***Implementing Dining Philosopher’s problem using threads (Single Terminal Solution)***

|  |
| --- |
| **//*Code:***  #include <stdio.h>  #include <pthread.h>  #include <semaphore.h>  #include <unistd.h>  #include <stdlib.h>  #define N 5  sem\_t mutex[N];  void\* thread(void\* arg)  {  int index=\*(int\*)arg;  printf("\nPHILOSPHER %d thinking\n",index+1);  sleep(0.1);  if(index<N-1)  {  sem\_wait(&mutex[index]);  sem\_wait(&mutex[(index+1)%N]);  printf("\nLeft fork %d and Right fork %d acquired by philospeher %d...\n",index, (index+1)%N, index+1 );  //critical section  printf("\n///////////////////////// PHILOSPHER %d EATING ////////////////////\n",index+1);  sleep(2);  //signal  sem\_post(&mutex[index]);  sem\_post(&mutex[(index+1)%N]);  printf("\nPHILOSPHER %d FINISHED EATING...\n",index+1);  }  else //deadlock condition  {  sem\_wait(&mutex[(index+1)%N]);  sem\_wait(&mutex[index]);  printf("\nRight fork %d and Left fork %d acquired by philospher %d..\n",(index+1)%N,index,index+1);  //critical section  printf("\n///////////////////////// PHILOSPHER %d EATING ////////////////////\n",index+1);  sleep(2);  //signal  sem\_post(&mutex[(index+1)%N]);  sem\_post(&mutex[index]);  printf("\nPHILOSPHER %d FINISHED EATING...\n",index+1);  }  }  int main(int argc,char\* argv[])  {  int i=0;  for(i=0;i<5;i++)  sem\_init(&mutex[i], 0, 1);  pthread\_t t[5];  int \*array=(int\*)malloc((N)\*sizeof(int));  int m;  for(m=0;m<N;m++)  {  array[m]=m;  }  int cn;  for (cn=0;cn<5;cn++)  {  pthread\_create(&t[cn],NULL,thread,&(array[cn]));  }  for(i=0;i<5;i++)  {  pthread\_join(t[i],NULL);  }  for(i=0;i<5;i++)  sem\_destroy(&mutex[i]);  return 0;  } |
| //***Output:*** |

***Implementing Dining Philosopher’s Problem using processes (Multiple Terminals Solution)***

|  |
| --- |
| **SERVER CODE:**  #include <unistd.h>  #include <semaphore.h>  #include <sys/shm.h>  #include <sys/ipc.h>  #include <string.h>  #include <stdio.h>  int main()  {  int id\_sem1=shmget(565, 1024, 0666 | IPC\_CREAT | IPC\_EXCL);  int id\_sem2=shmget(566, 1024, 0666 | IPC\_CREAT | IPC\_EXCL);  int id\_sem3=shmget(567, 1024, 0666 | IPC\_CREAT | IPC\_EXCL);  int id\_sem4=shmget(568, 1024, 0666 | IPC\_CREAT | IPC\_EXCL);  int id\_sem5=shmget(569, 1024, 0666 | IPC\_CREAT | IPC\_EXCL);  sem\_t \*s[5];  int arr[]={id\_sem1,id\_sem2,id\_sem3,id\_sem4,id\_sem5};    if (id\_sem1==-1 || id\_sem2==-1 || id\_sem3==-1 || id\_sem4==-1 || id\_sem5==-1 )  {  printf("\nERROR\n");  return 1;  }  int i=0;  for(i=0;i<5;i++)  s[i]= (sem\_t\*) shmat(arr[i], NULL, 0);  for(i=0;i<5;i++)  sem\_init(s[i], 1, 0); //1 means semaphore is used for process synchronization  char a;  printf("\nPress any key to continue:");  scanf("%c",&a);  for(i=0;i<5;i++)  sem\_post(s[i]);  sleep(45);  printf("\nSemaphores up |");    for(i=0;i<5;i++)  {  sem\_destroy(s[i]);  shmdt(s[i]);  }  shmctl(id\_sem1, IPC\_RMID, NULL);  shmctl(id\_sem2, IPC\_RMID, NULL);  shmctl(id\_sem3, IPC\_RMID, NULL);  shmctl(id\_sem4, IPC\_RMID, NULL);  shmctl(id\_sem5, IPC\_RMID, NULL);  }  **CLIENT CODE:**  #include <semaphore.h>  #include <sys/shm.h>  #include <sys/ipc.h>  #include <string.h>  #include <stdio.h>  #include <unistd.h>  #include <fcntl.h>  #define N 5  int main()  {  int id\_sem1=shmget(565, 1024, 0);  int id\_sem2=shmget(566, 1024, 0);  int id\_sem3=shmget(567, 1024, 0);  int id\_sem4=shmget(568, 1024, 0);  int id\_sem5=shmget(569, 1024, 0);  int fd=open("file",O\_RDWR | O\_APPEND );  int arr[]={id\_sem1,id\_sem2,id\_sem3,id\_sem4,id\_sem5};    sem\_t \*s[5];    if (id\_sem1==-1 || id\_sem2==-1 || id\_sem3==-1 || id\_sem4==-1 || id\_sem5==-1 )  {  printf("\nERROR\n");  return 1;  }  int i,process,j;  char buf;  char already\_in[5];  for(i=0;i<5;i++)  s[i]= (sem\_t\*) shmat(arr[i], NULL, 0);    i=0;  while(read(fd,&buf,1)>0)  {  already\_in[i]=buf;  i++;  }  already\_in[i]='\0';  if(strlen(already\_in)>=5)  {  printf("\nAll the five philosphers have eaten food, please reset file\n");  return 1;  }  if( strlen(already\_in)>0 )  {  printf("\nPhilosphers ");  for(i=0;i<strlen(already\_in);i++)  {  printf("%c,",already\_in[i]);  }  printf(" have already eaten food\n");  }  printf("\nEnter the philospher which wants to eat:");  scanf("%d",&process);  for(j=0;j<strlen(already\_in);j++)  if( (already\_in[j]-48) == process)  {  printf("PHILOSPHER %d HAVE ALREADY EATEN FOOD\n",process);  return 1;  }  int x=process-1;  //sem\_init(semaphore, 1, 1); //no need for this in client  if(process<=4) //p1 0 1 //p2 1 2 //p3 2 3 //p4 3 4 //p5 4 0  {  sem\_wait(s[x]); //left //i  sem\_wait(s[(x+1)%N]); //i+1%N  printf("\nPHILOSPHER %d ACQUIRED FORKS %d AND %d",process,x,(x+1)%N);  printf("\nPHILOSPHER %d IS EATING",process);  char varr=48+process;  write(fd,&varr,1);  sleep(5);  sem\_post(s[x]);  sem\_post(s[(x+1)%N]);  printf("\nPHILOSPHER %d FINISHED EATING\n",process);  }  else if (process==5) //deadlock condition  {  sem\_wait(s[(x+1)%N]); //right  sem\_wait(s[x]); //left  printf("\nPHILOSPHER %d ACQUIRED FORKS %d AND %d",process,(x+1)%N,x); //0 4  printf("\nPHILOSPHER %d IS EATING",process);  char varr=48+process;  write(fd,&varr,1);  sleep(5);  sem\_post(s[(x+1)%N]);  sem\_post(s[x]);  printf("\nPHILOSPHER %d FINISHED EATING\n",process);  }  else  {  printf("\nInvalid philospher\n");  return 1;  }  close(fd);  for(i=0;i<5;i++)  shmdt(s[i]);  } |
| //***Output:***  **BEFORE THE EXECUTION OF THE CODE FILE SHOWS :**    **CODE EXECUTED:**    **ORDER IN WHICH THE PROCESSES ATE FOOD IS DETERMINED BY LOOKING AT THIS FILE:**    **AFTER THIS OUTPUT:**    ***A mp4 video is attached with this report which gives a clearer picture of how mutual exclusion takes place when multiple processes want to enter in the critical region simultaneously.*** |