Name: Sagar Indolia

Roll No: B-44

Section: K18MS

Registration No: 11803103

GITHUB link :

|  |
| --- |
| ANS - 1 |
|  |
| #include<stdio.h>    #define n 4    int compltedPhilo = 0,i;    struct fork{  int taken;  }ForkAvil[n];    struct philosp{  int left;  int right;  }Philostatus[n];    void goForDinner(int philID){ //same like threads concept here cases implemented  if(Philostatus[philID].left==10 && Philostatus[philID].right==10)  printf("Philosopher %d completed his dinner\n",philID+1);  //if already completed dinner  else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){  //if just taken two forks  printf("Philosopher %d completed his dinner\n",philID+1);    Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by assigning value 10  int otherFork = philID-1;    if(otherFork== -1)  otherFork=(n-1);    ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks  printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);  compltedPhilo++;  }  else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for right fork  if(philID==(n-1)){  if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION  ForkAvil[philID].taken = Philostatus[philID].right = 1;  printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);  }else{  printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);  }  }else{ //except last philosopher case  int dupphilID = philID;  philID-=1;    if(philID== -1)  philID=(n-1);    if(ForkAvil[philID].taken == 0){  ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;  printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);  }else{  printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);  }  }  }  else if(Philostatus[philID].left==0){ //nothing taken yet  if(philID==(n-1)){  if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION  ForkAvil[philID-1].taken = Philostatus[philID].left = 1;  printf("Fork %d taken by philosopher %d\n",philID,philID+1);  }else{  printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);  }  }else{ //except last philosopher case  if(ForkAvil[philID].taken == 0){  ForkAvil[philID].taken = Philostatus[philID].left = 1;  printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);  }else{  printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);  }  }  }else{}  }    int main(){  for(i=0;i<n;i++)  ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;    while(compltedPhilo<n){  /\* Observe here carefully, while loop will run until all philosophers complete dinner  Actually problem of deadlock occur only thy try to take at same time  This for loop will say that they are trying at same time. And remaining status will print by go for dinner function  \*/  for(i=0;i<n;i++)  goForDinner(i);  printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);  }    return 0;  } |  |
|  |  |
| **OUTPUT** |  |
|  |  |
|  |  |
| ANS -2 : |  |
| |  | | --- | | #include<stdio.h> | |  | #include<pthread.h> | |  | int global[2]; | |  |  | |  | void \*sum\_thread(void \*arg) | |  | { | |  | int \*args\_array; | |  | args\_array = arg; | |  |  | |  | int n1,n2,sum; | |  | n1=args\_array[0]; | |  | n2=args\_array[1]; | |  | sum = n1+n2; | |  |  | |  | printf("N1 + N2 = %d\n",sum); | |  | } | |  |  | |  | int main() | |  | { | |  | printf("First number: "); | |  | scanf("%d",&global[0]); | |  |  | |  | printf("Second number: "); | |  | scanf("%d",&global[1]); | |  | pthread\_t tid\_sum; | |  | pthread\_create(&tid\_sum,NULL,sum\_thread,global); | |  | pthread\_join(tid\_sum,NULL); | |  |  | |  | return 0; | |  | }  **OUTPUT :** | |  |
|  |  |
|  |  |
| ANS- 3: |  |
| |  | | --- | | #include<iostream> | |  | #include<thread> | |  | #include<mutex> | |  | using namespace std; | |  | std::mutex m1; | |  | std::mutex m2; | |  | std::mutex m3; | |  | void thread1() { | |  | m1.lock(); | |  | m2.lock(); | |  | m3.lock(); | |  | cout<<"Critical section of Thread Thread One\n"; | |  | m1.unlock(); | |  | m2.unlock(); | |  | m3.unlock(); | |  | } | |  | void thread2() { | |  | m2.lock(); | |  | m1.lock(); | |  | m3.lock(); | |  | cout<<"Critical section of Thread Thread Two\n"; | |  | m2.unlock(); | |  | m1.unlock(); | |  | m3.unlock(); | |  | } | |  | void thread3() { | |  | m3.lock(); | |  | m1.lock(); | |  | m2.lock(); | |  | cout<<"Critical section of Thread Thread Three\n"; | |  | m3.unlock(); | |  | m1.unlock(); | |  | m2.unlock(); | |  | } | |  | int main() | |  | { | |  | thread t1(thread1); | |  | thread t2(thread2); | |  | thread t3(thread3); | |  | t1.join(); | |  | t2.join(); | |  | t3.join(); | |  | return 0; | |  | } |   **OUTPUT :** |  |
|  |  |
|  |  |
|  |  |
| ANS -4 :   |  | | --- | | #include<unistd.h> | |  | #include<stdio.h> | |  | #include<fcntl.h> | |  |  | |  | int main() | |  | { | |  | int fd, n, p; | |  | char arr[100]; | |  | fd = open("SEEK\_END.txt", O\_CREAT|O\_RDWR, 0777); | |  | n = read(0, arr, 100); | |  | write(fd, arr, n); | |  | p = lseek(fd, -5, SEEK\_END); | |  | read(fd, arr, 5); | |  | write(1, arr, 5); | |  | printf("\n"); | |  | } |   **OUTPUT :** |  |

