ROUND ROBIN

```
#include<stdio.h>
#include<conio.h>
void main()
{
   // initlialize the variable name
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Ti
me
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");  // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
```

```
    {
        temp[i] = temp[i] - quant;
        sum = sum + quant;
    }
    if(temp[i]==0 && count==1)
    {
        y--; //decrement the process no.
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-
at[i], sum-at[i]-bt[i]);
        wt = wt+sum-at[i]-bt[i];
        tat = tat+sum-at[i];
        count =0;
    }
    if(i==NOP-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}
```

OVERLAY CONCEPTS

```
#include <unistd.h>
#include <stdio.h>
int main()
{ printf("Transfer to execlp function \n");
execlp(“head”, “head”,”-2”,”f1”,NULL);
printf("This line will not execute \n”);
return 0;
}
```

MUTUAL EXCLUSION

```
#include <sys/sem.h>
#include <sys/ipc.h>
 int main()
{ int pid,semid,val;
struct sembuf sop;
semid=semget((key_t)6,1,IPC_CREAT|0666);
pid=fork();
sop.sem_num=0; sop.sem_op=0; sop.sem_flg=0;
if (pid!=0)
{ sleep(1);
printf("The Parent waits for WAIT signal\n");
semop(semid,&sop,1);
printf("The Parent WAKED UP & doing her job\n");
sleep(10);
printf("Parent Over\n"); }
else
 { printf("The Child sets WAIT signal & doing her job\n");
semctl(semid,0,SETVAL,1);
```

```
sleep(10);

printf("The Child sets WAKE signal & finished her job\n");

semctl(semid,0,SETVAL,0);

printf("Child Over\n"); }

return 0; }
```

READER WRITER PROBLEM

```
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>
sem_t x,y;
pthread_t tid;
pthread_t writerthreads[100],readerthreads[100];
int readercount;

void *reader(void* param)
{
   sem_wait(&x);
   readercount++;
   if(readercount==1)
   sem_wait(&y);
   sem_post(&x);
   printf("\n%d reader is inside",readercount);

   sem_wait(&x);
   readercount--;
   if(readercount==0)
   sem_post(&y);
   sem_post(&x);
   printf("\n%d Reader is leaving",readercount+1);
}

void *writer(void* param)
{
   printf("\nWriter is trying to enter");
   sem_wait(&y);
   printf("\nWriter has entered");
   sem_post(&y);
   printf("\nWriter is leaving");
}

int main()
{
   int n2,i;
   printf("Enter the number of readers:");
   scanf("%d",&n2);
   int n1[n2];
   sem_init(&x,0,1);
   sem_init(&y,0,1);
   for(i=0;i<n2;i++)
```

```
  {
    pthread_create(&writerthreads[i],NULL,reader,NULL);
    pthread_create(&readerthreads[i],NULL,writer,NULL);
  }
  for(i=0;i<n2;i++)
  {
    pthread_join(writerthreads[i],NULL);
    pthread_join(readerthreads[i],NULL);
  }

}
```

DINING PHILOSOPHERS PROBLEM

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>

sem_t room;
sem_t chopstick[5];

void * philosopher(void *);
void eat(int);
int main()
{
int i,a[5];
pthread_t tid[5];

sem_init(&room,0,4);

for(i=0;i<5;i++)
sem_init(&chopstick[i],0,1);

for(i=0;i<5;i++){
a[i]=i;
pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
}
for(i=0;i<5;i++)
pthread_join(tid[i],NULL);
}

void * philosopher(void * num)
{
int phil=*(int *)num;

sem_wait(&room);
printf("\nPhilosopher %d has entered room",phil);
sem_wait(&chopstick[phil]);
sem_wait(&chopstick[(phil+1)%5]);
```

```
eat(phil);
sleep(2);
printf("\nPhilosopher %d has finished eating",phil);

sem_post(&chopstick[(phil+1)%5]);
sem_post(&chopstick[phil]);
sem_post(&room);
}

void eat(int phil)
{
printf("\nPhilosopher %d is eating",phil);
}
```

SUM OF NOS IN PARENT AND CHILD PROCESS

```
#include <iostream>
#include <unistd.h>
using namespace std;

// Driver code
int main()
{
   int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   int sumOdd = 0, sumEven = 0, n, i;
   n = fork();

   // Checking if n is not 0
   if (n > 0) {
      for (i = 0; i < 10; i++) {
         if (a[i] % 2 == 0)
            sumEven = sumEven + a[i];
      }
      cout << "Parent process \n";
      cout << "Sum of even no. is " << sumEven << endl;
   }

   // If n is 0 i.e. we are in child process
   else {
      for (i = 0; i < 10; i++) {
         if (a[i] % 2 != 0)
            sumOdd = sumOdd + a[i];
      }
      cout << "Child process \n";
      cout << "\nSum of odd no. is " << sumOdd << endl;
   }
   return 0;
}
```