Assignment No : 1

Title : Study Experiment: Study of unix Commands.

Problem Statement :

a. Shell commands:

info - how to browse info pages, man - how to browse man pages, ls, cp, paste, diff, cat, find, ps, chown, chmod, echo, bc, head, tail, I/O Redirection, wc, cut, pr, sort, uniq, tee, use of Wildcard Characters, File Access Permissions (FAP)

b. Use of pipes & filters – grep and sed commands

Objectives:

1. To understand how to use Unix commands.
2. To understand How and Why they are used in Shell Programming.

Problems to be solved in the lab:

1. How to browse info pages
2. How to browse man pages.
3. List the contents of directory using *ls* command.
4. Copy contents of one file into the other file using *cp* command.
5. Merge the contents of data of two files using *paste* command.
6. Making two files similar using *diff* command.
7. How to create a new file and display contents of file using *cat* command.
8. To change file owner and file permissions using *chown* and *chmod.*
9. Perform calculations using *bc* command.
10. To study filter commands such as *head,* and *tail* for extracting the contents of the file from the beginning or the end.
11. Count the number of people logged in and also traps the users in a file using the *tee* command.
12. To search a pattern from the file using *grep* and *sed* commands.

Applications :

1. To enable the user to communicate with the kernel through the command interpreter.
2. Useful in Shell Programming

FAQS

1. What is a pipe?
2. What is a filter?
3. What is the purpose of the grep command?
4. How does input output redirection take place?

References :

- UNIX by Sumitabha Das
  Publisher - Tata McGraw Hill
- Manual Pages in Linux System

Assignment No : 2

Title : Shell Programming and AWK programming.

Problem Statement :_

a. Write a shell script to find single digit sum of four digit number entered through keyboard (ex. 1234 -> 1+2+3+4 = 10)

b. Write an AWK program which will accept employee as database file containing empNo, empName, age, designation, basicSal, HRA is 15% of basicSal, DA is 35% of basicSal, grossSal is basicSal + DA + HRA. Database should contain at least 10 records.

Print those records where grossSal < 15000.

Objectives :

1. To understand how to perform Shell programming in Unix/Linux.
2. To explain the purpose of shell programs
3. Design and write shell programs of moderate complexity using variables, special variables, flow control mechanisms, operators, arithmetic and functions.
4. To understand how to design & develop AWK Programs in Unix/Linux.

Algorithm :

Shell Program:

1. Accept any four digit number from the user.
2. Raise error if the number is not of four digit number.
3. Find the sum of digits.
4. Display the Result on the Screen.
5. Stop.

awk Program:

1. Create a file which contains the following information of employees empId, empName, designation, basicSal with at least 10 rows.
2. Calculate the DA, and HRA of the employee by performing operation on the column of basicSal and add them as column 5 and column 6 in the same file.
3. Calculate the grossSal and add it in the same file as column 7.
4. Print those records where grossSal < 15000.
5. Stop.

Applications :

1. Shell - Useful for System Administrator.
2. awk - It is useful for report generation.

FAQS :

1. What are different control structures used in shell programming?
2. What do you mean by positional parameters & enlist them?
3. What do you mean by shell?
4. For what purpose is AWK programming used?
5. Why is AWK more useful compared to shell programming when dealing with database files?
6. What do you mean by $1, $2,$3..etc in AWK programming?
7. What are the different built in variables?

References :

- UNIX by Sumitabha Das

  Publisher - Tata McGraw Hill
- Manual Pages in Linux System

Assignment No : 3

Title : Implementation of Process Scheduling Algorithms.

Problem Statement :

Implement following Algorithms:

1. Priority (Non Preemptive)
2. SJF (Preemptive)
3. RR

Objective :

1. To understand how processes are actually scheduled in the system.
2. To understand different scheduling criteria like waiting time and turn around time on which the scheduling algorithm is selected.

Algorithm :

Preemptive Shortest Job First (SJF) :

1. Accept the number of processes from the user.
2. Accept arrival time and burst times for each process.
3. Sort all the processes according to the arrival time.
4. Start with the first process.
5. After the first time slice of the process, if any other process has less arrival time then execute that.
6. Continue this process till all the processes are completed.
7. Display turnaround time, waiting time for each process, and average turnaround time, average waiting time.
8. Display Gantt chart.
9. Stop.

Round Robin :

1. Accept the number of processes from the user.
2. Accept arrival time, burst times for each process and time quantum.
3. Execute the process present at the head of ready queue for one time slot.
4. Check if some process has arrived. If so; preempt the running process and allocate the CPU to the next process.
5. Insert the selected process at the tail of ready queue.
6. Continue steps 3 to 5 until all the processes are completed.
7. Display turnaround time, waiting time for each process, and average turnaround time, average waiting time.
8. Display Gantt chart.
9. Stop.

Non- Preemptive Priority:

1. Accept the number of processes from the user.
2. Accept arrival time, burst times and priority for each process.
3. Select job with highest priority in the system available.
4. Complete the selected process.
5. Continue steps 4 & 5 until all processes are completed.
6. Display turnaround time, waiting time for each process, and average turnaround time, average waiting time.
7. Display Gantt chart.
8. Stop.

Input :

1. Number of Processes in the System.
2. Burst time of each process.
3. Arrival time for each process.
4. Time Quantum (for Round-Robin Algorithm).
5. Priority of each process (for Priority Algorithm).

Output :

For each algorithm output should be in the form of:

1. Gantt chart showing execution of processes.
2. A table showing Waiting time and Turn-Around Time of each process.
3. Average Waiting time and Average Turn-Around Time.

Applications :

Operating system use scheduling algorithms in order to provide good response to users.

FAQS :

1. Why is scheduling used?
2. What is the difference between preemptive & non- preemptive scheduling?
3. Which algorithm is more useful & why?
4. What do you mean by time quantum?

References :

- Operating System Principles by Silberschatz, Galvin, Gagne
  Publisher - Wiley Publication
- Operating Systems by William Stalling
  Publisher - Prentice Hall India

Assignment No : 4

Title : To implement Banker's Algorithm.

Objective :

To develop a generalized solution which accepts n processes and m resource types with multiple instances of each resource type. The solution should be able to determine the safety of the system after granting the request, and should be able to deny the request if system is moving to unsafe state.

Algorithm :

Resource-Request Algorithm

$Request_i$ = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

   $Available = Available - Request_i;$

   $Allocation_i = Allocation_i + Request_i;$

   $Need_i = Need_i - Request_i;$
4. Call Safety Algorithm
   - *If safe ⇒the resources are allocated to Pi.*
   - *If unsafe ⇒Pi must wait, and the old resource-allocation state is restored*

Safety Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:
   Work = *Available*
   Finish [*i*] = *false* for *i* = 0, 1, …, *n*- 1.
2. Find and *i* such that both:
   (a) *Finish* [*i*] = *false*
   (b) $Need_i \leq Work$
   If no such *i* exists, go to step 4.
3. $Work = Work + Allocation_i$
   Finish[*i*] = *true*
   go to step 2.
4. If *Finish* [*i*] == true for all *i*, then the system is in a safe state.

Input :

1. Number of Process competing for the resources.
2. Number of instances of each Resource type available (Available Vector).
3. Number of instances of each Resource type allocated to processes (Allocation Matrix)
4. Maximum number of instances of each Resource type required by each process (Max Matrix).

Output :

A Menu-Driven Output with following menu:

1. Display Current Allocation Matrix.
2. Check the state of the system.
3. Make a Request.
4. Exit.

Applications :

Can be used in a system where prior information regarding usage of resources for different processes is known in advance.

FAQS :

1. What do you mean by deadlock?
2. What are 4 conditions necessary for deadlock existence?
3. What are time complexities of deadlock avoidance and deadlock detection algorithm?

References :

- Operating System Principles by Silberschatz, Galvin, Gagne
  Publisher - Wiley Publication
- Operating Systems by William Stalling
  Publisher - Prentice Hall India

Assignment No: 5

Title : Implementation of Reader/Writer problem using semaphore and threads

.

Scope :

1. Implement First Reader / Writer Problem.
2. Create at least 5 Reader Threads.
3. Create at least 3 Writer Threads.

Objectives :

1. To understand the solution to the problems of mutual exclusion.
2. To grasp techniques and to develop the skills in the use of the tools: semaphores, threads, mutex in concurrent programming.

Algorithm :

PART A]

PseudoCode For Readers :

```
var wrt, mutex: semaphore;
    readcount: integer;
    begin
    readcount := 0;
    wrt := 1;
    mutex := 1;
    parbegin
    repeat                          (* reader process *)
    (* do something *)
    wait(mutex);
```

```
readcount := readcount + 1;
if readcount = 1 then
  wait(wrt);
signal(mutex);
(* read the Shared Data *)
wait(mutex);
readcount := readcount - 1;
if readcount = 0 then
signal(wrt);
signal(mutex);
(* do something else *)
until false;
end
```

PART B]

Pseudo code For Writers:

```
repeat (* writer process *)
(* do something *)
wait(wrt);
(* write to the Shared data *)
signal(wrt);
(* do something else *)
until false;
parend;
end.
```

Input :

1. No. of Readers in the system
2. No. of Writers in the system.
3. Shared data to be used .

Output :

Output produced by Readers & Writer threads depicting contention free content of shared data.

Applications :

Applicable in all applications where synchronization is used such as all Operating systems.

FAQS :

1. What is difference between thread and process?
2. What mean by is semaphore?
3. What is mean by critical section?
4. How will you change the solution for n readers and n writer?

References :

- Operating System Principles by Silberschatz, Galvin, Gagne
  Publisher - Wiley Publication
- Beginning Linux Programming by Neil Mattew and Richard Stones
  Publisher - Wrox Publication

Assignment No : 6

Title : To implement Dinning Philosopher's problem using threads.

Scope :

1. Create 5 threads for representing 5 philosophers
2. Make use of semaphore variables to represent the chopsticks.
3. Implement the algorithm which may lead to deadlock as well as the algorithm which overcome this drawback by implementing asymmetric solution.

Objectives :

How to solve the synchronization problem using multithreading

Output :

Message depicting whether particular Philosopher is Thinking or Eating.

Algorithm :

This algorithm which may lead to deadlock wherein chopstick is array of semaphore and i being philosopher thread under execution.

```
While (true)
{
        // think
        wait ( chopstick[i] );
        wait ( chopstick[ (i + 1) % 5] );
                // eat
         signal ( chopstick[i] );
         signal (chopstick[ (i + 1) % 5] );
  }
```

Applications :

Applicable in all applications where synchronization is used such as all Operating systems.

FAQS :

1. What do you mean by multithreading?
2. What is the Dining Philosopher's problem?
3. If there are *n* philosophers, what is the maximum number of philosophers that can eat at a time?

References :

- Operating System Principles by Silberschatz, Galvin, Gagne
  Publisher - Wiley Publication
- Beginning Linux Programming by Neil Mattew and Richard Stones
  Publisher - Wrox Publication

Assignment No : 7

Title : To simulate page replacement policies.

Scope :

To implement following Algorithms:

1. LRU
2. OPTIMAL
3. MFU

Objective :

1. To understand the concept of demand paging.
2. To understand how pages are selected for replacement.

Algorithm :

Most Frequently Used (MFU):

1. Start.
2. Accept references string and number of page frames in memory.
3. Initialize a count for each page in main memory to zero.
4. Check if the page is already present in the main memory; if so increment its count by one.
5. If the page is not present, find the page in memory with maximum count, which is the most frequently used and replace it.
6. Set its count of the newly loaded page to zero.
7. Repeat from step 4 to 6 till all pages in string are over.
8. Stop.

Optimal:

1. Start.

2. Accept the references string and number of page frames in memory.
3. When a page is referred, check if the page is already present in the main memory.
4. Allocate frame to new page referred if free frame available else goto step 5.
5. If the page is not present, find the page which will not be used for longest period.
6. Repeat steps 3 to 5 till all pages in string are over.
7. Stop.

Least Recently Used (LRU):

1. Start.
2. Accept the references string and number of page frames in memory.
3. Initialize a stack of size equal to frames in memory.
4. When a page is referred, check if the page is already present in the main memory.
5. If present; move it to the top of the stack
6. If the page is not present, remove the page from the bottom of the stack which is least recently used page.
7. Insert the new page on the top of the stack.
8. Repeat steps 3 to 5 till all pages in string are over.
9. Stop.

Input :

1. Number of Page Frames in memory in case of MFU and OPTIMAL or Size of Stack in case of LRU.
2. Reference String (Page referred by process during execution).

Output :

1. Graphical output showing allocation of frames to pages in case of MFU and OPTIMAL and Stack position with every reference in case of LRU.
2. Number of page faults occurred.

Applications :

Used in memory management in Operating Systems.

FAQS :

1. What do you mean by virtual memory?
2. What is demand paging?
3. How page-faults are handled?
4. What is thrashing?
5. What is Belady's anomaly?

References :

- Operating System Principles by Silberschatz, Galvin, Gagne
  Publisher - Wiley Publication
- Operating Systems by William Stalling
  Publisher - Prentice Hall India

Assignment No : 8

Title : To simulate disk scheduling Algorithms.

Scope :

Implement following Algorithms:

1. FCSF
2. C-SCAN
3. SSTF

Objectives :

To understand how efficiency of the system is increased by scheduling the disk requests.

Algorithm :

First Come First Serve (FCFS):

1. Start.
2. Accept the number of tracks *n*.
3. Accept the requested tracks and store it in the track [ ].
4. Consider the first value of track[ ] as starting track .
5. Process the track values in given order to calculate difference as trackdiff [i] = track[i] - track[i+1] .
6. Calculate totaldiff = totaldiff + trackdiff[i].
7. Calculate average seek time.
8. Display the value of average seek time.
9. Stop.

Shortest Service Time First (SSTF):

1. Start.
2. Accept the number of tracks *n.*
3. Accept the requested tracks and store it in track[ ].
4. Consider the first value of track[ ] as starting track .
5. First process all the tracks which are having value less than starting track in decreasing order of tracks as trackdiff[i] = track[i]-track[i+1] .
6. Then process all the tracks which are having value greater than starting in increasing order of track as trackdiff[i] = track[i] - track[i+1].
7. Calculate totaldiff = totaldiff + trackdiff[i]
8. Calculate average seek time.
9. Display the value of average seek time.
10. Stop.

C-SCAN:

1. Start.
2. Accept the number of tracks *n.*
3. Accept the requested tracks and store it in track[ ]
4. Consider the first value of track[ ] as starting track .
5. First process all the tracks which are having value greater than starting track in increasing order of track as trackdiff[i]= track[i] - track[i+1]
6. Then process all the tracks which are having value less than starting track in increasing order of track(Wrap around to starting track) as trackdiff[i] = track[i] - track[i+1]
7. Calculate totaldiff = totaldiff + trackdiff[i]
8. Calculate average seek time.
9. Display the value of average seek time.
10. Stop

Input :

1. Number of cylinders in the disks
2. Sequence of disk requests

Output :

1. Graphical output showing how the requests were satisfied.
2. Average seek-time.

Applications :

1. Applicable in Multimedia application to reduce disk access.

FAQS :

1. What do you mean by seek time?
2. What is difference between SCAN & C-SCAN method?
3. Give an analysis of comparison of 4 algorithms.

References :

- Operating System Principles by Silberschatz, Galvin, Gagne
  Publisher - Wiley Publication
- Operating Systems by William Stalling
  Publisher - Prentice Hall India

Assignment No : 9

Title : A program to implement inter process communication using pipe system call.

Objective :

1. To demonstrate the IPC between two different processes.
2. To understand how to make use of fork ( ) system call to create the new process.
3. To understand how to make use of pipe ( ) system call as a medium of IPC between parent and child processes.

Algorithm :

1. Create the child process using fork( ) system call
2. Accept the input data in parent process.
3. Create the pipe as a medium of IPC using pipe ( ) system call.
4. Parent process should send the accepted data through write end of created pipe to child process.
5. Child process should receive the sent item through read end of pipe & perform necessary operations on it & print the result.

Input :

Data requested by parent process.

Output :

Data manipulated by child process

References :

- Beginning Linux Programming by Neil Mattew and Richard Stones
  Publisher - Wrox Publication
- Manual Pages in Linux System