**Bitmap**

* BitMap là một lớp được thiết kế để quản lý một mảng các bit, trong đó mỗi bit có thể được đặt thành ON (1) hoặc OFF (0). Lớp này thường được sử dụng để quản lý tài nguyên trong hệ thống, như các khối đĩa hoặc trang bộ nhớ, bằng cách đánh dấu chúng là đã được sử dụng hoặc chưa được sử dụng.
* Các thuộc tính chính:
  + “numbits”: số lượng bits trong bitmap
  + “numWords”: số lượng words cần thiết để lưu trữ bitmap, dựa trên “numBits”
  + “map”: con trỏ đến mảng các words, mỗi word lưu trữ 32 bit trên hệ thống.
* Các phương thức chính:
  + “BitMap(int nitems)”: khởi tạo bitmap với “nitems” bits, với tất cả các bits được thiết lập về 0 (clear).
  + Destructor “~BitMap()”: giải phóng bộ nhớ được cấp phát cho bitmap.
  + “void Mark(int which)”: đặt bit ở vị trí “which” thành 1.
  + “void Clear(int which)”: xóa bit ở vị trí “which”, thiết lập nó về 0.
  + “bool Test(int which)”: kiểm tra xem bit ở vị trí “which” có đang được set 1 hay không. Trả về “true” nếu bit đó được set.
  + “int Find()”: Tìm bit đầu tiên không được set 0 trong bitmap, đặt nó thành 1 và trả về chỉ số của bit đó. Nếu tất cả các bit đều đã được set, phương thức trả về -1.
  + “int NumClear()”: đếm và trả về số lượng bits chưa được set 0 trong bitmap.
  + “void Print()”: in chỉ số của tất cả các bits được set 1 trong bitmap, dùng cho mục đích debug.
  + “void FetchFrom(OpenFile\* file)”: đọc bitmap từ file.
  + “void WriteBack(OpenFile\* file)”: ghi bitmap vào file.

**openfile.h**

* Định nghĩa các cấu trúc dữ liệu và giao diện cho việc mở, đóng, đọc, và viết vào các file riêng lẻ. Các thao tác được hỗ trợ tương tự như các thao tác trong UNIX. File này cung cấp hai triển khai: một là “STUB” dùng cho mô phỏng, chuyển các thao tác file trực tiếp thành các thao tác UNIX tương ứng, và một là triển khai “thực” chuyển các thao tác này thành các yêu cầu đọc/ghi các sector trên đĩa. Trong triển khai cơ bản của hệ thống tệp, không xử lý truy cập đồng thời vào hệ thống file từ nhiều luồng khác nhau.
* Lớp “OpenFile” (STUB): phần này được sử dụng khi “FILESYS\_STUB” được định nghĩa, nghĩa là triển khai này chủ yếu là một “stub” tạm thời chuyển các thao tác hệ thống tập Nachos sang các thao tác tệp UNIX.
  + Constructor: khởi tạo một file với đối số là file descriptor “f”. “currentOffset được khởi tạo là 0.
  + Destructor: đóng file.
  + “ReadAt”: đọc “numBytes” bytes từ file vào buffer “into” bắt đầu từ vị trí “position” không thay đổi “currenOffset”.
  + “WriteAt”: ghi “numBytes” bytes từ buffer “from” vào file bắt đầu từ vị trí “position” không thay đổi “currentOffset”.
  + “Read”: đọc “numBytes” bytes từ file vào buffer “into” bắt đầu từ “currentOffset” và cập nhật “currentOffset”.
  + “Write”: ghi “numbytes” bytes từ buffer “from” vào file bắt đầu từ “currentOffset” và cập nhật “currentOffset”.
  + “Length”: trả về độ dài của file.
* Lớp “OpenFile” (FILESYS): phần này triển khai “thực” của “OpenFile”, dùng khi “FILESYS” được định nghĩa, tức là triển khai này chuyển các thao tác vào file thanhf các yêu cầu đọc/ghi sector trên đĩa.
  + Constructor: mở một file có header nằm ở sector chỉ định trên đĩa.
  + Destructor: đóng file
  + “Seek”: đặt vị trí từ đó và bắt đầu đọc/ghi – tương tự như lseek trong UNIX.
  + **“**Read”/”Write”: Đọc/ghi “numBytes” bytes từ/đến file, bắt đầu từ vị trí hiện tại. Trả về số byte thực sự được đọc/ghi và cập nhật vị trí trong file.
  + **“**ReadAt”/”WriteAt”: đọc/ghi “numBytes” từ/đến file mà không qua vị trí hiện tại. Điều này cho phép truy cập trực tiếp tới một vị trí cụ thể trong file mà không thay đổi vị trí đọc/ghi hiện tại.
  + **“**Length”: trả về số lượng bytes trong file.

**translate**

* Cung cấp các cấu trúc dữ liệu và hàm để quản lí việc dịch từ số trang ảo sang số trang vật lý, được sử dụng để quản lí bộ nhớ vật lí cho các chương trình người dùng
* Lớp “TranslationEntry”: là một entry trong bảng dịch, được sử dụng cho cả bảng trang và bộ đệm dịch (TLB). Mỗi entry định nghĩa một ánh xạ từ một số trang ảo tới một số trang vật lí. Ngoài ra, có một số bit phụ trợ để kiểm soát truy cập (valid và read-only) và một số bit để lưu thông tin sử dụng (use và dirty).
  + “virtualPage”: số trang ảo trong bộ nhớ ảo.
  + “physicalPage”: số trang vật lí trong bộ nhớ vật lí (tính từ đầu của “mainMemory”).
  + “valid”: đánh dấu xem entry này có hợp lệ không. Nếu bit này được set, thì dịch sẽ bị bỏ qua.
  + “readOnly”: đánh dấu xem trang này có được phép chỉ đọc không.
  + “use”: đánh dấu xem trang này đã được tham chiếu (sử dụng) hay chưa.
  + “dirty”: đánh dấu xem trang này đã được sửa đổi (dirty) hay chưa.
* Lớp “Machine”:
  + “ReadMem”: đọc “size” (1,2,4) byte từ bộ nhớ ảo tại địa chỉ “addr” vào vị trí được trỏ bởi “value”. Phương thức này trả về FALSE nếu quá trình dịch từ bộ nhớ ảo sang bộ nhớ vật lí thất bại.
  + “WriteMem”: ghi “size” byte từ dữ liệu được trỏ bởi “value” vào bộ nhớ ảo tại địa chỉ “addr”. Phương thức này trả về FALSE nếu quá trình dịch từ bộ nhớ ảo sang bộ nhớ vật lí thất bại.
  + “Translate”: dịch một địa chỉ bộ nhớ ảo thành một địa chỉ bộ nhớ vật lí, sử dụng entries trong bảng trang hoặc TLB. Nếu mọi thứ đều ổn, phương thức này sẽ cài đặt các bit use/dirty trong entry của bảng dịch và lưu địa chỉ bộ nhớ vật lí được dịch vào “physAddr”. Nếu có lỗi, nó sẽ trả về loại ngoại lệ.