IOPS LAB ASSIGNEMENT-6

B. Tech. (ECE) – 3rd Semester; JULY – DECEMBER' 2018

Deadline: 02 November 2018

Course Instructor: Dr. S. MAITY

Assignment on Deadlock Handling and Memory Management & Virtual Memory

Important Instructions:

Group	Q. No.
1	1
2	2
3	3
4	4
5	5
6	6
7	1
8	2
9	3
10	4
11	5
12	6
13	1
14	2
15	3
16	4
17	5

Q1. Based on Banker's algorithm, write and implement an algorithm to detect Deadlock in a system. Your program should take the following inputs:-

- *m* : number of resource types
- *n* : number of processes
- *Available* : an array of length *m* (indicating number of resources available of each type)
- *Allocation* : A (*n* x *m*) matrix (defines number of resources of each type currently allocated to each process)
- *Request* : A (*n* x *m*) matrix (indicates the current request for each resource type by each of the *n* processes)

Output: Print the result of the detection algorithm, i.e., for the above inputs, whether the system is currently in Deadlock state or not. If not, then print a *Safe Sequence* supporting the answer. Assume that the processes are allowed to execute in the order according to your safe sequence. Then show (print) what would be the values in the *Available* array after every process in the sequence finish their executions.

Q2. Implement Banker's algorithm to detect whether a system is in Safe State or not. Your program should take the following inputs:-

- *m* : number of resource types
- *n* : number of processes
- *Available* : an array of length *m* (indicating number of resources available of each type)
- *Allocation* : A (*n* x *m*) matrix (defines number of resources of each type currently allocated to each process)
- *Max* : A (*n* x *m*) matrix (indicates the maximum demand for each resource type by each of the *n* processes)

Output: Print the result of the detection algorithm, i.e., for the above inputs, whether the system is currently in Safe State or not. If not, then print a *Safe Sequence* supporting the answer. Assume that the processes are allowed to execute in the order according to your safe sequence. Then show (print) what would be the values in the *Available* array after every process in the sequence finish their executions.

Q3. Suppose in a memory management system, the page-size is 2^n words and size of the logical address space for a process P is 2^m words. Write an efficient program which will take values of *m*, *n*, and values for an integer array of length 2^{m-n}, (the *Page-Table*), as the input values. Note that *Page-Table*[*i*] indicates the frame number corresponding to page number *i* ($0 <= i <= (2^{m-n}-1)$). Then, for any logical address (in the range 0 and 2^m-1) input by user, the program should output the corresponding physical address.

Q4. Implement the FIFO page replacement algorithm.

Input:

- Reference string of length *m* (*m* can be constant, but very large, for your program)
- *n* : number of allocated frames

Output:

- *f* : Number of page faults
- Show the contents of all the frames after every memory reference

Experiment: Run the program with 10 different values of *n* and record the value of *f* in every case. Save the data in tabular format and plot a graph showing *f* vs. *n*. Generate the reference strings randomly, for the above experiment.

Q5. Implement the LRU page replacement algorithm using counter (logical clock).

Input:

- Reference string of length *m* (*m* can be constant, but very large, for your program)
- *n* : number of allocated frames

Output:

- *f* : Number of page faults
- Show the contents of all the frames after every memory reference

Experiment: Run the program with 10 different values of *n* and record the value of *f* in every case. Save the data in tabular format and plot a graph showing *f* vs. *n*. Generate the reference strings randomly, for the above experiment.

Q6. Implement the LRU page replacement algorithm using Stack (using double linked list).

Input:

- Reference string of length *m* (*m* can be constant, but very large, for your program)
- *n* : number of allocated frames

Output:

- *f* : Number of page faults
- Show the contents of all the frames after every memory reference

Experiment: Run the program with 10 different values of *n* and record the value of *f* in every case. Save the data in tabular format and plot a graph showing *f* vs. *n*. Generate the reference strings randomly, for the above experiment.

---------- * ----------