 ***DEPARTMENT OF COMPUTER ENGINEERING***

Experiment No.

|  |  |
| --- | --- |
| Semester | S.E. Semester IV – Computer Engineering |
| Subject | Operating System |
| Subject Professor In-charge | Snehal Andhare |
| Assisting Teachers | Ms. Rasika Ransing |
| Laboratory | M310B – Computer Engineering Laboratory |

|  |  |  |
| --- | --- | --- |
| Student Name | B. Niranjana | |
| Roll Number | 18104A0014 | |
| Grade and Subject Teacher’s Signature |  |  |

|  |  |  |
| --- | --- | --- |
| Experiment Number |  | |
| Experiment Title | PBL 1 | |
| Resources / Apparatus Required | Hardware:  Computer | Software:  Compiler |
| Objectives  (Skill Set / Knowledge Tested / Imparted) | Implementation of Banker’s Algorithm for Deadlock handling | |
| Theory: | The Banker algorithm, sometimes referred to as the detection algorithm, is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.  For the Banker's algorithm to work, it needs to know three things:   * How much of each resource each process could possibly request[MAX] * How much of each resource each process is currently holding[ALLOCATED] * How much of each resource the system currently has available[AVAILABLE] | |
| Code: | #include <stdio.h>  int current[5][5], maximum\_claim[5][5], available[5];  int allocation[5] = {0, 0, 0, 0, 0};  int maxres[5], running[5], safe = 0;  int counter = 0, i, j, exec, resources, processes, k = 1;  void main()  {  printf("\nEnter number of processes: ");  scanf("%d", &processes);  for (i = 0; i < processes; i++)  {  running[i] = 1;  counter++;  }  printf("\nEnter number of resources: ");  scanf("%d", &resources);  printf("\nEnter Available Resources:");  for (i = 0; i < resources; i++)  {  scanf("%d", &maxres[i]);  }  printf("\nEnter Allocated Resources:\n");  for (i = 0; i < processes; i++)  {  for(j = 0; j < resources; j++)  {  scanf("%d", &current[i][j]);  }  }  printf("\nEnter Maximum Resources needed:\n");  for (i = 0; i < processes; i++)  {  for(j = 0; j < resources; j++)  {  scanf("%d", &maximum\_claim[i][j]);  }  }  printf("\nThe Available is: ");  for (i = 0; i < resources; i++)  {  printf("\t%d", maxres[i]);  }  printf("\nThe Allocated Resources:\n");  for (i = 0; i < processes; i++)  {  for (j = 0; j < resources; j++)  {  printf("\t%d", current[i][j]);  }  printf("\n");  }  printf("\nThe Maximum Resources needed:\n");  for (i = 0; i < processes; i++)  {  for (j = 0; j < resources; j++)  {  printf("\t%d", maximum\_claim[i][j]);  }  printf("\n");  }  for (i = 0; i < processes; i++)  {  for (j = 0; j < resources; j++)  {  allocation[j] += current[i][j];  }  }  printf("\nAllocated resources:");  for (i = 0; i < resources; i++)  {  printf("\t%d", allocation[i]);  }  for (i = 0; i < resources; i++)  {  available[i] = maxres[i] - allocation[i];  }  printf("\nAvailable resources:");  for (i = 0; i < resources; i++)  {  printf("\t%d", available[i]);  }  printf("\n");  while (counter != 0)  {  safe = 0;  for (i = 0; i < processes; i++)  {  if (running[i])  {  exec = 1;  for (j = 0; j < resources; j++)  {  if (maximum\_claim[i][j] - current[i][j] > available[j])  {  exec = 0;  break;  }  }  if (exec)  {  printf("\nProcess%d is executing\n", i + 1);  running[i] = 0;  counter--;  safe = 1;  for (j = 0; j < resources; j++)  {  available[j] += current[i][j];  }  break;  }  }  }  if (!safe)  {  printf("\nThe processes are in unsafe state.\n");  break;  }  else  {  printf("\nThe process is in safe state");  printf("\nAvailable vector:");  for (i = 0; i < resources; i++)  {  printf("\t%d", available[i]);  }  printf("\n");  }  }  } | |
| Output: |  | |