# Operating System Lab Project
# Theme B
# Virtual Memory for the Geek OS

Rohit Kumar
120050028
rohit@cse.iitb.ac.in

Deependra Patel
120050032
deependra@cse.iitb.ac.in

Jayesh Bageriya
120050022
jayeshbageriya@cse.iitb.ac.in

Abhishek Thakur
120050008
abhishekthakur@cse.iitb.ac.in

April 1, 2015

## Disk Simulator

## Swap Space

## Page Table Design

## Handle page faults

## Algorithm for page replacement

## OS thread for managing free frames of RAM

## Demo

# 1 Disk Simulator

A main controller controls and executes all the functionality provided.

The main controller provides a mapping of page number to disk addresses. It has an access to a cache controller which manages the disk cache and interfaces between the disk-system and the main controller. The cache controller checks if the data is present in the cache and returns it. If not present, it passes the request to a disk controller which simulates the disk-system. The disk controller passes this request to a request buffer which in turn processes these requests using the elevator algorithm.

While processing the requests various logs and data relating to measure the performance parameters etc. are also being stored.

The details of the various data structures and algorithms used in these classes are described in the below section.

## 1.1 Main Controller

The main controller stores a mapping of page numbers of a particular file to the disk address. To do this we use a map.

Whenever a read request is made for a particular page by the paged file module, MC checks if there is an address corresponding to this page is present in the map. If present, a read request is made to the disk controller, otherwise it is an invalid request and it is dropped.

Whenever a write request is made, if the mapping is present then that means that the file is present and is being overwritten and if mapping is not present then a new mapping is assigned to the page randomly from the set of available free addresses in the disk.

If a file is being deleted from the disk then its mapping is deleted from the map.

This main controller passes the requests to the disk controller for performing the operations.

## 1.2 Disk-System Controller

Disk Controller handles all the requests(read/write) made to the disk. The input disk address is first translated into corresponding disk and CHS(Cylinder Head Sector) format. For calculating CHS, we do as follows:

### 1.2.1 Disk Number and CHS conversion

We have only one disk in our system.

The disk has N cylinders, M tracks/cylinder, and K sectors per track. Then the disk address of this sector can be calculated as :
c is the cylinder number. c belongs to [0, 1, 2, ... N - 1]
t is the head/track number. h belongs to [0, 1, 2, ... M - 1]
s is the sector number. s belongs to [0, 1, 2, ... K - 1]

So, given the address (a), CHS can be calculated as:
$c = a/(M*K)$
$t = [\ a/K\ ]$ modulo M
$s = a$ modulo K

A disk object is created and read/write operations are done on it.
Read Sector - Sector is read from the disk and data is returned back.
Write Sector - The requested block is written.

After any operation, one pending request from buffer of each disk is executed.

### 1.3 Request Buffer and Elevator algorithm

There is a request buffer which contains all the requsets and it is processed using the elevator algorithm. A data-structure for every request is created which also stores the time-stamp of the request.

Upon serving the requests for a particular cylinder, the request structure is deleted and the arm will now move to the next closest request in the direction of movement. When IO is completed an I/O interrupt will be raised, kernel will check which process's IO is completed and make that process ready.

## 2 Swap space

The swap space will be located on the disk. Whenever a process runs out of pages and can't request new page, old page will be paged out to swap space. Whenever a process is made by os, the code and data of the process is first copied to swap space. On request of some address not found in physical memory (page table gives invalid bit), the page has to brought in main memory. We keep a list of all pages of a process here.

## 3 Page table design

There will be a multilevel page table for translation of logical address to physical address. The root table (main table) will contain the address/pointer of another table which contains the corresponding entry of physical address (or another table which stores the entry in case of higher level). We will compare the performance achieved for various depths/levels (2 and 3) and accordingly program it to achieve best efficiency.

Each entry in page table contains a valid bit that corresponds to whether the entry (logical address) is present in memory or not. This will be updated in when a page of a process is swapped in and swapped out in a hierarchical manner i.e. if a page is swapped out, the valid bit for the base table will be set to 0. If all the valid bits of a table become 0 then this table will be freed/de-allocated from memory and the valid bit of it's parent table will be set to 0 implying that the pages in whole range (which the parent table entry contains) is not contained in the table.

Similar approach will be followed when a page is swapped in. The valid bit would be set to 1. This might require creating of a new table if the table also, and that also will be done in hierarchical manner.

This hierarchical approach will be followed for dirty bits also which will be useful when a page has to be swapped. If parent's table entry has dirty bit as 0 then when this set of pages is replaced, then it doesn't needs to be swapped out into swap space as the whole set of pages enclosed hasn't been written.

## 4 Handle page faults

When the page is not found in memory, it has to be brought from disk. When MMU finds a logical address, it tries to convert that into logical address. It uses page table of the process to translate but doesn't find an entry into page table [after not finding into TLB]. So it will raise a syscall, kernel subroutine to handle page fault will start.

Page fault handler will try to allocate a page in memory. It will block the process and bring page from disk into main memory.

This will also update the page table with new physical addresses.

## 5 OS thread for managing free frames of RAM

This kernel process will keep running in background. It will keep track of free-list of pages. When a process requests new page, it will allocate from free-list if available otherwise process will be swapped out.

Whenever a process frees a page [eg. freeing page of a page table when all pages of that table are swapped out], it gets added to freelist. At the end of execution of process, all pages used by process will be freed and added to free-list.

## 6 Page Replacement

We are going to implement Two handed chance/clock algorithm for page replacement. It is a pseudo-LRU algorithm which gives reasonable efficiency and also has lesser overhead as compared to LRU.

The algorithm :- There are two clock hands, one that sets bit associated with each page in process to 0 while the other one finds for the page whose bit is set 0, which it replaces. Whenever a page is found (hit) in memory, the bit corresponding to that page is set to 1.

## 7 TLB [Translation Look-aside Buffer]

TLB will be implemented for faster translation from virtual address to physical address. We will keep direct mapped TLB for lower overhead. It will have fields - virtual addr, physical addr, valid bit, dirty bit (for write back while replacement), permission bit.

On context switch, entire TLB will be flushed.

## 8 Demo

The entire module can be tested by opening a large file which initially will give page fault. Later more pages will be paged in. Process will get blocked when pages are being loaded from