**Student Name: Shubham S. Garad**

**Student ID: 11700069**

**Email Address:** [**shubham.11700069@lpu.in**](mailto:shubham.11700069@lpu.in)

**GitHub Link:** [**https://github.com/shubham-1170069/OS**](https://github.com/shubham-1170069/OS)

**Problem:**

1. There are 3 student processes and 1 teacher process. Students are supposed to do their assignments and they need 3 things for that pen, paper and question paper. The teacher has an infinite supply of all the three things. One students has pen,an other has paper and another has question paper. The teacher places two things on a shared table and the student having the third complementary thing makes the assignment and tells the teacher on completion. The teacher then places another two things out of the three and again the student having the third thing makes the assignment and tells the teacher on completion. This cycle continues. WAP to synchronize the teacher and the students.

**Explanation:**

1. In the given solution I have taken all the student processes and resources in 2D array and initialized then to 0.
2. I have made 3 student processes in three different functions which will be executed by single s\_thread and one t\_thread for execution of teacher process.
3. User will get a menu to select any two out of three resources that are to be placed on shared table.
4. If one process is completed there will be a message printed on the screen saying process is completed.
5. When one process is executing no other student or teacher process will execute and for achieving this I have used Mutex lock.
6. When a process starts to execute it acquires the lock and when it completes the execution releases the lock.
7. After completion of all the three processes the program will end.

**Code Snippet:**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<stdbool.h>

int student[3][4]={0};

void \*teacher();

void \*student1();

void \*student2();

void \*student3();

pthread\_mutex\_t lck;

int ch1,ch2;

int r1,r2;

int main()

{

printf("\t\t\t---Welcome---\n");

pthread\_mutex\_init(&lck,NULL);\

student[1][1]=1;

student[2][2]=2;student[3][3]=1;

pthread\_t t\_thread;

pthread\_t s\_thread;

printf("Resources Menu: \n\t\tPress '1' for pen\n\t\tPress '2' for paper \n\t\tPress '3' for question\_paper \n");

while(1)

{

if(student[1][4]==1 && student[2][4]==1 && student[3][4]==1){break;}

pthread\_create(&t\_thread, NULL, teacher, NULL);

pthread\_join(t\_thread,NULL);

if((ch1==1 && ch2==2 || ch2==1 && ch1==2 ) && student[3][4]==0)

{

pthread\_create(&s\_thread, NULL, student3, NULL);

pthread\_join(s\_thread,NULL);

}

else if((ch1==1 && ch2==3 || ch2==1 && ch1==3 ) && student[2][4]==0)

{

pthread\_create(&s\_thread, NULL, student2, NULL);

pthread\_join(s\_thread,NULL);

}

else if((ch1==2 && ch2==3 || ch2==2 && ch1==3 ) && student[1][4]==0)

{

pthread\_create(&s\_thread, NULL, student1, NULL);

pthread\_join(s\_thread,NULL);

}

else

{

printf("\n\tError (007): try again.. with different choices.\n");

}

}

printf("\n\t----Done---\n");

}

void \*teacher()

{

pthread\_mutex\_lock(&lck);

printf("\nFirst Resource on shared tabel:-\t");

scanf("%d",&ch1);

printf("Second Resource on shared tabel:-\t");

scanf("%d",&ch2);

pthread\_mutex\_unlock(&lck);

}

void \*student2()

{

pthread\_mutex\_lock(&lck);

printf("\nChoices Made = 'pen', 'question\_paper'\n");

student[2][4]=1;

printf("\n\tStudent 2 has Completed the assignment. \n");

pthread\_mutex\_unlock(&lck);

}

void \*student3()

{

pthread\_mutex\_lock(&lck);

printf("\nChoices Made = 'pen', 'paper'\n");

student[3][4]=1;

printf("\n\tStudent 3 has Completed the assignment.\n");

pthread\_mutex\_unlock(&lck);

}

void \*student1()

{

pthread\_mutex\_lock(&lck);

printf("\nChoices Made = 'paper', 'question\_paper'\n");

student[1][4]=1;

printf("\n\tStudent 1 has Completed the assignment.\n");

pthread\_mutex\_unlock(&lck);

}

**Student Name: Shubham S. Garad**

**Student ID: 11700069**

**Email Address:** [**shubham.11700069@lpu.in**](mailto:shubham.11700069@lpu.in)

**GitHub Link:** [**https://github.com/shubham-1170069/OS**](https://github.com/shubham-1170069/OS)

**Problem:**

1. Design a scheduling program that is capable of scheduling many processes that comes in at some time interval and are allocated the CPU not more than 10 time units. CPU must schedule processes having short execution time first. CPU is idle for 3 time units and does not entertain any process prior this time. Scheduler must maintain a queue that keeps the order of execution of all the processes. Compute average waiting and turnaround time.

**Explanation:**

1. For solving this problem the processes are scheduled considering their arrival and Wait time.
2. I have made 5 functions: readData(); Init(); getNextProcess(); dispTime(); computeSRT();
3. User will get to enter process burst time and arrival time.
4. Depending on that ComputeSRT function will calculate the shortest run time of the processes and it will be entered in gantt chart.
5. After every unit time when a process will entered by user ComputeSRT will compare the remaining run time of currently running process and if less than new process then it will be preumpted.
6. After completion of all the processes the program will prepare a gantt chart and program will end.

**Code Snippet:**

#include<stdio.h>

int No,bt[10],at[10],tat[10],wt[10],rt[10],finish[10],twt,ttat,total,tq=0;

void readData()

{

printf("Enter no. of processes\n");

scanf("%d",&No);

printf("Enter the arrival times in order:\n");at[0]=0;

printf("\n%d ----(Note:-for 1st process AT is by default '0' so start from of 2nd  process)\n",at[0]);

for(int i=1;i<No;i++)

scanf("%d",&at[i]);

printf("Enter the burst times in order :\n");

for(int i=0;i<No;i++)

scanf("%d",&bt[i]);

}

void Init()

{

total=0;

twt=0;

ttat=0;

for(int i=0; i<No; i++)

{

rt[i]=bt[i];

finish[i]=0;

wt[i]=0;

tat[i]=0;

total+=bt[i];

}

}

int getNextProcess(int time)

{

int i,low;

for(i=0;i<No;i++)

{ if(finish[i]==0) { low=i; break; } }

for(i=0;i<No;i++)

{

if(finish[i]!=1)

{ if(rt[i]<rt[low] && at[i]<=time) { low=i; } }

}

return low;

}

void dispTime()

{

for(int i=0;i<No;i++)

{

twt+=wt[i];

tat[i]=wt[i]+bt[i];

ttat+=tat[i];

printf("waiting time for P%d == %d, Turnaround time= %d \n",i+1,wt[i],tat[i]);

}

printf("Avg Waiting time = %f and Avg Turnaround time = %f\n",(double)twt/No,(double)ttat/No);

printf("\n\t\t\t--Scheduling complete--\n");

}

void computeSRT()

{

readData();

Init();

int time,next=0,old,i,new\_time;

printf("Gantt Chart\n ");

for(time=0;time<total;time++)

{

old=next;

next=getNextProcess(time);

if((old!=next &&tq==0 ) ||time==0)

{ printf("(t=%d)|----p%d----|",time,next+1); }

rt[next]=rt[next]-1;

if(rt[next]==0)

{ finish[next]=1; }

for(i=0;i<No;i++)

{

if(i!=next && finish[i]==0 && at[i]<=time)

{ wt[i]++;}

}

if(time!=0&&old==next&&time%10==0){next=getNextProcess(time);printf("(t=%d)|---- p%d----|",time,next+1);}

}

printf("( %d ) \n",total);

for(i=0;i<No;i++)

{ if(!finish[i])

{

printf("Scheduling failed, cannot continue \n ");

return;

}

dispTime();

}

}

int main()

{

printf("\t<-------------------------Welcome------------------------->\t\n");

printf("....(--please Extend your command prompt to appropriate size for Gantt chart--).....\n");

void readData();

void computeSRT();

void Init();

void dispTime();

int getNextProcess(int);

computeSRT();

printf("\t<-------------------------------Rights reserved to Shubham>\t\n");

}