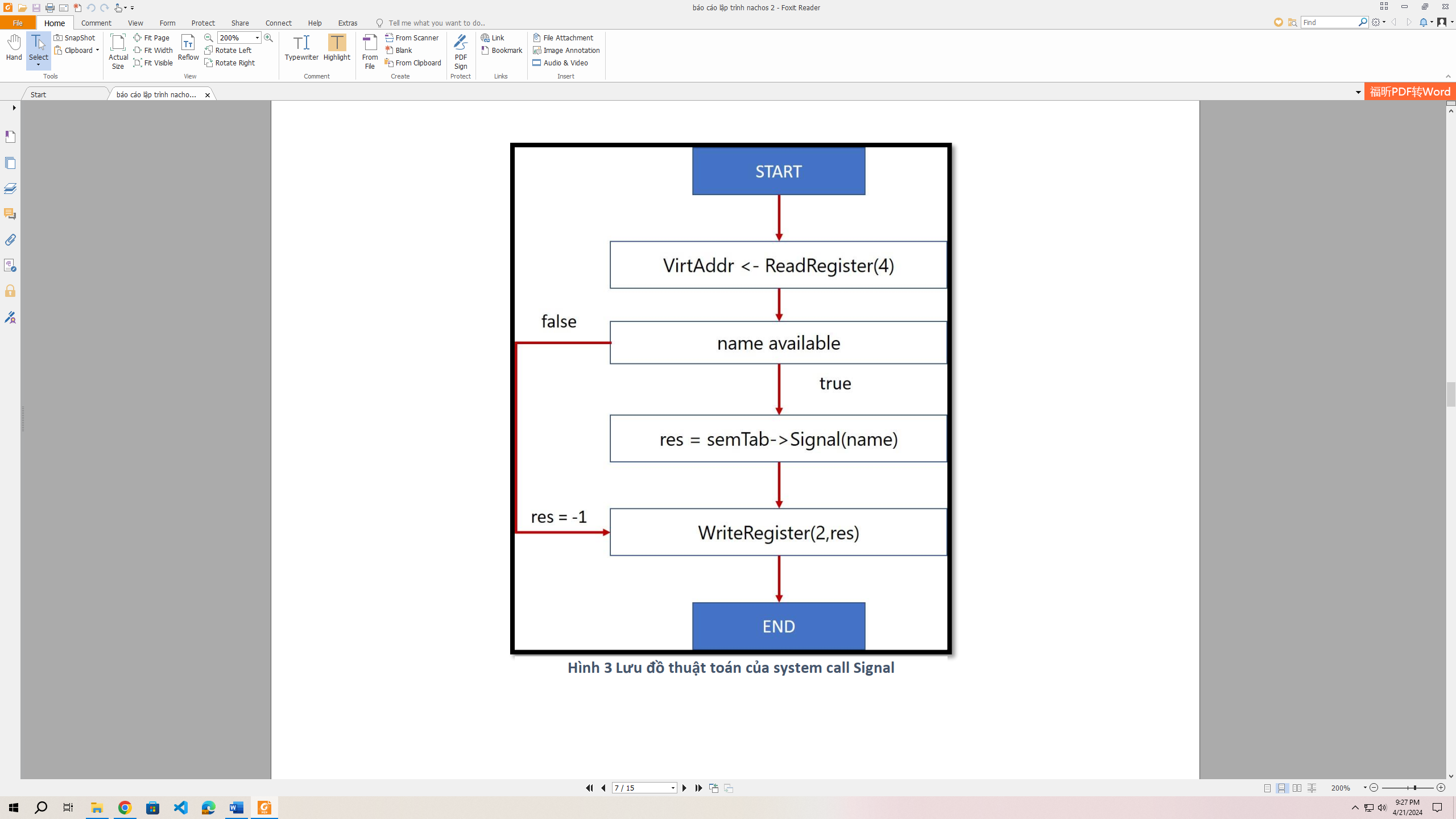
* + Sysvall Signal

▪ Khai báo prototype Signal(char\* name) trong ./userprog/syscall.h

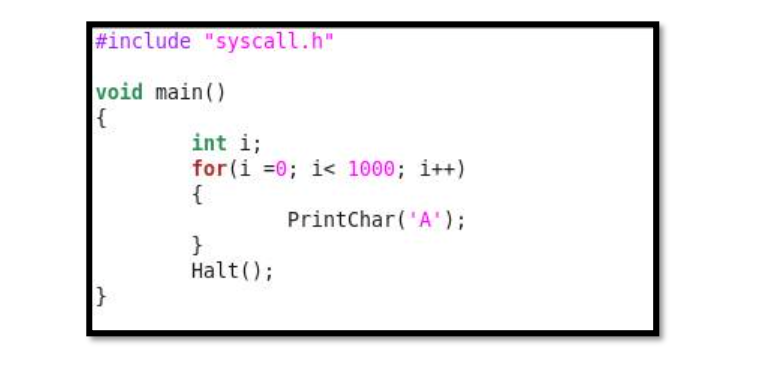
▪ Cài đặt hàm Signal(char \*name, int pid) ở lớp STable.

▪ Cài đặt hàm signal() ở lớp Sem.

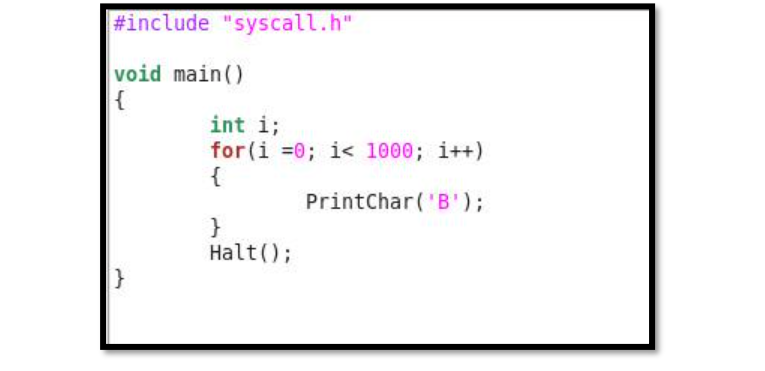
Lưu đồ thuật toán



1. **CHƯƠNG TRÌNH NGƯỜI DÙNG**
2. **Chương trình Ping**

****

1. **Chương trình Pong**

****

1. **Chương trình Scheduler**

****

1. **Kết quả: ./userprog/nachos -rs 1023 -x ./test/scheduler**

****

1. **TÀI LIỆU THAM KHẢO**

**nachos\_canban.pdf**

**nachos\_study\_book.pdf**

**DoAn3\_NachOS.pdf**

**HuongDan\_Project3.pdf**

**Huong Dan Cac Syscall Ve Da Chuong.pdf**