UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

REPORT

OPERATING SYSTEM - NACHOS

BẠCH GIA HUY - 21127615
NGUYỄN HỒNG THÁI - 21127689
HUỲNH SỈ KHA - 21127734

Lecturers:

Phạm Tuấn Sơn

Lê Viết Long

17th April 2023

Contents

Mức độ hoàn thành: Hoàn thành 100% các yêu cầu của đồ án.

Bảng phân công việc			
Tên	**MSSV**	**Công việc**	**Mức độ hoàn thành**
Bạch Gia Huy	21127615	Thiết kế, lên ý tưởng thuật toán cho syscall, viết báo cáo	100%
Nguyễn Hồng Thái	21127689	Thiết kế, lên ý tưởng thuật toán, cài đặt syscall, viết báo cáo	100%
Huỳnh Sỉ Kha	21127734	Cài đặt Syscall, viết báo cáo	100%

1. Class mở rộng

1.1 Class quản lý cấp phát bộ nhớ - BitMap

Khởi tạo đối tượng gPhysPage thuộc lớp BitMap để quản lý việc cấp phát bộ nhớ cũng như tiến trình. Class gồm 1 số phương thức chính sau:

1. *void Mark (int which):* Đánh dấu vị trí tại which (bit thứ n) là từ trạng thái trống sang trạng thái đang sử dụng.
2. *void Clear (int which):* Giải phóng, đánh dấu vị trí tại which từ trạng thái đang sử dụng sang trống.
3. *bool Test (int which):* Kiểm tra vị trí tại which là trống hay đã được dùng.
4. *int Find():* Tìm kiếm vị trí trống. Sau khi tìm được vị trí trống, đánh dấu và trả về vị trí đó, hoặc trả về -1 nếu không có.

1.2 Class quản lý không gian địa chỉ - Addrspace

Class được dùng để khởi tạo không gian địa chỉ cho 1 tiến trình mới.

1. *Hàm khởi tạo Addrspace():*
   - Khởi tạo không gian địa chỉ cho một tiến trình mới. Thông qua việc đọc và xử lý header của file thực thi được truyền vào, cấp phát các trang bộ nhớ cho tiến trình mới và sao chép các phân đoạn mã và dữ liệu từ file thực thi vào các trang bộ nhớ.
   - Nếu số trang bộ nhớ cần thiết vượt quá số trang bộ nhớ vật lý sẵn có, báo lỗi và không cấp phát không gian địa chỉ cho tiến trình mới. Nếu không, tiếp tục bằng việc khởi tạo bảng bản đồ trang (page table) cho tiến trình mới, trong đó mỗi trang bộ nhớ vật lý được ánh xạ tới một trang bộ nhớ ảo. Các trang bộ nhớ vật lý này được cấp phát bằng cách gọi hàm Find() của lớp BitMap.
   - Sau đó, sao chép các phân đoạn mã và dữ liệu từ file thực thi vào các trang bộ nhớ ảo tương ứng. Nếu phân đoạn mã không trống, sao chép các trang phân đoạn mã từ file thực thi vào các trang bộ nhớ ảo được cấp phát cho phân đoạn mã. Tương tự, nếu phân đoạn dữ liệu khởi tạo không trống, constructor sẽ sao chép các trang phân đoạn dữ liệu khởi tạo từ file thực thi vào các trang bộ nhớ ảo được cấp phát cho phân đoạn dữ liệu khởi tạo.

1.3 Class quản lý tiến trình - Thread:

Khởi tạo đối tượng thuộc lớp Thread để quản lý các phương thức thực thi 1 tiến trình. Class gồm 1 số thành phần chính sau:

1. *int m_processId:* lưu id của tiến trình đó. Trong đồ án này là thứ tự của tiến trình trong mảng quản lý các tiến trình.
2. *void Fork (VoidFunctionPtr func, int arg):* Được sử dụng để tạo 1 tiến trình mới và bắt đầu thực thi với 1 hàm được chỉ định (func) với tham số đầu vào là arg.
3. *void Finish()*: Được gọi khi 1 tiến trình hoàn thành phương thức Fork.

1.4 Class quản lý đa chương - ManageProc:

Khởi tạo đối tượng thuộc lớp ManageProc để quản lý việc các tiến trình của chương trình. Class gồm có các phương thức sau:

1. Kế thừa từ class BitMap, 1 mảng 10 phần tử để quản lý việc đóng và chạy tiến trình. Khi thực hiện tiến trình, gọi hàm *Mark* của class BitMap để đánh dấu là tiến trình đang được mở.
2. Mảng kiểu Thread* 10 phần tử dùng để chứa các con trỏ trỏ tới đối tượng thread đang mở.

Các hàm xử lý của class ManageProc:

1. *Hàm khởi tạo - ManageProc():*
   - Kế thừa class BitMap, khởi tạo 1 lớp BitMap 10 phần tử dùng cho việc quản lý đóng và mở các tiến trình.
   - Mảng con trỏ 10 phần tử dùng để trỏ tới đối tượng là các tiến trình đang xử lý.
2. *Hàm huỷ ~ ManagerProc():* Giải phóng vùng nhớ BitMap và mảng con trỏ.
3. *Hàm thực thi - int doExec (char* procName):*
   - Kiểm tra tính hợp lệ của tên tiến trình. Trả về -1 nếu không tên là NULL.
   - Ngăn không cho tiến trình cần thực thi trùng với tiến trình chính. Nếu trùng thì trả về -1.
   - Kiểm tra không gian trống cho tiến trình cần thực thi. Trả về -1 nếu hết không gian.

- Khởi tạo 1 luồng mới với tên là procName được truyền vào. Kiểm tra nếu hết bộ nhớ thì trả về -1. Gán luồng mới vào mảng quản lý các tiến trình tại vị trí đã tìm được. Gán ID là vị trí của tiến trình trong mảng.
- Sau khi hàm Fork được gọi, 1 tiến trình mới sẽ bắt đầu thực thi với hàm StartProcess_2, đồng thời tiến trình hiện tại sẽ tiếp tục thực thi mã lệnh ngay sau khi hàm Fork được gọi. Nghĩa là tiến trình có thể thực hiện 1 công việc khác với tiến trình gọi nó, trong khi vẫn chạy trong cùng 1 chương trình.

4. *Hàm thoát - int checkExit(int exitStatus):* Kiểm tra trạng thái thoát của một tiến trình và thực hiện một số hành động để thoát. Trả về Id của tiến trình nếu thành công và -1 nếu lỗi hoặc exitStatus khác 0.
   - Nếu exitStatus khác 0, trả về -1.
   - Nếu exitStatus bằng 0 thì lấy process Id của tiến trình hiện tại để tiến hành kiểm tra.
   - Nếu Id của tiến trình hiện tại bằng 0 (nghĩa là tiến trình chính) thì tiến hành xoá không gian vùng nhớ Addrspace (nếu có) và gọi phương thức Halt của đối tượng interupt để kết thúc chương trình.
   - Nếu Id khác 0: Tiến hành xoá không gian vùng nhớ *Addrspace*, sau đó gọi phương thức Finish của lớp Thread để hoàn thành tiến trình. Xoá bộ nhớ cấp phát cho tiến trình hiện tại cũng như huỷ đánh dấu trong lớp BitMap. Trả về Id của tiến trình.

2. Các Syscall xử lý đa chương trong chương trình:

2.1 SpaceId Exec(char* name)

Trả về Id của tiến trình hoặc -1 nếu lỗi.

1. Đọc từ thanh ghi tham số do người dùng truyền vào. Đọc từ thanh ghi số 4 địa chỉ của chuỗi c-string. Chuyển đổi từ không gian người dùng sang không gian hệ thống để lấy tham số là tên tiến trình cần thực thi.
2. Hàm bắt đầu bằng việc kiểm tra xem tham số đầu vào proccess name có tồn tại không. Nếu không, nó sẽ in ra màn hình và thoát khỏi hàm. Nếu có, kiểm tra xem độ dài của filename có lớn hơn 0 không. Nếu không, hàm sẽ in ra lỗi và trả về -1.
3. Nếu process name hợp lệ, mở file bằng phương thức Open của đối tượng

fileSystem. Nếu việc mở file không thành công, nghĩa là file cần thực thi không tồn tại, in ra lỗi và trả về -1.

4. Sau đó, hàm gọi phương thức **doExec** [1.4.2] của đối tượng manProc với tham số filename để thực thi file. Nếu quá trình thực thi file không thành công (trả về giá trị âm), in ra thông báo lỗi, trả về -1.
5. Cuối cùng, hàm trả về giá trị processId nếu thành công, hoặc giá trị -1 nếu không thành công.

2.2 int Exit(int exitStatus)

1. Gọi phương thức **checkExit** [1.4.3] của lớp ManageProc với tham số là exitStatus để thực hiện quá trình thoát tiến trình. Trả về exitcode là Id của tiến trình cần thoát.

3. Các vấn đề của xử lý đa chương

3.1 Cấp phát frame bộ nhớ vật lý

Khởi tạo đối tượng của class BitMap có 256 phần tử tượng trưng cho 1 trang của bộ nhớ vật lý.

Ban đầu: Cấp phát không gian địa chỉ với giả thiết là các tiến trình là liên tiếp nhau. Vấn đề phát sinh khi chương trình chạy đa chương, các tiến trình không còn liên tiếp nhau hoặc khi 1 tiến trình hoàn thành và thoát ra sẽ tạo ra các vị trí trống trong vùng nhớ. Nhưng cấp phát kiểu cũ không phát hiện được các vị trí trống gây ra lãng phí bộ nhớ.

Giải pháp: Sử dụng class BitMap để tìm kiếm không gian trống. Nếu tiến trình đã hoàn thành thì bỏ đánh dấu trong BitMap và xem vùng nhớ đó là trống, sẵn sàng cấp phát cho tiến trình kế tiếp.

4. Tài liệu tham khảo:

- Tài liệu hướng dẫn đồ án lập trình NachOS - Môn hệ điều hành - Phạm Tuấn Sơn, Lê Viết Long
- Tài liệu tham khảo