LAB MANUAL FOR OPERATING SYSTEMS

(CSPC-20)

EXPERIMENT-1

1. **1.1** Study of hardware and software requirements of different operating systems (UNIX, LINUX, WINDOWS XP, WINDOWS 7/8). (1 lab session)
   1.2 Execute various UNIX system calls for - (1 lab session)
   - **a**. Process management
   - **b**. File management
   - **c**. Input / Output System calls

The **OBJECTIVE** of this practical is to obtain general overview of various popular OS .

(i) Their development and distribution
(ii) Compatibility
(iii)Security issues and Thread detection and its solution
(iv)The GUI etc..

IMPLEMENTATION

(i) Along with the above mentioned activities execution of various UNIX commands is also helpful:
(ii) cat, cd, cp, chmod, df, less, ls, mkdir, more, mv, pwd, rmdir, rm, man, uname, who, ps, vi, cal, date, echo, bc, grep

EXPERIMENT-2

2. Implement CPU scheduling policies – (2-3 lab sessions)
2.1 SJF
2.2 Priority
2.3 FCFS
2.4 Multi-level queue

INPUT/s:

2.1 The number of processes in the system

2.2 The CPU Burst, priority, arrival time of process.

IMPLEMENTATION

2.1 For **SJF** algorithm,

i) We randomly generate the number of jobs. There must be a limit on the number of process in a system.

ii) The execution time of the generated jobs is also not known. These jobs may vary in their real execution. Here, we are randomly generating the execution time of each job making use of the past history.

All the jobs are then arranged in a queue where searching is done to find the one with the least execution time. There may be two jobs in queue with the same execution time then FCFS is to be performed.

Case a) If the algorithm is **non preemptive** in nature, then the newly arriving job is to be added to the job queue even though it is of lesser execution time than the one running on the processor.

Case b) Otherwise **preemption** is performed.

2.2 For **Priority scheduling**, we prefer to compute the CPU burst rather than manually entering it. Priority may be assigned on the basis of their CPU burst or some other parameter. **Priority (preemption)** and **priority (non preemption)** nature of priority scheduling is performed.

2.3 The **FCFS scheduling** is performed on the basis of arrival time irrespective of their other parameters.

2.4 In **multi-level queue scheduling**, different queues are to be created

OUTPUT/s:

The average throughput, average turnaround time, waiting time of process/s.

EXPERIMENT-3

3. Implement file storage allocation techniques – (2 lab session)
3.1 Contiguous (using array)
3.2 linked –list (using linked list)
3.3 indirect allocation (indexing)

INPUT/s :

Free storage blocks to allocate storage to files.

ASSUMPTIONS/ CONSIDERATION:

3.1 Contiguous allocation strategy is implemented using array data structure.

3.2 Linked list allocation technique is implemented using linked list.

3.3 Indirect allocation is performed using indexing concept.

3.4 For performing the given algorithm, we consider files from secondary storage or the process's memory requirement for its execution.

OUTPUT/s:

Files/ Programs are allocated storage space through appropriate storage allocation techniques.

EXPERIMENT-4

4. Implementation of Contiguous allocation techniques – (2 lab session)
4.1 Worst-fit
4.2 Best-fit
4.3 First-fit

INPUT/s

1. Free space list of blocks from system.
2. List processes and files from the system.

IMPLEMENTATION CONSIDERATION:

4.1 We consider some free space list of blocks from system.

4.2 Now consider processes and files from the system or stored on some storage media. Obtain their size by making use of some system calls.

4.3 Implement the above mentioned three contiguous allocation techniques.

4.3.1 worst-fit: In worst fit technique largest available block/partition which will hold the page is selected. Blocks are sorted according to their size in descending order.

4.3.2 best-fit: Best-fit is one of the optimal technique in which page is stored in the block/partition which is large enough to hold it. Blocks are sorted according to their size in ascending order.

4.3.3 first-fit: In first-fit technique page is stored in the block which is encountered first that is big enough to hold it.

4.4 Also, list the amount of free space blocks left out after performing allocation.

OUTPUT/s:

Processes and files allocated to free blocks. List of those processes and files which is not allocated memory. List of free space blocks, left out after performing allocation.

EXPERIMENT-5

5. Calculation of external and internal fragmentation. (1 lab session)

It is assumed that files are generated randomly having varying size. Every time program is executed for different page size.

INPUT/s

1. Free space list of blocks from system.
2. List processes and files from the system.

IMPLEMENTATION CONSIDERATION:

5.1 We consider some free space list of blocks from system.

5.2 Now consider processes and files from the system or stored on some storage media. Obtain their size by making use of some system calls.

5.3 Implement the above mentioned three contiguous allocation techniques one by one.

5.4 After implementing each allocation algorithm, list the amount of free space blocks left out after performing allocation.

5.5 When a block which is not at all allocated to any process or file, it adds to external fragmentation.

5.6 When a file or process is allocated the free block and still some part of it is left unused, we count such unused portion into internal fragmentation.

OUTPUT/s:

Processes and files allocated to free blocks. From the list of unused blocks, we determine the count of total internal and external fragmentation.

EXPERIMENT-6

6. Implementation of Compaction for the continually changing memory layout and calculate total movement of data. (1 lab session)

Compaction is a technique used to remove internal fragmentation. Assumption that has to be taken into consideration is that the files must be inserted and deleted continuously. User must provide memory image at different instances of time.

In the practical no 5, we have obtained total internal and external fragmentation. To the above practical, we continue performing the compaction. Hereby, this activity is left for students to think and perform compaction to the above practical.

EXPERIMENT-7

7. Implementation of resource allocation graph. (1 lab session)

INPUT/s:

1. List of resources
2. Instance of each resource
3. List of processes
4. Resource allocated by each process
5. Resource request by each process

IMPLEMENTATION:

We have to consider two cases have to be:–

Case 1 – Each resource has single instance

Case 2 – Multiple instances of resource

Two methods are used for storing graph -

a. **Ajdacent matrix**: a 2-D array of size NxN where N is the number of vertices in the graph (includes processes and resources). For each adj[i][j] = 1 indicates that there is an edge from vertex I to vertex j. Since resource allocation graph is directed graph, hence it is not necessary to be symmetric.
b. **Adjacent list:** An array of linked list is used. Size of the array is equal to number of vertices (processes) in the graph. An entry arr[i] represents the linked list of vertices (resources requested by process) adjacent to the i^{th} vertex.

OUTPUT/s:

Output is a Resource allocation graph.

EXEPERIMENT-8

8. Implementation of bankers algorithm. (2 lab sessions)

INPUT:

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

- Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
- Max: An $n \times m$ matrix defines the maximum demand of each process. If Max[i,j] = k, then P_i may request at most k instances of resource type R_j.
- Allocation: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process P_i is currently allocated k instances of resource type R_j.
- Need: An $n \times m$ matrix indicates the remaining resource need of each process. If Need[i,j] = k, then P_i may need k more instances of resource type R_j to complete the task.

Note: Need[i,j] = Max[i,j] - Allocation[i,j].

IMPLEMENTATION:

We call Banker's algorithm when a request for R is made. Let n be the number of processes in the system, and m be the number of resource types.

Define:

- available[m]: the number of units of R currently unallocated (e.g., available[3] = 2)
- max[n][m]: describes the maximum demands of each process (e.g., max[3][1] = 2)
- allocation[n][m]: describes the current allocation status (e.g., allocation[5][1] = 3)
- need[n][m]: describes the *remaining* possible need (i.e., need[i][j] = max[i][j] - allocation[i][j])

Resource-request algorithm:
Define:

- request[n][m]: describes the current outstanding requests of all processes (e.g., request[2][1] = 3)

1. If request[i][j] <= need[i][j], to to step 2; otherwise, raise an error condition.
2. If request[i][j] > available[j], then the process must wait.
3. Otherwise, *pretend* to allocate the requested resources to P_i :
4. available[j] = available[j] - request[i][j]

5. allocation[i][j] = allocation[i][j] + request[i][j]
   need[i][j] = need[i][j] - request[i][j]

   Once the resources are *allocated*, check to see if the system state is safe. If unsafe, the

   process must wait and the old resource-allocated state is restored.

Safety algorithm (to check for a safe state):

1. Let work be an integer array of length m, initialized to available.
   Let finish be a boolean array of length n, initialized to false.
2. Find an i such that both:
   - finish[i] == false
   - need[i] <= 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, otherwise unsafe.

OUTPUT

Detection process specifies if a deadlock is present in system with listed processes and their needs or not.

** Some of the resources that are tracked in real systems are memory, semaphores and some interface access.

EXEPERIMENT-9

9. Conversion of resource allocation graph to wait for graph for each type of method used for storing graph. (2 lab sessions)

One such deadlock detection algorithm makes use of a wait-for graph to track which other processes a process is currently blocking on. In a wait-for graph, processes are represented as nodes, and an edge from process P_i to P_j implies P_j is holding a resource that P_i needs and thus P_i is waiting for P_j to release its lock on that resource. There may be processes waiting for

more than a single resource to become available. Graph cycles imply the possibility of a deadlock.

INPUT/s:

Resource allocation graph (RAG) (as in practical 7)

IMPLEMENTATION

9.1 Identify the waiting processes in the RAG.
9.2 Accordingly draw Wait-for graph for the given RAG.
9.3 We identify circular chain of dependency (i.e., appearance of loops in the graph)

OUTPUT:

The wait-for-graph.

Also, check presence of loop to detect if loop is present.

EXEPERIMENT-10

10. Implementation of FORK and JOIN construct. (1 lab)

- **a.** Program where parent process counts number of vowels in the given sentence and child process will count number of words in the same sentence.
- **b.** Program where parent process sorts array elements in descending order and child process sorts array elements in ascending order.

Parent process and child process can communicate with each other with the help of shared memory. Parent process and child process both will work on the data available in that shared memory and according to them provide their outputs. Two cases will arise according to their sequence of termination –
Case a) Child process waits until parent process terminates.
Case b) Parent process waits until child process terminates.

If child process terminates before parent process, process execution will be unsuccessful.

At a time only one process is executing. Control will not be transferred to another process until one process does not complete its execution i.e. in non-preemptive manner. According to their termination proper message must be displayed. Also show the orphan and zombie states for above questions.

EXEPERIMENT-11

Implementation of semaphores for concurrency. Implementation of Inter Process Communication techniques (implement any one of them) – (2 lab sessions)

- **a.** Bound Buffer
- **b.** Reader-Writer
- **c.** Dining-Philosopher

IMPLEMENTATION:

10.1 For programming this problem, we use JAVA (multi-threading concept) for implementing the synchronization problem using semaphores.

10.2 Our main focus is to obtain three conditions of i) mutual exclusion ii) progress iii) bounded wait.

10.3 Implement semaphore concept considering above mentioned problem.

OUTPUT/s:

Synchronization of the problem satisfying conditions of mutual exclusion, progress and bounded wait.