Nachos – Implementing a minimal OS Kernel

OS Project-Abstract

Group members:
121002 : Aakanksha Agrawal
121005 : Anokhi Gandhi
121014 : Geet Patel
121015 : Harsh Prajapati
121028 : Mansi Patel
121031 : Nishant Patel
121037 : Priya Shah
121038 : Priyanka Shah
121050 : Shikhar Pandya
121061 : Varsh Thaker

Introduction:

NachOS (Not Another Completely Heuristic Operating System) is an instructional software developed for the understanding of students and experience the implementation of real operating system.

The main difference between Nachos and real operating system is that basically nachos runs on Unix Process while real operating system runs on a real machine. It is like a guest OS running on host OS.

There are 5 main modules which plays important role to form a system.

1. Processes / Environments.
2. Multiprogramming and Forks.
3. Network Drivers.
4. Virtual memory.
5. File Systems.

We provide a basic understanding of each of the module and explain it briefly from the point of view of nachos throughout the abstract.

1 Processes / Environments

A program in execution is called process.

Components of the process:

- Object Programs : The set of programs that a process need to execute.
- Data : To execute particular program, data is required.
- Resources : While executing some program, it may require some resources.
- Status of the process Execution : Component is used for verifying the status of the process execution.

When process executes, it changes state. There are different 5 states of the process execution.

- New : Process has just been created.
- Ready : Processes which are in the ready queue (i.e. they have not been allocated CPU cycles).
- Running : The process currently under execution.
- Blocked : The processes that cannot be executed because they are waiting for some I/O operation to finish first.
- Terminated : The processes which have finished their execution.

The change of state for a particular process happens when the operating system place the process PCB in the list that corresponds to the new state.

Handling the suspended processes:

The suspended processes are not available for execution immediately.
Swapping is used to move all of a process from main memory to disk and vice-versa.
The following things trigger a process to get into suspension:-

- Swapping: OS requires main memory to create a process or bring in a process for execution.
- Timing: While waiting for next time interval.
- Parent Process request

->We intend to implement the above state transitions and also consider the appropriate situations wherein we need to do so.

Scheduling queues:

- **Long Term Scheduler:** The job scheduler, it determines which process to be run next. When processes change state from new to ready, it is done by long term scheduler.
- **Short Term Scheduler:** The processes which have memory available with them but have not been allocated CPU, such processes are placed in short term queue and handled by Short term Scheduler. Also known as dispatcher.
- **Medium Term Scheduler:** It removes the processes from memory and handles the swapped out processes.
- **I/O scheduler:** The processes which are waiting for a particular I/O resource are placed in the I/O queue and scheduled by the I/O scheduler.

->We hope to implement the above schedulers.

Context switching:-

One of the primary tasks in process management is to perform context switching. The following information needs to be handled when doing context switching:-

- Program Counter
- Scheduling Information
- Currently used register
- Changed state
- I/O state

The above information needs to be stored appropriately for each process. Apart from these, processes need to be created and deleted dynamically; providing environment for process operation also need to be done. In Nachos the functions for creating, terminating and waiting have already been implemented.

- spaceID Exec (char *executable, int pipectrl);

This function creates a user process by creating a new address space. It does so by reading the executable file into it and creating a new internal thread to run it. It both creates a new process and executes a specified program within its context.

- Void exit (int status)

A user process calls exit to indicate that it has finished its operation.

- Int Join(SpaceID joineeID)

This is called by a process to wait for another process to finish its execution.

-> We would like to replace these functions by our own functions.

2 Multiprogramming and Forks:

Multiprogramming is a technique to achieve concurrency between processes. Initially when a single process is running and it makes a system call, the CPU sits idle hence reducing the CPU efficiency. Multiprogramming accelerates the throughput of the system by efficiently using the CPU time. It allows other processes to execute simultaneously while one or more processes are in the wait state.

At present the Nachos assumes the existence of one user at a time, i.e. it uses a one to one mapping scheme. Only 1 program at a time can be loaded form virtual memory to main memory for execution. Hence only 1 process can run at any given time which wastes the CPU cycle. Hence multiprogramming needs to be appended within Nachos. To achieve this we need to maintain synchronization between processes.

There is uniprogramming already implemented in nachos. There are some variables which are defined for uniprogramming in AddrSpace class which needs to be modified. This is done while translating and loading the instructions of a process from prrogram m emory to main memory. Some of them are:

1. pageTable[i].virtualPage

2. pageTable[i].physicalPage

This 2 variables are assigned similar values which means just one process or instruction is loaded at one particular time. This variables / values can help achieve multiprogramming. We would modify/replace the AddrSpace class / variables to achieve multiprogramming and concurrency between processes.

3 Network Drivers:

Networking basically means communicating with other processes through a network. Here in Nachos we have to build a network between one computer system with various other computer systems.

Nachos contains various classes for networking. Some of them are :

- Post Office Input and Post Office Output
- Mailbox
- Mail and mail header

Driver development for networking contains the following steps :-

- Detection of the device
- Enabling the device
- Understanding the device
- Bus-independent service access
- Initializing the network device
- Making it ready for transmitting and receiving data.

4 Virtual Memory:

Virtual memory provides a way of running more processes than can physically fit within a computer's physical address space.

This functionality gives processes the illusion of a virtual memory that is larger than the available machine memory.

Implementation:

We will implement and debug virtual memory in two steps.

- Demand paging using page faults to load process from logical memory to physical memory on demand.
- Page replacement, to evict a virtual page from memory to free up a physical page frame to satisfy a page fault.

5 File System:

A file system consists of methods and data structures that an operating system uses to keep track of the files on a disk, i.e., the manner in which files are organized on the disk.

File system is also used to refer to a partition or disk that is used to store the files or the type of the files system.

File system in Nachos:

File system implementation in Nachos comes with two versions: a ‘stub’ and a 'real' version.
The stub version basically uses the UNIX file system operations, provided on the machine where Nachos is running. The 'real' file system version is built on top of a disk simulator. The disk is simulated using the UNIX file system.
In Nachos, there are several classes and structures that describes the file system. The most important ones are:

- FileSystem
- OpenFile
- SynchDisk
- FileHeader

- Directory

Implementing file systems in Nachos:

The second version allows students to implement a file system on top of a raw disk, which is in fact a regular Unix file that can only be accessed through a simulated disk.

On disk, both the list of free sectors and the top-level directory are stored within regular Nachos files. The free list is stored as a bit map, one bit per sector, with the file itself stored in fnode 0. Thus, finding a free sector in the file system requires reading the file associated with fnode 0, using the bitmap functions to locate free sector(s), and then flushing the file changes back to disk to make the changes permanent.

Reference:

- http://cseweb.ucsd.edu/classes/fa14/cse120-b/projects/vm.html
- http://ecomputernotes.com/fundamental/disk-operating-system/what-is-demand-paging
- https://cseweb.ucsd.edu/classes/fa04/cse120/lectures/replace.pdf
- www.cs.nyu.edu/courses/spring00/V22.0202-001/nachos-labs.htm#_Toc472677485
- www.mu.ac.in/myweb_test/MCA%20study%20material/OS%20-%20PDF.pdf
- http://cs.gmu.edu/cne/modules/vm/green/multip.html
- https://www.ida.liu.se/~TDDI12/material/begguide/
- www.tldp.org/LDP/sag/html/filesystems.html
- https://www.cs.nyu.edu/courses/spring00/V22.0202-001/nachos-labs.htm#_Toc472677481
- http://www.wisegeek.com/what-is-a-network-driver.htm