Page No.
Date 19 06 24

Rate-Monotonic

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort(int prec[], int b[], int pt[], int n)
{
  int temp=0;
  for(int i=0; i<n; i++)
  {
    for(int j=i; j<n; j++)
    {
      if(pt[j] < pt[i])
      {
        temp = pt[i];
        pt[i] = pt[j];
        pt[j] = temp;
        temp = b[j];
        b[i] = temp;
        temp = prec[i];
        prec[i] = prec[j];
        prec[j] = temp;
      }
    }
  }
}
int gcd(int a, int b)
{
  int r;
  while(b>0)
  {
    r = a%b;
    a = b;
    b = r;
  }
```

```
return a;
}
int lcmul (int p[], int n)
{
 int lcm = p[0];
 for(int i=0; i<n; i++)
  {
    lcm = (lcm*p[i])/gcd(lcm, p[i]);
  }
 return lcm;
}
void main()
{
 int n;
 Printf(" enter the number of processes:");
 scanf(" %d", &n);
 int proc[n], b[n], p[n], rem[n];
 printf("enter the CPU burst time:\n");
 for(int i=0; i<n; i++)
 {
   scanf("%d", &b[i]);
   rem[i] = b[i];
 }
 Printf(" Enter the time periods:\n");
 for(int i=0; i<n; i++);
  scanf("%d", &p[i]);
for(int i=0; i<n; i++)
  proc[i] = i+1;
sort(proc, b, p, n);
Printf("LCM = %d \n", l);
Printf("\n Rate monotonic Scheduling\n");
printf("PID \t Burst \t Period \n");
```

```
for(int i=0; i<n; i++)
  printf("%d\t\t%d\t\t%d\n", proc[i], b[i], pt[i]);
//feasibility
double sum=0.0;
for(int i=0; i<n; i++)
{
  sum += (double)b[i] / pt[i];
}
double rhs = n*(pow(2.0, (1.0/n)) - 1.0);
printf("\n %.1f <= %.1f => %s\n", sum, rhs, (sum<=rhs)
      "true" : "false");
if(sum>rhs)
  exit(0);
printf("Scheduling occures for %d ms\n\n");
//Rms
int time=0, prev=0, x=0;
while(time<t)
{
  int f=0;
  for(int i=0; i<n; i++)
  {
    if(time%pt[i] == 0)
      rem[i] = b[i];
    if(rem[i]>0)
    {
      if(prev != proc[i])
      {
        printf("%dms onwards: Process %d running\n",
              time, proc[i]);
        prev = proc[i];
      }
      rem[i]--;
```

```
t=1;
break;
x=0;
}
}
if(!t)
{
if(x!=1)
{
printf("%d ms sowards; CPU is idle %d\n",time);
x=1;
}
}
timett;
}
}
```

Output: Enter the CPU burst time:

3
2
2

Enter the time periods:

20
5
10

LCM=20

Rate Monotic Scheduling

P[i]	Burst	Period
2	2	5
3	2	10
1	3	20

0.75000 <= 0.779763 ⇒ true

Scheduling occures for 2ms

0ms onwards: process 0 running

2ms onwards: process 3 running

4ms onwards: process 1 running.

5ms onwards: Process 2 running.

7ms onwards: process 1 running.

8 ms onwards: CPU is idle

10 ms Onwards: Process 2 running.

Page No.
Date 19 06 24

Proportional Scheduling

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  srand(NULL));
  int n;
  printf("enter the number of processes:");
  scanf("%d",&n);
  int p[n], t[n], cum[n], m[n]; int c=0; int total=0,
      count=0;
  Printf(" Enter tickets of the processes \n");
  for(int i=0;i<n;i++)
  {
    scanf("%d",&t[i]);
    c+=t[i];
    cu[i]=c;
    p[i]=i+1;
    m[i]=0;
    totalt+=t[i];
  }
  while(count<n)
  {
    int wt=rand()%total;
    for(int i=0;i<n;i++)
    {
      if(int i=0;i<n;i++)
      {
        Printf("the winning number is %d and winning
        participant is :%d\n", wt, p[i]);
```

```
        m[i]=1;
        count++;
      }
    }
  }
  printf("\n Probabilities:\n");
  for(int i=0; i<n; i++)
  {
    printf("The probability of P%d winning: %.2f
      %%\n", p[i], ((double)t[i]/total)*100);
  }
}
```

Output:

Enter number of processes=4
Enter tickets of the processes:
1
2
3
4

The winning number is 6 and winning participant : 4
The winning number is 3 and winning participant : 3
The winning number is 0 and winning participant : 1
The winning number is 0 and winning participant : 2

Probabilities:
The probability of P1 winning : 10.00%
The probability of P2 winning : 20.00
The probability of P3 winning : 30.00
The probability of P4 winning : 40.00%

Page No.
Date 12 06 24

3. Earliest dead line First

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort (int proc[], int d[], int b[], int P[], int n)
{
  int temp=0;
  for(int i=0; i<n; i++)
  {
    for (int j=i; j<n; j++)
    {
      if (d[j] < d[i])
      {
        temp = d[j];
        d[j] = d[i];
        d[i] = temp;
        int temp = P[i];
        P[i] = P[j];
        P[j] = temp;
        temp = b[j];
        b[j] = b[i];
        b[i] = temp;
        temp = proc[i];
        Proc[i] = proc[j];
        Proc[j] = temp;
      }
    }
  }
}
```

4
3
1
2

```
int gcd (int a, int b)
{
  int r;
  while (b>0)
  {
    r = a%b;
    a = b;
    b = r;
  }
  return a;
}
int lcmul (int p[], int n)
{
   int lcm = p[0];
   for (int i=1; i<n; i++)
   {
     lcm = (lcm * p[i]) / gcd (lcm, p[i]);
   }
  return lcm;
void main ()
{
  int n;
  printf (" Enter the number of processes \n");
   scanf ("%d", &n);
   int proc[n], b[n], pt[n], a[n], rem[n];
  Printf (" enter the CPU burst time \n");
  for (i=0; i<n; i++)
  {
   scanf ("%d", &b[i]);
    rem[i] = b[i];
}
```

```
printf(" enter the deadlines \n");
for(i=0; i<n; i++)
    scanf("%d", &a[i]);
printf(" enter the time periods\n");
for(i=0; i<n; i++)
    scanf("%d", &p[i]);
for (int i=0; i<n; i++)
    proc[i] = i+1;
sort(proc, d, b, pt, n);
int l = lcmul(pt, n);
//LCM
// printf("\ earliest deadline Scheduling:\n");
    Printf("P[]\t Burst \t deadline \t Period\n");
    for (int i=0; i<n; i++)
        printf("%d \t %d \t %d \t %d\n", proc[i],
                b[i], d[i], p[i]);
    printf(" Scheduling occures for %d mins\n", l);

int time=0, pre=0, x=0;
int next_Deadlines[n];
for(int i=0; i<n; i++)
{
  next_deadlines[i] = d[i];
   rem[i] = b[i];
}
while (time<l)
{
 for(int i=0; i<n; i++)
  {
    if (time %p[i]==0 && time!=0)
     {
        next_Deadlines[i] = time+d[i];
        rem[i] = b[i];
} }
```

else
{
printf("\nLeave CPU is idle\n");
}
}

Output:

Enter number of processes
3
Enter the CPU burst times
2
2
Enter the deadlines:
4
4
8
Enter the time periods:
20
5
1

Earliest Deadline Scheduling:

PID	Burst	Deadline	Period
1	2	4	20
2	2	4	5
3	2	8	1

Scheduling occurs for 40ms
0-2ms: Task 1 is running.
2-4ms: Task 2 is running
4-5ms: Task 1 is running
5-10ms: Task 2 is running
10-12ms: Task 1
365-40ms: CPU is idle

Page No.
Date 12 04 24

Write a C program to simulate producer consumer problem using semaphores

```
#include<stdio.h>
int mutex=1, full=0, empty=3, x=0;
int main()
{
 int n;
 void producer();
 void consumer();
 int wait(int);
 int signal(int);
printf("\n1. producer \n2. Consumer \n3. Exit");
while(1)
{
    printf("enter your choice:\n");
    scanf("%d",&n);
    switch(n)
    {
    case 1: if((mutex==1)&&(empty!=0))
    producer();
    else
        printf("Buffer is full");
        break;
    Case 2: if(mutex==1&&full!=0)
            consumer();
            else
                printf("Buffer is empty");
            break;
    Case 3: exit(0);
            break;
    }
}
return 0;
```

```
int wait (int s) {
    return (--s);
}
int signal (int s) {
    return (++s);
}
void producer () {
    mutex = wait (mutex);
    full = signal (full);
    empty = wait (empty);
    x++;
    printf ("Producer produces the item %d", x);
    mutex = signal (mutex);
}
void consumer () {
    mutex = wait (mutex);
    full = wait (full);
    empty = signal (empty);
    printf ("Consumer consumes item %d", x);
    x--;
mutex = signal (mutex);
}
```

output:

1. Producer 2. Consumer 3. Exit
Enter your choice: 1
Producer produces item 1
Enter your choice: 1
producer produces item 2
Enter your choice: 1
Producer produces item 3
Buffer is full!!

Page No.
Date

Enter your choice: 2
Consumer consumes Item 3
Enter your choice: 2
Consumer consumes Item 2
Enter your choice: 2
Consumer consumes Item 2
Buffer is empty!!

Page No.
Date 20 06

~~#include<stdio.h>~~

~~#include~~

1. Write a C program to simulate the concept of Dining philosopher problem.

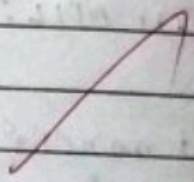
```
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<unistd.h>
#define NUM_PHILOSOPHERS 3
Pthread_mutex_t forks[NUM_PHILOSOPHERS];
pthread_t philosophers[NUM_PHILOSOPHERS];
void *philosopher(void *arg)
{
    int philosopher_id = *(int*)arg;
    int left_fork = philosopher_id;
    int right_fork = (philosopher_id + 1) % NUM_PHILOSOPHERS
While(1){
    //thinking
    printf("philosopher %d is thinking\n", philosopher_id);
    sleep(rand() % 3+1);
    printf("philosopher %d is hungry and trying to pick
            up forks\n", philosopher_id);
    sem_wait(&forks[left_fork]);
    printf("philosopher %d picked up left fork %d.\n",
            philosopher_id, left_fork);
    sem_wait(&forks[right_fork]);
    printf("philosopher %d picked up right fork %d.\n",
            philosopher_id, right_fork);
```

```
printf ("Philosopher %d is eating.\n");
sleep(rand()%2+1);

sem_post(&forks[left_fork]);
sem_post(&forks[right_fork]);
printf("Philosopher %d finished eating and released forks.\n",
        philosopher_id);
}
return NULL;
}

int main() {
  int i;
  int ids[NUM_PHILOSOPHERS];
  for(i=0; i<NUM_PHILOSOPHERS; i++) {
    if(sem_init(&forks[i], 0, 1) != 0) {
      perror("Semaphore initialization failed");
      exit(EXIT_FAILURE);
    }
  }

for(i=0; i<NUM_PHILOSOPHERS; i++) {
  if(pthread_join(philosophers[i], NULL) != 0) {
    perror("Thread join failed");
    exit(EXIT_FAILURE);
  }
}

for(i=0; i<NUM_PHILOSOPHERS; i++) {
  if(sem_destroy(&forks[i]) != 0) {
    perror("Semaphore destruction failed");
    exit(EXIT_FAILURE);
  }
}

return 0;
}
```

Page No.
Date

output:

philosopher 1 is thinking
philosopher 2 is thinking
philosopher 0 is thinking.
philosopher 0 is hungry and trying to pick up forks.
philosopher 0 is hungry and trying to pick up forks.
philosopher 2 picked up left fork 2
philosopher 0 picked up right fork 1



Page No.
Date 20 06 24

(2) Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance

```
#include<stdio.h>
int main()
{
    int a,m,i,j,k;
    n=5; //no. of process
    m=3; // no. of resources.
    int alloc[5][3] ={ {0,1,0}, {2,0,0}, {3,0,2}, {2,1,1},
                       {0,0,2}};  // Allocation matrix.

    int max[5][3]={ {7,5,3}, {3,2,2}, {9,0,2}, {2,2,2},
                    {4,3,3}};
    int avail[3] = {3,3,2}; // Available resources
    int f[n], ans[n], ind=0;
    for(k=0; k<n; k++) {
        f[k]=0;
    }
    int need[n][m];
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y=0;
    for(k=0; k<5; k++) {
        for(i=0; i<n; i++) {
            if(f[i]==0) {
                int flag=0;
                for(j=0; j<m; j++) {
                    if(need[i][j] > avail[j]) {
```

```
                flag=1;
                break;
            }
        }
        if(flag==0){
            ans[ind++]=i;
            for(y=0;y<m;y++)
                avail[y]+=alloc[i][y];
            f[i]=1;
        }
    }
}
int flag=1;
for(int i=0;i<n;i++)
{
    if(f[i]==0)
    {
        flag=0;
        printf("The following system is not safe");
        break;
    }
}
if(flag==1)
{
    printf("Following is the safe sequence\n");
    for(i=0;i<n-1;i++)
        printf("P%d->",ans[i]);
    printf("P%d",ans[n-1]);
}
return(0);
}
```

Page No.
Date

output: Following is the SAFE Sequence
P1 → P3 → P4 → P0 → P2

(3)

Write the C program to simulate deadlock detection

```
#include <stdio.h>
Void main()
{
  int n,m,i,j;
  printf("Enter the number of processes and number
          of types of resources:\n");
  scanf("%d %d",&n,&m);
  int max[n][m],need[n][m],all[n][m],ava[m],flag=1,
      finish[n],dead[n],c=0;
  printf("Entent the maximum number of each type
          of resource needed by each process:\n");
  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
    {
      scanf("%d",&max[i][j]);
    }
  }
  printf("enter the allocated number of each type of
          resources needed by each process:\n");
  for(i=0;i<n;j++)
  {
    scanf("%d",&all[i][j]);
  }
}
```

```
printf("enter the available number of each type of
        resources:\n");
for(j=0;j<m;j++)
{
  for(j=0;j<m;j++)
  {
     need[i][j]=max[i][j]-all[i][j];
  }
}
for(i=0;i<n;i++)
{
  finish[i]=0;
}
while(flag)
{
  flag=0;
  for(i=0;i<n;i++)
  {
    c=0;
   for(j=0;j<m;j++)
   {
     if(finish[i]==0&& need[i][j]<=ava[j])
     {
       c++;
       if(c==m)
       {
         for(j=0;j<m;j++)
         {
           ava[j]+=all[i][j];
           finish[i]=1;
           flag=1;
         }
```

```
if (finish[i] == 1)
{
    i = n;
}
}
}
}
j=0;
flag=0;
for(i=0; i<n; i++)
{
    if(finish[i]==0)
    {
        dead[j]=i;
        j++;
        flag=1;
    }
}
if(flag==1)
{
printf("Deadlock has occured\n");
printf("the deadlock processes are:\n");
for(i=0; i<n; i++)
{
    printf("P%d", dead[i]);
}
}
else
printf("No deadlock has occured!\n");
}
```

Page No.
Date

Output:

Enter the number of process and number of type of resources
5 4
Enter the maximum number of each type of resource needed by each process:
5 1 1 7
3 2 1 1
3 3 2 1
4 0 1 2
0 5 0 1
4 2 1 0
Enter the available allocated number of each type of resource for each process:
3 0 1 4
2 2 1 0
3 1 2 1
0 5 0 1
4 2 1 2
Enter the available number of each type of resources:
0 3 0 1
Deadlock has occured:
The deadlocked process are:
P0 P4

20/6/24