08/05/24.

FCFS:

```
#include <stdio.h>

void swap (int *a, int *b)
{
    int temp = *a ;
    *a = *b ;
    *b = temp ;
}

void sort (int *pid, int *at, int *bt, int n)
{
  for (int i =0 ; i<n ; i++)
  {
    for ( int j=0 ; j<n ; j++)
    {
      if ( at[i] < at [j])
      {
        swap( &at[i], &at[j] ) ;
        swap( &bt[i] , &bt[j]);
        swap( &pid[i] , &pid[j]);
      }
    }
  }
}

int main()
{
  int n ;
  printf(" Enter the number of procss: ");
  scanf("%d",&n);
  int at[n], bt[n], pid[n] ;
```

```
int a, b;
for (int i=0; i<n; i++)
{
printf("Enter the arrival time & burst time:\n");
scanf("%d %d", &a, &b);
at[i] = a;
bt[i] = b;
pid[i] = pid[i+1];
}
sort(pid, at, bt, n);
int ct[n], tat[n], wt[n];
for (int i=0; i<n; i++)
{
if (i==0)
{
ct[i] = at[i] + bt[i];
}
else
{
ct[i] = ct[i-1] + bt[i];
}
tat[i] = ct[i] - at[i];
wt[i] = tat[i] - bt[i];
}

for (int i=0; i<n; i++);
{
printf("pid: %d \t at: %d \t bt: %d \t
ct: %d \t tat: %d \t wt: %d \
\n", i, at[i], bt[i], ct[i], tat[i],
wt[i]);
}
```

Output:

Enter the number of process

Enter the arrival time & burst time.

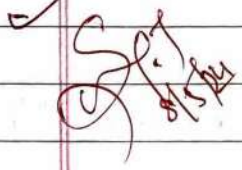
0
5

1
3

2
8

3
6

pid: 0 at: 0 bt: 5 ct: 5 tat: 5 wt: 0
pid: 1 at: 1 bt: 3 ct: 8 tat: 7 wt: 4
pid: 2 at: 2 bt: 8 ct: 16 tat: 14 wt: 6
pid: 3 at: 3 bt: 6 ct: 22 tat: 19 wt: 13



15/05/24

SJF.

```
#include <stdio.h>
void swap(int *a, int *b) {
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;
}

void sort(int *pid, int *at, int *bt, int s, int n){
    for(int i=s; i<n; i++){
        for(int j=s; j<n; j++) {
            if(at[i] < at[j])
            {
                swap(&at[i], &at[j]);
                swap(&bt[i], &bt[j]);
                swap(&pid[i], &pid[j]);
            }
        }
    }
int main(){
    int n;
    printf("Enter the number of processes:");
    scanf("%d", &n);
    int pid[n], at[n], bt[n], ct[n], tat[n],
        wt[n];
    for(int i=0; i<n; i++){
        printf("Enter arrival time and
                burst time:");
        scanf("%d %d", &at[i], &bt[i]);
        pid[i] = i+1;
    }
```

```
sort (pid, at, bt, 0, n);
int c = at[0] + bt[0];
ct[0] = c;
for (int i = 1; i<n ; i++){
    int t = i;
    int x = i;
    while (at[i] < c){
        t++;
        i++;
    }
    sortb (pid, at, bt, x, t);
    if (at[x] > c)
        c = at[x];
    for ( x; x<t ; x++) {
        ct[x] = c + bt[x];
        c = ct[x];
    }
}

for (int i = 0; i<n ; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
}

float avg_tat = 0;
float avg_wt = 0;
for (int i = 0; i<n; i++){
    avg_tat += tat[i];
    avg_wt += wt[i];
}
```

```
for (int i=0; i<n; i++) {
    printf ("%d \t %d \t %d \t %d \t %d \t %d \t
            %d \t", pid[i], at[i], bt[i],
        ct[i], tat[i], wt[i]);
    printf("\n");
}

printf ("Average tat : %f", avg_tat/n);
printf ("Avarag wt : %f", avg_wt/n);
}
```

Output:

Enter the number of processes : 5

Enter arrival time and burst time : 2
1

1
3

4
1

0
6

2
3

4	0	6	6	6	0
1	2	1	7	5	4
3	4	1	8	4	3
5	2	3	11	9	6
2	1	3	14	13	10

Avg-tat = 7.40000
Avg-wt = 4.6000

Round Robin.

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

struct queue {
    int pid;
    struct queue* next;
};

struct queue *rq = NULL;
struct queue *create(int p) {
    struct queue *nn = malloc(sizeof(struct
                                    queue));
    nn->pid = p;
    nn->next = NULL;
    return nn;
}

void enqueue(int p) {
    struct queue *nn = create(p);
    if(rq == NULL)
        rq = nn;
    else {
        struct queue *temp = rq;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = nn;
    }
}
```

```
int dequeue()
{
    int x=0;
    if(rq==NULL)
        return x;
    else{
        struct queue *temp=rq;
        x=temp->pid;
        rq=rq->next;
        free(temp);
    }
    return x;
}

void printq(){
    struct queue *temp=rq;
    while(temp!=NULL){
        printf("%d\t",temp->pid);
        temp=temp->next;
    }
    printf("\n");
}

void swap(int *a, int *b){
    *a=*a+*b;
    *b=*a-*b;
    *a=*a-*b;
}
```

```
void sort( int *pid, int *at, int *bt, int n){
    for(int i=0 ; i<n; i++){
        for(int j=0 ; j<n; j++){
            if (at[i] < at[j]) {
                swap (&at[i], &at[j]);
                swap (&bt[i], &bt[j]);
                swap (&pid[i], &pid[j]);
            }
        }
    }
}

int main(){
    int n, t, x=1;
    printf("Enter the number of processes:");
    scanf("%d", &n);
    printf("Enter the time quantum: ");
    scanf("%d", &t);
    int pid[n], at[n], bt1[n], ct[n], tat[n], wt[n]
        bt2[n], rt[n];
    for(int i=0; i<n; i++){
        printf("Enter arrival time and burst
                time :");
        scanf("%d%d", &at[i], &bt1[i]);
        pid[i] = i+1;
    }

    sort(pid, at, bt1, n);
    enqueue(pid[0]);
    for(int i=0; i<n; i++){
        bt2[i] = bt1[i];
        rt[i] = -1;
    }
```

```
int count = 0;
int ctvar = at[0];
while (count != n) {
        int curp = rq -> pid;
        int curi = 0;
        for (int i = 0; i < n; i++) {
            if (pid[i] == curp) {
                curi = i;
                break;
            }
        }
    if (rt[curi] == -1) {
        rt[curi] = ctvar - at[curi];
    }
    if (rt[curi] <= t) {
        ctvar += bt2[curi];
        bt2[curi] = 0;
    }
    else {
        ctvar += t;
        bt2[curi] -= t;
    }
    while (at[x] <= ctvar && x < n) {
        enqueue (pid[x]);
        x += 1;
    }
    if (bt2[curi] > 0)
        enqueue (pid[curi]);
    if (bt2[curi] == 0) {
        count += 1;
        ct[curi] = ctvar;
    }
dequeue ();
}
```

```
for(int i=0; i<n; i++){
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt1[i];
}

float avg_tat = 0;
float avg_wt = 0;
for (int i=0; i<n; i++) {
        avg_tat += tat[i];
        avg_wt += wt[i];
}

printf("pid \t at \t bt \t ct \t tat \t wt
                \t\t\n");
for(int i=0; i<n; i++){
    printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t
        pid[i], at[i], bt1[i], ct[i], tat[i], wt[i], rt[i]
    printf("\n");
}

printf("\n Average Turn around time : %f",
                        avg_tat/n);
printf("\n Average waiting time : %f", avg_wt/
return 0;
}
```

Output:

pid	at	bt	ct	tat	wt	rt
1	0	5	13	13	8	0
2	1	3	12	11	8	1
3	2	1	5	3	2	2
4	3	2	9	6	4	4
5	4	3	14	10	7	5

Avg_tat = 8.6

Avg_wt = 5.8.

Priority Scheduling.

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

void swap (int *a, int *b){
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;
}

void sort(int *pid, int *at, int *bt, int *prior,
                                    int n){
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
        if(at[i] < at[j]){
            swap(&at[i], &at[j]);
            swap(&bt[i], &bt[j]);
            swap(&pid[i], &pid[j]);
            swap(&prior[i], &prior[j]);
        }
    }
}
}

int highest_priority(int *prior, int s, int e){
    int x = prior[s];
    int j = s;
    for(int i=s; i<e; i++){
        if(prior[i] > x){
            x = prior[i];
            j = i;
        }
    } return j; }.
```

```
int main ()
{
  int n, t, x;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  int pid[n], at[n], bt1[n], ct[n], tat[n], wt[n],
  bt2[n] rt[n], prior[n];
  for (int i=0; i<n; i++) {
  printf("Enter arrival time, burst time and
                priority: ");
  scanf("%d%d%d", &at[i], &bt1[i], &prior[i]);
  pid[i] = i+1;
}

sort(pid, at, bt1, prior, n);
for (int i=0; i<n; i++) {
    bt2[i] = bt1[i];
    rt[i] = -1;
}

int arvc = 0;
int count = 0;
int ctvar = at[0];
int curi = 0;
while (count != n) {
  if (rt[curi] == -1) {
    rt[curi] = ctvar - at[curi];
  }
  if (arvc == n) {
    ctvar += bt2[curi];
    bt2[curi] = 0;
  }
```

```
else{
    ctvar+=1;
    bt2[curi]--=1;
}
for(int i=0; at[i]<=ctvar; i++){
    arvc+=1;
    x=1;
}

if(bt2[curi]==0){
    count+=1;
    ct[curi]=ctvar;
    prior[curi]=-1;
}
curi=highest_priority(prior,0,x+1);
}

for(int i=0; i<n; i++){
    tat[i]=ct[i]-at[i];
    wt[i]=tat[i]-bt1[i];
}

float avg_tat=0;
float avg_wt=0;
for(int i=0; i<n; i++){
    avg_tat+=tat[i];
    avg_wt+=wt[i];
}

printf("pid \t at \t bt \t ct \t tat \t wt \n
\n\n");
```

```
for(int i=0 ; i<n; i++) {
    printf("%d \t %d \t %d \t %d \t %d \t
            %d \t %d \t %d \t", pid[i], at[i], bt1[i],
        ct[i], tat[i], wt[i], rt[i]);
{ printf("\n");
```

```
printf("\n Average Turn around time :%f",
                                avg-tat/n);
printf("\n Average waiting time : %f", avg-wt/n);
return 0;
}.
```

Output :

Pid	priority	AT	BT	CT	TAT	WT	RT
1	10	0	6	12	12	7	0
2	20	1	4	8	7	3	0
3	30	2	2	4	2	0	0
4	40	4	1	5	1	0	0

Sjf

05/06/24

Q. Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories - System processes and user processes. System processes are to be given higher priority than the user processes. Use FCFS scheduling for the processes in each queue.

```
#include <stdio.h>
#include <limits.h>

struct P{
    int id;
    int bt;
    int at;
    int q;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
    int v;
};

void main(){
    int n, ct = 0;
    float awt = 0, atat = 0, art = 0, tp;
    printf("Queue 1 is system process\n
            Queue 2 is user
            process\n");
```

```
printf("Enter the number of processes:");
scanf("%d", &n);
struct P p[n];
struct P temp;
for(int i=0; i<n; i++){
    p[i].id = i+1;
    p[i].v=0;
    printf("Enter Burst Time, Arrival Time
        and Queue of P%d:", i+1);
    scanf("%d %d %d", &p[i].bt, &p[i].at,
            &p[i].q);
}

for(int i=0; i<n; i++){
    for(int j=i+1; j<n; j++){
        if(p[i].at > p[j].at){
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
        }
    if(p[i].at == p[j].at){
        if(p[i].q > p[j].q){
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
        }
    }
}
}
```

```
int ct = p[0].bt, min = INT_MAX, m, count=0;
p[0].v = 1;

while (count < n-1) {
  for (int i=0; i<n; i++) {
    if (p[i].at <= ct && p[i].v == 0) {
      if (p[i].q < min) {
        min = p[i].q;
        m = i;
      }
    }
  }
p[m].v = 1;
ct += p[m].bt;
min = INT_MAX;

temp = p[count + 1];
p[count + 1] = p[m];
p[m] = temp;

count++;
}

printf("\n Process \t Waiting Time \t Turn
      Around Time \t Response Time \n");

for(int i=0; i<n; i++) {
   if (p[i].at < ct) {
      p[i].st = ct;
   }
   else {
      p[i].st = p[i].at;
   }
```

```
p[i].et = p[i].st + p[i].bt ;
ct += p[i].bt ;

p[i].tat = p[i].et - p[i].at ;
p[i].wt = p[i].tat - p[i].bt ;
p[i].rt = p[i].st - p[i].at ;

printf("%d \t \t %d \t \t %d \t \t %d \n",
        p[i].id, p[i].wt, p[i].tat, p[i].rt);
    awt += p[i].wt ;
    atat += p[i].tat ;
    art += p[i].rt ;
}

tp = (float)p[n-1].et/n ;
printf("Average Waiting Time: %2f \n", awt/n);
printf("Average Turn Around Time: %.2f \n", atat/n);
printf("Average Response Time: %.2f \n", art/n);
printf("Throughput : %2f", tp);
}
```

Output :

Queue 1 is system process
Queue 2 is User process.
Enter number of processes : 6

Enter Burst Time, arrival Time and Queue
of P1 : 5 0 1

P2 : 3 2 2

" " P3 : 1 4 1
" " P4 : 2 6 2
" " P5 : 4 8 1
" " P6 : 2 10 2

Process	Waiting Time	Turn Around Time	Response Time
1	0	5	0
2	1	2	1
3	4	7	4
4	1	5	1
5	7	9	7
6	5	7	5

Average Waiting Time : 3.00.
Average Turn Around Time : 5.85
Average Response Time : 3.00.
Throughput : 2.83

Q Write a C program to stimulate Real-time CPU Scheduling algorithms:

a) Rate - Monotonic

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

void sort( int proc[], 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[j] = b[i];
              b[i] = temp;
              temp = proc[i];
              proc[i] = proc[j];
              proc[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 = 1; i<n ; i++)
    {
      lcm = (lcm* p[i]) / gcd (lcm, p[i]);
    }
  return lcm;
}

void main()
{ int n;
  printf(" Enter the number of process:");
  scanf(" %d", &n);
  int proc[n], b[n], pt[n], rem[n];
  printf (Enter the CPU burst times: \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", &pt[i]);
for(int i=0; i<n; i++)
    proc[i] = i+1;

sort(proc, b, pt, n);
int l = lcmul(pt, n);
printf("LCM = %d\n", l);

printf("Rate Monotone 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]);

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 %lf <= %lf => %s \n", sum, rhs,
      (sum<=rhs) ? "true" : "false");
if(sum > rhs)
    exit(0);

printf("Scheduling occurs for %d ms \n\n", l);


int time = 0, prev = 0, x = 0;
while(time < l)
```

```
{
  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]--;
    f=1;
    break;
    x=0;
  }
  }

  if(!f)
  {
    if(x!=1)
      {
        printf("%dms onwards: CPU is idle\n
                                time);
        x=1;
      }
  }
  time++;
  }
}
```

Output:

Enter the number of processes: 3
Enter the CPU burst times:
3
2
2
Enter the time period is
20 5 10
LCM = 20

Rate Monotonic Scheduling:

2	2	5
3	2	10
1	3	20

0.75000 <= 0.779763 => true
Scheduling occurs for 20 ms

0 ms onwards: Process P2 is running.
2 ms " : " 3 " "
4 ms " : " 1 " "
5 ms " : " 2 " "
7 ms " : " 1 " "
9 ms " : CPU is idle
10 ms " : " P2 " "
12 ms " : " 3 " "
14 ms " : CPU is idle.
15 ms " : " P2 " "

(ii) Earliest-deadline first:

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

void sort(int proc[], int d[], int b[], int pt[],
          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;
         temp = pt[i];
         pt[i] = pt[j];
         pt[j] = temp;
         temp = b[j];
         b[j] = b[i];
         b[i] = temp;
         temp = proc[i];
         proc[i] = proc[j];
         proc[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=1; 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], pt[n], d[n], rem[n];
   printf("Enter the CPU burst times: \n");
   for (int i=0; i<n; i++)
     {
        scanf("%d", &b[i]);
        rem[i] = b[i];
     }
```

```
printf("Enter the deadlines: \n");
for (int i=0; i<n; i++)
    scanf("%d", &d[i]);
printf("Enter the time periods: \n");
for (int i=0; i<n; i++) {
    scanf("%d", &pt[i]);
for (int i=0; i<n; i++)
    proc[i] = i+1;

sort (proc, d, b, pt, n);
int l = lcmul(pt, n);

printf("Earliest deadline Scheduling: \n");
printf("PID \t Burst \t Deadline \t Period \n");
for (int i=0; i<n; i++)
  printf("%d \t\t %d \t\t %d \t\t %d \n",
          proc[i], b[i], d[i], pt[i]);

printf("Scheduling occurs for %d ms \n \n", l);

int time = 0; prev=0, x=0;
int nextDeadlines[n];
for (int i=0; i<n; i++)
    {
      nextDeadlines[i] = d[i];
      rem[i] = b[i];
    }

while (time < l)
  {
    for ( int i=0; i<n; i++)
```

```
{
    if (time % pt[i] == 0 && time != 0)
    {
        nextDeadlines[i] = time + d[i];
        rem[i] = b[i];
    }
}
int minDeadline = INT_MAX + 1;
int taskToExecute = -1;
for (int i = 0; i < n; i++)
{
    if (rem[i] > 0 && nextDeadlines[i] < minDeadline)
    {
        minDeadline = nextDeadline[i];
        taskToExecute = i;
    }
}

if (taskToExecute != -1) {
    printf("%d ms : Task %d is running \n",
        time, proc[taskToExecute]);
    rem[taskToExecute]--;
}
else
{
    printf("%d ms : CPU is idle \n", time);
}
time++;
}
```

Output :-

Enter the number of processes : 3.
Enter the CPU burst times :
3
2
2
Enter the deadlines :
7
4
8
Enter the time periods
20
5
10

Earliest deadline first Scheduling :.

PID	Burst	Deadline	Period
2	2	4	5
1	3	7	20
3	2	8	10

Scheduling occurs for 20 ms.

0 ms : Task 2 is running.
1 ms : Task 2 " "
2 ms : Task 1 " "
3 ms : " 1 "
4 ms : " 1 "
5 ms : " 3 "
6 ms : " 3 "
7 ms : " 2 "
8 ms : CPU is idle. 2

9ms : CPU is idle.
10ms : task 2 is running
11ms : " 2 —" —
12ms : " 3 — " —
13 ms : " 3 — " —
14 ms : CPU is idle
15 ms : " 2 —" —
16 ms : " 2 — " —
17 ms : CPU is idle
18 ms —" —
19 ms —" —

Sol.1
5/6/24

07/06/24

Proportional Scheduling :-

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

int main() {
    int n, sOT = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int pid[n];
    int l[n+1];
    l[0] = 0;

    printf("\n Enter the number of tickets for
            each process :\n");
    for (int i = 0; i<n; i++) {
        printf("PID %d : ", i+1);
        scanf("%d", &pid[i]);
        sOT += pid[i];
        l[i+1] = pid[i];
    }
    int t = 1;
    int sum = sOT;
    for (int i = 0; i < n; i++){
        printf("Probability of servicing process %d
                is %d %%\n", i+1, (pid[i] *100)/
                                        sOT
    }
```

```
srand (time (NULL));
while (sum > 0) {
    int x = rand () % 501;
    int j;
    for (j = 0; j < n; j++) {
        if (x < t[j+1]) {
            printf("%d ms: Servicing ticket of
                process %d \n", t, j+1);
    pid[j]--;
    sum--;
    t++;
    break;
    }
  }
}

for (int i = 0; i < n; i++) {
    if (pid[i] == 0) {
        printf("PID%d has finished executing \n",
                                    i+1);
        }
    }
    return 0;
}
```

Output:

Enter the number of processes : 2.

Enter the number of tickets for each process :
PID1 : 10
PID2 : 20

Probability of servicing process 1 is 33%
Probability of servicing process 2 is 66%.

1ms : Servicing ticket of process 2
2ms : " 2
3ms : " 1
4ms : " 1
5ms : " 1
6ms : " 1
7ms : " 2
8ms : " 1
9ms : " 1
10ms : " 1

Producer - Consumer :

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

void producer();
void consumer();
int wait(int);
int signal(int);

int mutex = 1, full = 0, empty = 7, x = 0;

int main() {
    int n;
    printf("\n1. Producer \n2. Consumer
            \n 3. Exit");
while(1) {
    printf("Enter your choice:");
    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:
    printf("\n Number of Items remaining in
            buffer : %d \n", x);
    exit(0);
default:
    printf("Invalid choice \n");
    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("\n Producer produces the item
            %d", x);
    mutex = signal(mutex);
}
```

```
void consumer() {
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf(" In Consumer consumes item %d", x);
    x--;
    mutex = signal(mutex);
}
```

Output:

Menu.
1. Producer.
2. Consumer
3. Exit.

Enter choice: 1.
Buffer value: 1.

Enter choice: 2.
-1 consumed from buffer.

Enter choice: 3
Exit.

Sp.1
12/6/24

19/06/24

Banker's Algorithm :-

```
#include <stdio.h>
#include <stdbool.h>

#define NUM_PROCESSES 5
#define NUM_RESOURCES 3

int available[NUM_RESOURCES];
int maximum[NUM_PROCESSES][NUM_RESOURCES];
int allocation[NUM_PROCESSES][NUM_RESOURCES];
int need[NUM_PROCESSES][NUM_RESOURCES];
int safe_sequence[NUM_PROCESSES];
int scount = 0;

void calculateNeed(){
    for(int i=0; i<NUM_PROCESSES; i++){
    for(int j=0; j<NUM_RESOURCES; j++){
     need[i][j] = maximum[i][j]-allocation[i]
    }
}
}

bool isSafe(){
    int work[NUM_RESOURCES];
    bool finish[NUM_PROCESSES]={false};

    for(int i=0; i<NUM_RESOURCES; i++){
        work[i] = available[i];
    }
```

```
while (true) {
  bool found = false;

for (int i = 0; i < NUM_PROCESSES; i++) {
   if ( ! finish[i]) {
      bool canProceed = true;

      for (int j = 0; j < NUM_RESOURCES; j++) {
         if (need[i][j] > work[j]) {
            canProceed = false;
            break;
      }
}

if (canProceed) {
   for (int j = 0; j < NUM_RESOURCES; j++) {
         work[j] += allocation[i][j];
   }
   printf ("Process %d is visited (%d, %d, %d)\n",
         i, work[0], work[1], work[2]);
finish[i] = true;
safe_sequence[scount] = i;
scount += 1;
found = true;
}
}
}

if ( !found) {
      break;
}
}
```

```
for (int i=0; i<NUM_PROCESSES; i++) {
    if (!finish[i]) {
        return false;
    }
}
return true;
}

int main() {
    int i, j;
    printf("Enter the Available Resources
            Vector : \n");
    for(i=0; i<NUM_RESOURCES; i++){
        scanf("%d", &available[i]);
    }

    printf("Enter the Maximum Matrix: \n");
    for(i=0; i<NUM_PROCESSES; i++) {
        for(j=0; j<NUM_RESOURCES; j++) {
            scanf("%d", &maximum[i][j]);
        }
    }

    printf("Enter the Allocation Matrix: \n");
    for(i=0; i<NUM_PROCESSES; i++) {
        for(j=0; j<NUM_RESOURCES; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }
```

```
calculateNeed();
printf("Need Matrix \n");
for( int i=0; i<NUM_PROCESSES; i++) {
    for (int j=0; j<NUM_RESOURCES; j++) {
        printf("%d \t", need[i][j]);
    }
    printf("\n");
}

if (isSafe()) {
    printf("\n The systems is in a safe
                state! \n");
    printf("\n Safe Sequence \n");
    for (int i=0; i<NUM_PROCESSES; i++)
        printf("%d \t", safe_sequence[i]);
} else {
    printf("\n The system is not in a
                safe state. \n");
}

return 0;
}
```

Output:

```
Enter the Available Resources Vector:
3 3 2
Enter the Maximum Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
```

Enter the Allocation Matrix:

0	1	0
2	0	0
3	0	2
2	1	1
0	0	2

Need Matrix.

7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Process 1 is visited (5, 3, 2)
Process 3 — " — (7, 4, 3)
Process 4 — " — (7, 4, 5)
Process 0 — " — (7, 5, 5)
Process 2 — " — (10, 5, 7)

The system is in a safe state.

Safe Sequence

1 3 4 0 2.

19/06/24.

Dining Algorithm :

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

#define MAX_PHILOSOPHERS 10

typedef enum {THINKING, HUNGRY, EATING} state_t;

state_t states[MAX_PHILOSOPHERS];
int num_philosophers;
int num_hungry;
int hungry_philosophers[MAX_PHILOSOPHERS];
int forks[MAX_philosophers];

void print_state() {
        printf("\n");
    for (int i=0; i<num_philosophers; ++i) {
    if (states[i]== THINKING) printf("P %d is
            thinking \n", i+1);
    else if (states[i]=HUNGRY) printf("P %d is
            waiting \n", i+1);
    else if (states[i]==EATING) printf("P %d is
            eating \n", i+1);
    }
}
```

```
void simulate (int allow_two) {
    int eating_count = 0;
  for (int i = 0; i < num_philosophers; ++i) {
    if (states[i] == HUNGRY) {
      int left_fork = i;
      int right_fork = (i+1) % num_philosophers;

    if (forks[left_fork] == 0 && forks[right_fork] == 0) {
      forks[left_fork] = forks[right_fork] = 1;
      states[i] = EATING;
      eating_count++;
      printf("P %d is granted to eat \n", i+1);
      if (!allow_two && eating_count == 1) break;
      if (allow_two && eating_count == 2) break;

      }
    }
  }

sleep(1);

for (int i = 0; i < num_philosophers; ++i) {
    if (states[i] == EATING) {
      int left_fork = i;
      int right_fork = (i+1) % num_philosophers;
      forks[left_fork] = forks[right_fork] = 0;
      states[i] = THINKING;
    }
  }
}
```

```
int main() {
    printf("Enter the total number of philosophers
            (max %d): ", MAX_philosophers);

    scanf("%d", &num_philosophers);

    if (num_philosophers < 2 || num_philosophers >
                                max_philosophers) {
        printf("Invalid number");
        return 1;
    }

    printf("How many are Hungry : ");
    scanf("%d", &num_hungry);

    for(int i=0; i< num_hungry; ++i) {
        printf("Enter philosopher %d position: ", i+1);
        int position;
        scanf("%d",&position);
        hungry_philosophers[i] = position-1;
        states[hungry_philosophers[i]] = HUNGRY;
    }

    for (int i=0; i< num_philosophers; i++) {
        forks[i] = 0;
    }

    int choice;
    do {
        print_state();
```

```
printf("\n1. One can eat at a time\n");
printf("\n2. Two can eat at a time\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch(choice) {
    case 1:
        stimulate(0);
        break;
    case 2:
        stimulate(1);
        break;
    case 3:
        printf("Exiting.\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
        break;
}
} while(choice != 3);
return 0;
}
```

Output:

Total no. of philosopher : 5
how many are hungry : 3
Enter P1 position : 2
" P2 " : 4
" P3 " : 5

P1 is thinking.
P2 is waiting.
P3 " thinking.
P4 " waiting
P5 " "

1. One can eat at a time
2. two " "
3. Exit

Enter your choice : 1.
P2 is granted to eat.

P1 is thinky.
P2 " "
P3 " "
P4 is waiting.
P5 " "

Choice : 2
P4 is granted to eat.
P1 is thinking.
P2 " "
P3 " "
P4 " "
P5 is waiting.
Choice : 3 exited.

3/7/24

Program - 8.

Deadlock

Write a C program to stimulate deadlock detection.

```
#include <stdio.h>
#include <stdbool.h>

#define NUM_PROCESS 5
#define NUM_PROCESS 3

int available[NUM_RESOURCES];
int allocation[NUM_PROCESSES][NUM_RESOURCES];
int request[NUM_PROCESS][NUM_RESOURCES];
int avail_matrix[NUM_PROCESS+1][NUM_RESOURCES];
bool deadlockDetection(int *safeSequence){
    int work[NUM_RESOURCES];
    bool finish[NUM_PROCESS] = {false};

    for(int i=0; i<NUM_RESOURCES; i++){
        work[i] = available[i];
        avail_matrix[0][i] = work[i];
    }
    int count=0;
    while(count<NUM_PROCESSES){
        bool found = false;

        for(int i=0; i<NUM_PROCESSES; i++){
            if(!finish[i]){
                bool canProceed = true;

                for(int j=0; j<NUM_PROCESSES; j++)
```

```
if (request[i][j] > work[j]) {
        canProceed = false;
        break;
    }
}

if (canProceed) {
for (int j = 0; j < NUM_RESOURCES; j++) {
    work[j] += allocation[i][j];
}
safeSequence[count++] = i;
finish[i] = true;
found = true;
for (int k = 0; k < NUM_RESOURCES; k++) {
availMatrix[count][k] = work[k];
    }
  }
 }
}

if (!found) {
    break;
  }
}

for (int i = 0; i < NUM_PROCESSES; i++) {
    if (!finish[i])
    printf("Deadlock detected. Process P%d is in
        deadlock.\n", i);
    return false;
  }
}
```

```
printf("No deadlock detected. The system is in
        a safe state. \n");
printf(" Safe Sequence : ");
for( int i=0; i< NUM_PROCESSES; i++) {
    printf("P%d ", safeSequence[i]);
}

printf("\n");

return true;
}

int main() {
    int i, j;
    printf("Enter the Available Resources Vector:\n");
    for( i=0; i<NUM_RESOURCES; i++) {
        scanf("%d", &available[i]);
    }

    printf(" Available Resources : ");
    for( i=0; i<NUM_RESOURCES; i++){
        printf("%d", available[i]);
    }
    printf("\n");

    printf(" Enter the Allocation Matrix :\n");
    for( i=0; i< NUM_PROCESS; i++) {
        for( j=0; j< NUM_PROCESS; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }
```

```
printf("Enter the Request Matrix: \n");
for( i=0 ; i< NUM_PROCESSES ; i++){
    for(j=0 ; j<NUM_RESOURCES; j++){
        scanf("%d", &request[i][j]);
    }
}

int safeSequence[NUM_PROCESSES];
if(deadlockDetection(safeSequence)) {
    printf("Available Matrix : \n");
    for( i=0; i<= NUM_PROCESSES; i++){
        for(j=0; j< NUM_PROCESSES; j++){
            printf("%d", avail_matrix[i][j]);
        }
        printf("\n");
    }
}
return 0;
}
```

Output:

```
Enter Available Resources Vector:
0 0 0
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
```

Enter the Request Matrix:

0 0 0
2 0 2
0 0 0
1 0 0
0 0 2

No deadlock detected. The system is in safe state

Safe sequence: P0 P2 P3 P4 P1

Available Matrix:

0 0 0
0 1 0
3 1 3
5 2 4
5 2 6
7 2 6

Memory allocation.

```
#include <stdio.h>
#define MAX 25

void firstfit (int nb, int nf, int b[], int f[]){
  int frag[MAX], bf[MAX] = {0}, ff[MAX] = {0};
  int i, j, temp;

for (i=1; i<=nf; i++) {
   for (j=1; j<=nb; j++) {
      if (bf[j] != 1) {
         temp = b[j] - f[i];
         if (temp >= 0) {
            ff[i] = j;
         frag[i] = temp;
         bf[j] = 1;
         break;
      }
   }
  }
}

printf("\n Memory Management Scheme -
            first fit \n");
printf("File no: \t file_size : \t Block_no :
        \t Block_size : \t Fragement \n");
for (i=1; i<=nf; i++){
   printf("%d \t \t %d \t \t", i, f[i]);
   if (ff[i] != 0) {
      printf("%d \t \t %d \t \t %d \n",
      ff[i], b[ff[i]], frag[i]);
   }else{
      printf("Not Allocated \n");
   }
}
```

```
void bestFit (int nb, int nf, int b[], int
                                    f[]) {
  int frag[MAX], bf[MAX]= {0}, ff[MAX]= {0};
  int i, j, temp, lowest = 10000;

  for (i=1; i<=nf; i++) {
    for (j=1; j<=nb; j++) {
      if (bf[j]!=1) {
        temp = b[j] - f[i];
        if (temp >=0 && lowest >temp) {
          ff[i] = j;
          lowest = temp;
        }
      }
    }
    frag[i] = lowest;
    bf[ff[i]] = 1;
    lowest = 10000;
  }

printf("\n Memory Management Scheme -
                        Best Fit \n");
printf(" File No\t FileSize \t Block No \t
        Block Size\t Fragment \n");
for( i=1; i<=nf; i++){
    printf("%d \t\t %d \t\t",i,f[i]);
    if (ff[i] != 0) {
      printf("%d \t\t %d \t\t %d \n",
        ff[i], b[ff[i]], frag[i]);
    }else {
      printf("Not Allocated \n");
    }
  }
}
```

```
void worstfit(int nb, int nf, int b[], int f[]) {
    int frag[MAX], bf[MAX] = {0}, ff[MAX] = {0};
    int i, j, temp, highest = 0;

for(i=1; i<=nf; i++) {
    for(j=1; j<=nb; j++) {
        if(bf[j]!=1) {
            temp = b[j] - f[i];
            if(temp>=0 && highest<temp) {
                ff[i]=j;
                highest = temp;
            }
        }
    }

frag[i] = highest;
bf[ff[i]] = 1;
highest = 0;
}

printf("\n Memory Management Scheme -
        Worst Fit \n");
printf("File_no:\t File_Size:\t Block_no:
        \t Block_size:\t Fragement \n");
for(i=1; i<=nf; i++) {
    printf("%d \t\t %d \t\t", i, f[i]);
    if(ff[i] != 0) {
        printf("%d \t\t %d \t\t %d \n",
            ff[i], b[ff[i]], frag[i]);
    }else{
        printf("Not Allocated \n");
    }
}
}
```

```
int main(){
    int b[MAX], f[MAX], nb, nf;
    printf("\n Enter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\n Enter the size of blocks:-\n");
    for (int i=1; i<=nb; i++){
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files:-\n");
    for (int i=1; i<=nf; i++){
        printf("File %d:", i);
        scanf("%d", &f[i]);
    }

    int b1[MAX], b2[MAX], b3[MAX];
    for (int i=1; i<=nb; i++){
        b1[i]=b[i];
        b2[i]=b[i];
        b3[i]=b[i];
    }

    firstfit(nb, nf, b1, f);
    bestfit(nb, nf, b2, f);
    worstfit(nb, nf, b3, f);

    return 0;
}
```

Output:

Memory Management Schemes

Enter the number of block : 3
Enter the number of files : 2

Enter the size of the blocks:
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:
File 1: 10
File 2: 4

Memory Management Scheme - First Fit

File_no	File_size	Block_no	Block_size	Fragment
1	1	1	5	4
2	4	3	7	3

Memory Management Scheme - Best Fit

File_no	File_size	Block_no	Block_size	Fragment
1	1	2	2	1
2	4	1	5	1

Memory Management Scheme - Worst fit.

File_no	File_size	Block_no	Block_size	Fragment
1	1	3	7	6
2	4	1	5	1

3/12/24

10/7/24

Page Replacement Algorithm

```
#include <stdio.h>

int isPagePresent(int frames[], int n, int page){
    for(int i=0; i<n; i++){
        if(frames[i] == page){
            return 1;
        }
    }
    return 0;
}

void printFrames(int frames[], int n)
{
    for(int i=0; i<n; i++){
        if(frames[i] != -1){
            printf("%d", frames[i]);
        }else{
            printf("-");
        }
    }
    printf("\n");
}

void fifoPageReplacement(int pages[], int
        numPages, int numFrames){
    int frames[numFrames];
    int front = 0, pageFaults = 0;
```

```
for (int i = 0; i < numFrames; i++) {
    frames[i] = -1;
}

printf("FIFO Replacement \n");
printf("Reference String \t Frames \n");
for (int i = 0; i < numPages; i++) {
    printf("%d \t \t", pages[i]);

    if (!isPageFresent(frames, numFrames, pages[i])) {
        frames[front] = pages[i];
        front = (front + 1) % numFrames;
        pageFaults++;
    }
    printFrames(frames, numFrames);
}

printf("\n Total Page Faults: %d \n \n", pageFaults);
}

int findOptimalReplacementIndex(int pages[],
    int numPages, int frames[], int numFrames,
    int currentIndex) {
    int farthest = currentIndex;
    int index = -1;

    for (int i = 0; i < numFrames; i++) {
        int j;
        for (j = currentIndex; j < numPages; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    index = i;
                }
```

```
            break;
        }
    }
    if (j == numPages) {
        return i;
    }
}

return (index == -1) ? 0 : index;
}

void optPageReplacement(int pages[], int
        numPages, int numFrames) {
    int frames[numFrames];
    int pageFaults = 0;

    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
    }

printf("Optimal Replacement \n");
printf("Reference String \t Frames \n");
for (int i = 0; i < numPages; i++) {
    printf("%d \t \t", pages[i]);

    if (!isPagePresent(frames, numFrames, pages[i])) {
        if (isPagePresent(frames, numFrames, -1)) {
            for (int j = 0; j < numFrames; j++) {
                if (frame[j] == -1) {
                    frames[j] = pages[i];
                    break;
                }
```

```
    }
    else {
        int index = findOptimalReplacementIndex(
                pages, numPages, frames, numFrames,
                                        i+1);
        frames[index] = pages[i];
    }
    pageFaults++;
    printFrames(frames, numFrames);
}
    printf("\n Total Page Faults : %d \n\n",
                pageFaults);
}

void lruPageReplacement(int pages[], int
    numPages, int numFrames){

    int frames[numFrames];
    int pageFaults = 0;
    int timestamps[numFrames];

    for(int i = 0; i < numFrames; i++){
        frames[i] = -1;
        timestamps[i] = -1;
    }

    printf("LRU replacement \n");
    printf("Reference String \t Frames \n");
```

```
for (int i=0; i< numPages; i++) {
    printf("%d \t\t", pages[i]);

if (!isPagePresent(frames, numFrames, pages[i])) {
    int lruIndex = 0;
    for (int j = 1; j< numFrames; j++) {
        if (timestamps[j] < timestamps[lruIndex]) {
            lruIndex = j;
        }
    }
    frames[lruIndex] = pages[i];
    timestamps[lruIndex] = i;
    pageFaults++;
} else {
    for (int j=0; j< numFrames; j++) {
        if (frames[j] == pages[i]) {
            timestamps[j] = i;
            break;
        }
    }
}
printFrames(frames, numFrames);
}
printf("\n Total Page Faults: %d \n\n",
    pageFaults);
}

int main() {
    int numFrames, numPages;
    printf("Enter the number of frames:");
    scanf("%d", &numFrames);
```

```
printf("Enter the number of pages: ");
scanf("%d", &numPages);

int pages[numPages];

printf("Enter the reference string: ");
for (int i=0; i<numPages; i++){
        scanf("%d", &pages[i]);
    }

fifoPageReplacement(pages, numPages, numFrames);
optPageReplacement(pages, numPages, numFrames);
lruPageReplacement(pages, numpages, numFrames);

return 0;
}
```

Output :-

Enter the number of frames = 3
Enter the number of Pages = 20
Enter the reference string :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Fifo Replacement

Reference String	Frames
7	7 - -
0	7 0 -
1	7 0 1
2	2 0 1
0	2 0 1

3	2 3 1
0	2 3 0
4	4 3 0
2	4 2 0
3	4 2 3
0	0 2 3
3	0 2 3
2	0 2 3
1	0 1 3
2	0 1 2
0	0 1 2
1	0 1 2
7	7 1 2
0	7 0 2
1	7 0 1

Total Page Faults : 15.

Optimal Replacement

Reference String	Frames
7	7 - -
0	7 0 -
1	7 0 1
2	2 0 1
0	2 0 1
3	2 0 3
0	2 0 3
4	2 4 3
2	2 4 3
3	2 4 3
0	2 0 3

2	203
2	203
1	201
2	201
0	201
1	201
7	701
0	701
1	701.

Total Page fault : 9.

LRU replacement ~~frames~~.

Reference String	Frames.
7	7 - -
0	7 0 -
1	7 0 1
2	2 0 1
0	2 0 1
3	2 0 3
0	2 0 3
4	4 0 3
2	4 0 2
3	4 3 2
0	0 3 2
3	0 3 2
2	0 3 2
1	1 3 2
2	1 3 2
0	1 0 2
1	1 0 2
7	1 0 7
0	1 0 7
1	1 0 7

Total Page Faults : 12