PART-III
Experiments related to the Operating System

EXPERIMENT–18

Date: 15/02/2022

Aim:- Simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.

a) FCFS

b) SJF

c) Round Robin (pre-emptive)

d) Priority

Algorithm:-

FCFS

1. Input the processes along with their burst time (bt).
2. Find waiting time (wt) for all processes.
3. As first process that comes need not to wait so waiting time for process 1 will be 0 i.e. wt[0] = 0.
4. Find waiting time for all other processes i.e. for process i -> wt[i] = bt[i-1] + wt[i-1] .
5. Find turnaround time = waiting_time + burst_time for all processes.
6. Find average waiting time = total_waiting_time / no_of_processes.
7. Similarly, find average turnaround time = total_turn_around_time / no_of_processes.

SJF

1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.
4. Completion Time: Time at which process completes its execution.
5. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
6. Waiting Time(W.T): Time Difference between turn around time and burst time.
   Waiting Time = Turn Around Time – Burst Time

Round Robin

1. Create an array rem_bt[] to keep track of remaining burst time of processes. This array is initially a copy of bt[] (burst times array)

2. Create another array wt[] to store waiting times of processes. Initialize this array as 0.
3. Initialize time : t = 0
4. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
   1. If rem_bt[i] > quantum
      1. t = t + quantum
      2. rem_bt[i] -= quantum;
   2. Else // Last cycle for this process
      1. t = t + rem_bt[i];
      2. wt[i] = t - bt[i]
      3. rem_bt[i] = 0; // This process is over

Priority

1. First input the processes with their burst time and priority.
2. Sort the processes, burst time and priority according to the priority.
3. Now simply apply FCFS algorithm.

Result:- The given CPU scheduling algorithms are simulated and the average waiting time and turnaround time are calculated.

EXPERIMENT–19

Date: 15/02/2022

Aim:- Implement the banker’s algorithm for deadlock avoidance.

Algorithm:-

Safety Algorithm

```
1. Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
   Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
2. Find an i such that both
   1. Finish[i] = false
   2. Needi <= Work
      if no such i exists goto step (4)
3. Work = Work + Allocation[i]
      Finish[i] = true
      goto step (2)
4. if Finish [i] = true for all i
      then the system is in a safe state
```

Resource Request Algorithm

1. If $Request_i$ <= $Need_i$
   Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If $Request_i$ <= Available
   Goto step (3); otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:
   Available = Available – Requesti
   $Allocation_i$ = $Allocation_i$ + $Request_i$
   $Need_i$ = $Need_i$– $Request_i$

Result:- The banker’s algorithm is implemented for deadlock avoidance.

EXPERIMENT–20

Date: 15/02/2022

Aim:- Simulate the following page replacement algorithms:

a) FIFO

b) LRU

c) LFU

Algorithm:-

FIFO

1. Start traversing the pages.
   1. If set holds less pages than capacity.
      1. Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
      2. Simultaneously maintain the pages in the queue to perform FIFO.
      3. Increment page fault
   2. Else
      1. If current page is present in set, do nothing.
      2. Else
         1. Remove the first page from the queue as it was the first to be entered in the memory
         2. Replace the first page in the queue with the current page in the string.
         3. Store current page in the queue.
         4. Increment page faults.
2. Return page faults.

LRU

1. Start traversing the pages.
   1. If set holds less pages than capacity.
      1. Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
      2. Simultaneously maintain the recent occurred index of each page in a map called indexes.
      3. Increment page fault
   2. Else
      1. If current page is present in set, do nothing.
      2. Else
         1. Find the page in the set that was least recently used. We find it using index array. We basically need to replace the page with minimum index.
         2. Replace the found page with current page.
         3. Increment page faults.
         4. Update index of current page.
2. Return page faults.

LFU

1. Initialize count as 0.
2. Create a vector/array of size equal to memory capacity.

   Create a map to store frequency of pages
3. Traverse elements of pages[]
4. In each traversal:
   1. if(element is present in memory):
      1. remove the element and push the element at the end
      2. increase its frequency
   2. else:
      1. if(memory is full)
         1. remove the first element and decrease frequency of 1st element
      2. Increment count
      3. push the element at the end and increase its frequency
   3. Compare frequency with other pages starting from the 2nd last page
   4. Sort the pages based on their frequency and time at which they arrive
   5. if frequency is same, then, the page arriving first must be placed first

Result:- The given page replacement algorithms are simulated.