```
#include<stdio.h>

int main()
{
   int m,n;
   printf("Give the number of processes present:");
   scanf("%d",&m);
   printf("how many types of resources are there:");
   scanf("%d",&n);

   int allocation[m][n];
   int need[m][n];
   int available[n];
   int Rneed[m][n];
   int max[n];
   int safe[m];

   //give the total  resources   the system have
   printf("give the total  resources the system have\n");
   for(int i=0;i<n;i++)
   {
      printf("Number of resources of type %d:",i+1);
      scanf("%d",&max[i]);
   }

   //allocating resources
   for(int i=0;i<m;i++)
   {
      for(int j=0;j<n;j++)
      {
         printf("how much resources of type %d is allocated to the process %d:",j+1,i+1);
```

```
        scanf("%d",&allocation[i][j]);
    }
}

//finding total available resources after allocating resources
int temp[n];
for(int i=0;i<n;i++)
{
    temp[i]=0;
}

for(int i=0;i<n;i++)
{
    for(int j=0;j<m;j++)
    {
        temp[i]+=allocation[j][i];
    }
    available[i]=max[i]-temp[i];
}

//finding the needed resources
for(int i=0;i<m;i++)
{
    for(int j=0;j<n;j++)
    {
        printf("how much resources of type %d is needed to the process %d for execution:",j+1,i+1);
        scanf("%d",&need[i][j]);
    }
}

//finding the remaining need
```

```
for(int i=0;i<m;i++)
{
   for(int j=0;j<n;j++)
   {
      Rneed[i][j]=need[i][j]-allocation[i][j];
   }
}

int count = 0;
int counter = 0;
int finish[m];
int z=0;
for (int i = 0; i < m; i++) {
   finish[i] = 0; // Initialize all processes as not finished
}

// Using Banker's Algorithm
for (int k = 0; k < m; k++) {
   for (int i = 0; i < m; i++) {
      if (finish[i] == 0) { // If process i is not finished
         count = 0; // Reset count for each process
         for (int j = 0; j < n; j++) {
            if (available[j] >= Rneed[i][j]) {
               count++;
            }
         }
         if (count == n) {
            for (int j = 0; j < n; j++) {
               available[j] += allocation[i][j]; // Release the resources
            }
            finish[i] = 1; // Mark process as finished
```

```
                safe[z] = i;
                z++;

                counter++;
            }
          }
       }
    }

    if(counter==m)
    {
        printf("No Deadlock exist");
        printf("Safe sequence is:\n");
       for (int i = 0; i < m; i++)
       {
           printf("%d-",safe[i]);
       }

    }
    else{
        printf("Deadlock exist");
    }
    return 0;
}
```

Bankers

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

int main()
```

```
{
   int n,m;
   printf("Give the number of processes:");
   scanf("%d",&n);
   printf("Give the number of resources:");
   scanf("%d",&m);

   int allocation[n][m],request[n][m],available[m], work[m],finish[n],count,check=0;

   for (int i = 0; i < n; i++)
   {
      finish[i] = 0;
   }
   for (int i = 0; i < m; i++)
   {
      work[i] = available[i];
   }

   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < m; j++)
      {
         printf("Give the resource allocation for process %d:",i+1);
         scanf("%d",&allocation[i][j]);
      }
      for (int j = 0; j < m; j++)
      {
         printf("Give the request of resources for process %d:",i+1);
         scanf("%d",&request[i][j]);
      }
      finish[i] = 0;
```

```
}

for (int j = 0; j < m; j++)
{
   printf("Give the available resources of type %d:",j+1);
   scanf("%d",&available[j]);
   work[j] = available[j];
}


for (int i = 0; i < n; i++)
{
   for (int j = 0; j < n; j++)
   {
      count = 0;
      if(finish[j] == 0)
      {

         for (int k = 0; k < m; k++)
         {
            if(work[k] >= request[j][k])
            {
               count++;
            }
         }
         if(count == m)
         {
            check++;
            finish[j] = 1;
            printf("Process %d is allocated with the resources:\n",j+1);
            printf("Process %d is executing:\n",j+1);
```

```
                printf("Process %d executed sucessfully:\n",j+1);
                printf("Process %d is releasing its resources:\n",j+1);
                for (int l = 0; l < m; l++)
                {

                   work[l] = work[l] + allocation[j][l];
                   printf("available resources of type %d are %d\n",l+1,work[l]);
                }
                printf("----------------------------------------------------------------------");

             }

          }
       }
       if(check == n)
       {
          printf("The system is not in deadlock all processes executed sucessfully:\n");
          exit(0);
       }

    }
    printf("system is in deadlock:\n");



    return 0;
}
```

Deadlock

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

```
struct node {
   int arr[10];
   int f, r;
};

void enqueue(struct node* q, int val) {
   if (q->r == 9) {
      printf("Queue is full\n");
      return;
   }
   if (q->f == -1) q->f = 0;
   q->arr[++q->r] = val;
}

int dequeue(struct node* q) {
   if (q->f == -1 || q->f > q->r) {
      return -1;
   }
   return q->arr[q->f++];
}

int main() {
   struct node q;



   q.f = q.r = -1;

   int n, count = 0;
   printf("Give the number of processes: ");
```

```
scanf("%d", &n);

int AT[n], BT[n], CT[n], TAT[n], WT[n],finished[n],running[n];
for (int i = 0; i < n; i++) {
   printf("Give the Arrival time of process %d: ", i + 1);
   scanf("%d", &AT[i]);
   printf("Give the burst time of process %d: ", i + 1);
   scanf("%d", &BT[i]);
   finished[i] = 0;
   count += BT[i];
}

int t = 0, a, i = 0;

while(1)
{
   for (int j = 0; j < n; j++) {
      if (t >= AT[j] && finished[j] !=1) {
         enqueue(&q, j);
         finished[j] =1;
      }

   }
   t++;
   if(t == count)
   {
      break;
   }
}

t=0;
```

```
while (1) {



    a = dequeue(&q);
    if (a != -1) {
        for (int counter = 0; counter < BT[a]; counter++) {
            t++;
        }
        printf("Process %d is completed at %d\n", a, t);
        CT[i] = t;
        TAT[a] = t - AT[a];
        WT[a] = TAT[a] - BT[a];
        running[i++] = a+1;
        finished[a] = 1;
    } else {
        t++;
    }

    if (count == t) {
        break;
    }
}
printf(" ");
for (int i = 0; i < n; i++)
{
    printf("-----");
}

printf("\n|");

for (int i = 0; i < n; i++)
```

```
    {
        printf(" P%d |",running[i]);
    }

    printf("\n");
    printf(" ");
    for (int i = 0; i < n; i++)
    {
        printf("-----");
    }
    printf("\n0");
    for (int i = 0; i < n; i++)
    {
        printf("    %d",CT[i]);
    }
    return 0;
}
```

fcfs cpu scheduling

[22-11-2024 07:52] Soham: #include <stdio.h>

```
#include <stdlib.h>

struct node {
   int arr[30];
   int f, r;
};

void enqueue(struct node* q, int val) {
   if (q->f == -1) q->f = 0;
   q->arr[++q->r] = val;
}

int dequeue(struct node* q) {
   if (q->f == -1 || q->f > q->r) {
      return -1;
   }
   return q->arr[q->f++];
}

int main() {
   struct node q;
   q.f = q.r = -1;

   int n, totalBurstTime = 0;
   printf("Enter the number of processes: ");
   scanf("%d", &n);

   int AT[n], BT[n], CT[n], TAT[n], WT[n];
   int RBT[n], finished[n];
```

```
// Input Arrival and Burst Times
for (int i = 0; i < n; i++) {
    printf("Enter the Arrival Time of process %d: ", i + 1);
    scanf("%d", &AT[i]);
    printf("Enter the Burst Time of process %d: ", i + 1);
    scanf("%d", &BT[i]);

    RBT[i] = BT[i];  // Initialize Remaining Burst Time
    finished[i] = 0; // Process not finished
    totalBurstTime += BT[i];
}

int t = 0, minProcessIndex;
int totalTimeExecuted = 0; // Tracks total time executed for all processes

while (totalTimeExecuted < totalBurstTime) {
    int minBurstTime = 999;

    // Find the process with the smallest remaining burst time
    for (int j = 0; j < n; j++) {
        if (!finished[j] && AT[j] <= t && RBT[j] < minBurstTime) {
            minBurstTime = RBT[j];
            minProcessIndex = j;
        }
    }

    // If a process is ready to execute
    if (minBurstTime != 999) {
        enqueue(&q, minProcessIndex);
        RBT[minProcessIndex]--; // Execute one unit of burst time
        totalTimeExecuted++;
```

```
        t++;

        // If process is finished, set completion time
        if (RBT[minProcessIndex] == 0) {
            finished[minProcessIndex] = 1;
            CT[minProcessIndex] = t;
        }
    } else {
        t++; // No process was ready, increment time
    }
}

// Calculate Turnaround Time and Waiting Time
for (int i = 0; i < n; i++) {
    TAT[i] = CT[i] - AT[i];
    WT[i] = TAT[i] - BT[i]; // Calculate WT from TAT and original BT
}

// Print Burst Time, TAT, and WT for each process for debugging
printf("\nProcess | Burst Time | TAT | WT\n");
printf("--------------------------------\n");
for (int i = 0; i < n; i++) {
    printf(" P%d     | %d         | %d   | %d\n", i + 1, BT[i], TAT[i], WT[i]);
}

// Calculate and print average TAT and WT
float totalTAT = 0, totalWT = 0;
for (int i = 0; i < n; i++) {
    totalTAT += TAT[i];
    totalWT += WT[i];
}
```

```
    printf("\nAverage Turnaround Time (TAT): %.2f", totalTAT / n);
    printf("\nAverage Waiting Time (WT): %.2f\n", totalWT / n);

    return 0;
}
[22-11-2024 07:52] Soham: SRTF cpu scheduling
[22-11-2024 07:52] Soham: #include <stdio.h>
#include <stdlib.h>

struct node {
    int arr[10];
    int f, r;
};

void enqueue(struct node* q, int val) {
    if (q->r == 9) {
        printf("Queue is full\n");
        return;
    }
    if (q->f == -1) q->f = 0;
    q->arr[++q->r] = val;
}

int dequeue(struct node* q) {
    if (q->f == -1 || q->f > q->r) {
        return -1;
    }
    return q->arr[q->f++];
}
```

```
int main() {
    struct node q;
    q.f = q.r = -1;

    int n, count = 0;
    printf("Give the number of processes: ");
    scanf("%d", &n);

    int AT[n], BT[n], CT[n], TAT[n], WT[n],finished[n],running[n];
    for (int i = 0; i < n; i++) {
        printf("Give the Arrival time of process %d: ", i + 1);
        scanf("%d", &AT[i]);
        printf("Give the burst time of process %d: ", i + 1);
        scanf("%d", &BT[i]);
        finished[i] = 0;
        count += BT[i];
    }

    int t = 0, a,c, i = 0,min ;

    for(int k =0;k<n;k++)
    {
        min = 999;
        for (int j = 0; j < n; j++) {
            if (min > BT[j] && finished[j] !=1 && t>= AT[j]) {
                min = BT[j];
                c=j;
            }

        }
            t += BT[c];
```

```
            enqueue(&q, c);
            finished[c] =1;
    if(t == count)
    {
        break;
    }
}

t=0;
while (1) {


    a = dequeue(&q);
    if (a != -1) {
        for (int counter = 0; counter < BT[a]; counter++) {
            t++;
        }
        printf("Process %d is completed at %d\n", a, t);
        CT[i] = t;
        TAT[a] = t - AT[a];
        WT[a] = TAT[a] - BT[a];
        running[i++] = a+1;
        finished[a] = 1;
    } else {
        t++;
    }

    if (count == t) {
        break;
    }
}
```

```
    printf(" ");
    for (int i = 0; i < n; i++)
    {
        printf("-----");
    }

    printf("\n|");

    for (int i = 0; i < n; i++)
    {
        printf(" P%d |",running[i]);
    }

    printf("\n");
    printf(" ");
    for (int i = 0; i < n; i++)
    {
        printf("-----");
    }
    printf("\n0");
    for (int i = 0; i < n; i++)
    {
        printf("    %d",CT[i]);
    }
    return 0;
}
```

[22-11-2024 07:52] Soham: sjf cpu

[22-11-2024 07:53] Soham: #include<stdio.h>

```
struct node {
    int arr[50];
```

```
    int f, r;
};

void enqueue(struct node* q, int val) {
    if (q->f == -1) q->f = 0;
    q->arr[++q->r] = val;
}

int dequeue(struct node* q) {
    if (q->f == -1 || q->f > q->r) {
        return -1;
    }
    return q->arr[q->f++];
}

int main() {
    struct node q;
    q.f = q.r = -1;

    int n,quantum;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    int AT[n], BT[n], CT[n], TAT[n], WT[n],RCT[100];
    int RBT[n], finished[n],running[100];

    for (int i = 0; i < n; i++) {
        printf("Enter the Arrival Time of process %d: ", i + 1);
        scanf("%d", &AT[i]);
        printf("Enter the Burst Time of process %d: ", i + 1);
        scanf("%d", &BT[i]);
```

```
    RBT[i] = BT[i];
    finished[i] = 0;
}

printf("Give the time quantum:");
scanf("%d",&quantum);
int t=0,element,condition =1,a =0;
while(condition)
{
    for (int i = 0; i < n; i++)
    {
        if(AT[i] == t && finished[i] == 0)
        {
            enqueue(&q,i);
            condition = 0;
            break;
        }
    }t++;

}
t--;
while(1)
{




    element = dequeue(&q);
    if(element != -1)
    {
        running[a] = element;
        for (int k = 0; k < quantum; k++)
```

```
{


   RBT[element]--;
   t++;
   for (int i = 0; i < n; i++)
   {
      if(AT[i] == t && finished[i] == 0)
      {
         enqueue(&q,i);
      }
   }


   if(RBT[element] == 0)
   {
      break;
   }
}
if(RBT[element] == 0)
   {
      finished[element] = 1;
      CT[element] = t;
   }
   else
   {
      enqueue(&q,element);
   }




RCT[a++] = t;
```

```
        printf("Time: %d, Process: P%d, Remaining Burst: %d\n", t, element, RBT[element]);




    }
    else
    {
        break;
    }

}

for (int i = 0; i < n; i++)
{
    TAT[i] = CT[i] - AT[i];
    WT[i] = TAT[i] - BT[i];
}



printf("process | AT | BT | TAT | WT |\n");
for (int i = 0; i < n; i++)
{
    printf("%d       | %d  | %d  | %d   | %d  |\n",i,AT[i],BT[i],TAT[i],WT[i]);
}

 printf(" ");
for (int i = 0; i < a; i++)
{
    printf("-----");
}
```

```
printf("\n|");

for (int i = 0; i < a; i++)
{
   printf(" P%d |",running[i]);
}

printf("\n");
printf(" ");
for (int i = 0; i < a; i++)
{
   printf("-----");
}
printf("\n0");
for (int i = 0; i < a; i++)
{
   if(RCT[i]>9)
   {
      printf("   %d",RCT[i]);
   }
   else
   {
      printf("    %d",RCT[i]);
   }
}

printf("\n");
printf("\n");
printf("\n");
 int TT=0,W=0;
for (int i = 0; i < n; i++)
```

```
    {
        TT += TAT[i];
        printf("\n%d",TAT[i]);
    }
    float avgTAT =(float) TT/n;
    printf("\nAVERAGE TAT is:%f\n",avgTAT);
    for (int i = 0; i < n; i++)
    {
        printf("\n%d",WT[i]);
        W += WT[i];
    }
    float avgWT =(float) W/n;
    printf(" AVERAGE WT is:%f\n",avgWT);




    return 0;
}
[22-11-2024 07:53] Soham: round robin
[22-11-2024 07:53] Soham: #include<stdio.h>

struct process {
    int WT, AT, BT, TAT, PT;
};

int main() {
    struct process a[10];
    int n, temp[10], t = 0, count = 0, short_p;
    float total_WT = 0, total_TAT = 0, Avg_WT, Avg_TAT;

    printf("Enter the number of processes: ");
```

```
scanf("%d", &n);

printf("Enter the arrival time, burst time, and priority of the processes (AT BT PT):\n");
for(int i = 0; i < n; i++) {
   printf("Enter the arrival time of the process %d:\n",i+1);
   scanf("%d", &a[i].AT);
   printf("Enter the brust time of the process %d:\n",i+1);
   scanf("%d", &a[i].BT);
   printf("Enter the priority of the process %d:\n",i+1);
   scanf("%d", &a[i].PT);
   temp[i] = a[i].BT;  // Copying burst time for later use
}

a[9].PT = 10000;  // High priority value to ensure correct selection in the loop

while(count != n) {
   short_p = 9;
   for(int i = 0; i < n; i++) {
      if(a[i].PT < a[short_p].PT && a[i].AT <= t && a[i].BT > 0) {
         short_p = i;
      }
   }

   a[short_p].BT--;
   t++;  // Move time forward

   // If a process is completed
   if(a[short_p].BT == 0) {
      count++;
      a[short_p].WT = t - a[short_p].AT - temp[short_p];
      a[short_p].TAT = t - a[short_p].AT;
```

```
            total_WT += a[short_p].WT;
            total_TAT += a[short_p].TAT;
        }
    }

    Avg_WT = total_WT / n;
    Avg_TAT = total_TAT / n;

    printf("ID  WT   TAT\n");
    for(int i = 0; i < n; i++) {
        printf("%d   %d   %d\n", i + 1, a[i].WT, a[i].TAT);
    }

    printf("Avg waiting time of the processes is %.2f\n", Avg_WT);
    printf("Avg turn around time of the processes is %.2f\n", Avg_TAT);

    return 0;
}
```

[22-11-2024 07:53] Soham: preemptive priority

[22-11-2024 07:53] Soham: #include<stdio.h>

```
struct process {
    int WT, AT, BT, TAT, PT;
};

int main() {
    struct process a[10];
    int n, temp[10], t = 0, count = 0, short_p;
    float total_WT = 0, total_TAT = 0, Avg_WT, Avg_TAT;

    printf("Enter the number of processes: ");
```

```
scanf("%d", &n);

printf("Enter the arrival time, burst time, and priority of the processes (AT BT PT):\n");
for(int i = 0; i < n; i++) {
   printf("Enter the arrival time of the process %d:\n",i+1);
   scanf("%d", &a[i].AT);
   printf("Enter the brust time of the process %d:\n",i+1);
   scanf("%d", &a[i].BT);
   printf("Enter the priority of the process %d:\n",i+1);
   scanf("%d", &a[i].PT);
   temp[i] = a[i].BT;  // Copying burst time for later use
}

a[9].PT = 10000;  // High priority value to ensure correct selection in the loop

while(count != n) {
   short_p = 9;
   for(int i = 0; i < n; i++) {
      if(a[i].PT < a[short_p].PT && a[i].AT <= t && a[i].BT > 0) {
         short_p = i;
      }
   }
   while(a[short_p].BT!=0)
   {
      a[short_p].BT--;
      t++;  // Move time forward
   }
   // If a process is completed
   if(a[short_p].BT == 0) {
      count++;
      a[short_p].WT = t - a[short_p].AT - temp[short_p];
```

```
            a[short_p].TAT = t - a[short_p].AT;
            total_WT += a[short_p].WT;
            total_TAT += a[short_p].TAT;
        }
    }

    Avg_WT = total_WT / n;
    Avg_TAT = total_TAT / n;

    printf("ID  WT   TAT\n");
    for(int i = 0; i < n; i++) {
        printf("%d   %d   %d\n", i + 1, a[i].WT, a[i].TAT);
    }

    printf("Avg waiting time of the processes is %.2f\n", Avg_WT);
    printf("Avg turn around time of the processes is %.2f\n", Avg_TAT);

    return 0;
}
```

[22-11-2024 07:54] Soham: nonn preemptive priority

[22-11-2024 07:54] Soham:

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

int disk_size;

int FCFS(int arr[], int size,int head)
{
    printf("\n\nmovement | seek time\n");
    int movement = 0;
    for (int i = 0; i < size; i++)
```

```
    {
        movement += abs(arr[i] - head);
        printf("%d-%d    | %d\n",head,arr[i],movement);
        head = arr[i];
    }
    printf("\n\n\n");
    return movement;
}

int SSTF(int arr[],int size,int head,int finished[])
{
     int movement = 0,a;
    printf("\n\nmovement | seek time\n");
    for (int i = 0; i < size; i++)
    {
        int min =999;
        for (int j = 0; j < size; j++)
        {
            if(min > abs(arr[j]-head) && finished[j] == 0)
            {
                min = abs(arr[j]-head);
                a =j;
            }
        }

        movement += abs(arr[a] - head);
         printf("%d-%d    | %d\n",head,arr[i],movement);
        head = arr[a];
        finished[a] = 1;
        arr[a] = 999;
    }
```

```
    printf("\n\n\n");
    return movement;
}

void SCAN(int arr[],int size,int head)
{
     int right[size],left[size],left_count = 0,right_count = 0,seek = 0;
    printf("\n\nmovement | seek time\n");
     right[right_count++] = disk_size -1;

     for (int i = 0; i < size; i++)
     {
       if(arr[i]<head)
       {
          left[left_count++] = arr[i];
       }
       else
       {
          right[right_count++] = arr[i];
       }
     }

     for (int i = 0; i < right_count; i++)
     {
       for (int j = i+1; j < right_count; j++)
       {
          if(right[i]>right[j])
          {
             int swap = right[j];
             right[j] = right[i];
             right[i] = swap;
```

```
      }
    }

}
for (int i = 0; i < left_count; i++)
{
   for (int j = i+1; j < left_count; j++)
   {
      if(left[i]>left[j])
      {
         int swap = left[j];
         left[j] = left[i];
         left[i] = swap;
      }
   }

}

int i;
for (i = 0; i < right_count; i++)
{
   seek += abs(right[i]-head);
    printf("%d-%d    | %d\n",head,arr[i],seek);
   head = right[i];
}
for (int j = left_count-1; j >= 0 ; j--)
{
   seek += abs(left[j]-head);
    printf("%d-%d    | %d\n",head,arr[i],seek);
   head = left[j];
}
```

```
    printf("\n\n\n");
     printf("Total seek time: %d by SCAN\n",seek);

}

void CSCAN(int arr[],int size,int head)
{
    printf("\n\nmovement | seek time\n");
     int right[size],left[size],left_count = 0,right_count = 0,seek = 0;

     right[right_count++] = disk_size -1;
     left[left_count++] = 0;

     for (int i = 0; i < size; i++)
     {
       if(arr[i]<head)
       {
          left[left_count++] = arr[i];
       }
       else
       {
          right[right_count++] = arr[i];
       }
     }

     for (int i = 0; i < right_count; i++)
     {
       for (int j = i+1; j < right_count; j++)
       {
          if(right[i]>right[j])
          {
```

```
            int swap = right[j];
            right[j] = right[i];
            right[i] = swap;
        }
    }

}
for (int i = 0; i < left_count; i++)
{
    for (int j = i+1; j < left_count; j++)
    {
        if(left[i]>left[j])
        {
            int swap = left[j];
            left[j] = left[i];
            left[i] = swap;
        }
    }

}

int i;
for (i = 0; i < right_count; i++)
{
    seek += abs(right[i]-head);
     printf("%d-%d    | %d\n",head,arr[i],seek);
    head = right[i];
}
```

```
    for (int j = 0; j < left_count ; j++)
    {
       seek += abs(left[j]-head);
        printf("%d-%d    | %d\n",head,arr[i],seek);
       head = left[j];
    }
    printf("\n\n\n");
     printf("Total seek time:  %d by CSCAN \n",seek);

}

int main()
{
    printf("Give the disk size:");
    scanf("%d",&disk_size);
    int size,head;
    printf("How many traks positions are there:");
    scanf("%d",&size);
    int arr[size],finished[size];
    for (int i = 0; i < size; i++)
    {
       printf("Give the track no %d:",i+1);
       scanf("%d",&arr[i]);
       finished[i] = 0;
    }
    printf("Give initial head position:");
    scanf("%d",&head);
    SCAN(arr,size,head);
    CSCAN(arr,size,head);
    int movement;
    movement  = FCFS(arr,size,head);
```

```
    printf("seek time is:%d\nby FCFS\n",movement);
    movement  = SSTF(arr,size,head,finished);
    printf("seek time is:%d\nby SSTF\n",movement);
    return 0;
}
[22-11-2024 07:55] Soham: disk scheduling all
[22-11-2024 07:55] Soham: #include<stdio.h>
#include<stdlib.h>

struct node
{
    int *arr;
    int f,r;
};



void print(struct node *q,int size)
{
    printf(" ");
    for (int i = 0; i < size; i++)
    {
        printf("----");
    }

    printf("\n|");

    for (int i = 0; i < size; i++)
    {
        printf(" %d |",q->arr[i]);
    }
```

```
    printf("\n");
    printf(" ");

    for (int i = 0; i < size; i++)
    {
        printf("----");
    }
    printf("\n");

}

void enqueue(struct node **q,int val)
{
    (*q)->arr[++(*q)->r] = val;
}

void FIFO(int ref[],struct node* q,int size,int req)
{
    for (int j = 0; j < size; j++)
    {
        q->arr[j] = -1;
    }
    int exist,hit = 0,fault = 0;
    for (int i = 0; i < req; i++)
    {
        if(q->r+1 == size)
        {
            q->r = -1;
        }
        exist = 0;
        // check if the page is already exist
```

```
        for (int j = 0; j < size; j++)
        {
            if(q->arr[j] == ref[i])
            {
                exist = 1;
                print(q,size);
                hit++;
            }
        }
        if(exist != 1)
        {
            enqueue(&q,ref[i]);
            print(q,size);
            fault++;
        }


    }
    printf("Page fault = %d\n",fault);
    printf("Page hit = %d",hit);

}
int main()
{
    int size;
    printf("Give the number of page frames:");
    scanf("%d",&size);
    int req;
    printf("Give the number of reference string elements present:");
    scanf("%d",&req);
    int ref[req];
```

```
    for (int i = 0; i < req; i++)
    {
        printf("Give element no. %d ",i+1);
        scanf("%d",&ref[i]);
    }

    struct node *q = (struct node *)malloc(sizeof(struct node));
    q->r = q->f = -1;
    q->arr = (int *)malloc(size * sizeof(int));


    FIFO(ref,q,size,req);

    return 0;
}
```

[22-11-2024 07:55] Soham: fifo page replacement

[22-11-2024 07:55] Soham:

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

struct node
{
    int *arr;
    int f,r;
};


void print(struct node *q,int size)
{
    printf(" ");
    for (int i = 0; i < size; i++)
```

```
    {
        printf("----");
    }

    printf("\n|");

    for (int i = 0; i < size; i++)
    {
        printf(" %d |",q->arr[i]);
    }

    printf("\n");
    printf(" ");
    for (int i = 0; i < size; i++)
    {
        printf("----");
    }
    printf("\n");

}

void enqueue(struct node **q,int val)
{
    (*q)->arr[++(*q)->r] = val;
}
void enqueue2(struct node **q,int val,int index)
{
    (*q)->arr[index] = val;
}

void LRU(int ref[],struct node* q,int size,int req)
```

```
{
    for (int j = 0; j < size; j++)
    {
        q->arr[j] = -1;
    }

    int exist,hit = 0,fault = 0,dist = 0,max,unuse;
    for (int i = 0; i < req; i++)
    {
        exist = 0;
        for (int j = 0; j < size; j++)
        {
            if(ref[i] == q->arr[j])
            {
                print(q,size);
                exist = 1;
                hit++;
                break;
            }
        }


        if(!exist)
        {
            if(q->r < size-1)
            {
                enqueue(&q,ref[i]);
                print(q,size);
                fault++;
            }
```

```
else
{

   max = 0;
   for (int j = 0; j < size; j++)
   {
      dist = 0;
      for (int k = i; k >= 0; k--)
      {
         if(q->arr[j] == ref[k] && dist > max)
         {
            max = dist;
            unuse = j;
            break;
         }
         if(q->arr[j] == ref[k])
         {
            break;
         }
         if(k+1 == req)
         {
            unuse = j;
            max = 999;
            goto here;
         }
         dist++;
      }
   }

   here:
   enqueue2(&q,ref[i],unuse);
```

```
            fault++;
            print(q,size);
          }

       }
    }


    printf("Page fault = %d\n",fault);
    printf("Page hit = %d",hit);

}
int main()
{
    int size;
    printf("Give the number of page frames:");
    scanf("%d",&size);
    int req;
    printf("Give the number of reference string elements present:");
    scanf("%d",&req);
    int ref[req];
    for (int i = 0; i < req; i++)
    {
        printf("Give element no. %d ",i+1);
        scanf("%d",&ref[i]);
    }

    struct node *q = (struct node *)malloc(sizeof(struct node));
    q->r = q->f = -1;
    q->arr = (int *)malloc(size * sizeof(int));
```

```
    LRU(ref,q,size,req);

    return 0;
}
```

[22-11-2024 07:56] Soham: LRU page replacement

[22-11-2024 07:56] Soham:

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

struct node
{
    int *arr;
    int f,r;
};


void print(struct node *q,int size)
{
    printf(" ");
    for (int i = 0; i < size; i++)
    {
        printf("----");
    }

    printf("\n|");

    for (int i = 0; i < size; i++)
    {
        printf(" %d |",q->arr[i]);
    }
```

```
    printf("\n");
    printf(" ");
    for (int i = 0; i < size; i++)
    {
        printf("----");
    }
    printf("\n");

}

void enqueue(struct node **q,int val)
{
    (*q)->arr[++(*q)->r] = val;
}
void enqueue2(struct node **q,int val,int index)
{
    (*q)->arr[index] = val;
}

void optimal(int ref[],struct node* q,int size,int req)
{
    for (int j = 0; j < size; j++)
    {
        q->arr[j] = -1;
    }

    int exist,hit = 0,fault = 0,dist = 0,max,unuse;
    for (int i = 0; i < req; i++)
    {
        exist = 0;
        for (int j = 0; j < size; j++)
```

```
{
   if(ref[i] == q->arr[j])
   {
      exist = 1;
      print(q,size);
      hit++;
      break;
   }
}



if(!exist)
{
   if(q->r < size-1)
   {
      enqueue(&q,ref[i]);
      print(q,size);
      fault++;
   }
   else
   {


      max = 0;
      for (int j = 0; j < size; j++)
      {
         dist = 0;
         for (int k = i; k < req; k++)
         {
            if(q->arr[j] == ref[k] && dist > max)
            {
```

```
                max = dist;
                unuse = j;
                break;
            }
            if(q->arr[j] == ref[k])
            {
                break;
            }
            if(k+1 == req)
            {
                unuse = j;
                max = 999;
                goto here;
            }
            dist++;
          }
        }

        here:
        enqueue2(&q,ref[i],unuse);
        fault++;
        print(q,size);
      }

   }
}


printf("Page fault = %d\n",fault);
printf("Page hit = %d",hit);
```

```
}
int main()
{
   int size;
   printf("Give the number of page frames:");
   scanf("%d",&size);
   int req;
   printf("Give the number of reference string elements present:");
   scanf("%d",&req);
   int ref[req];
   for (int i = 0; i < req; i++)
   {
      printf("Give element no. %d ",i+1);
      scanf("%d",&ref[i]);
   }

   struct node *q = (struct node *)malloc(sizeof(struct node));
   q->r = q->f = -1;
   q->arr = (int *)malloc(size * sizeof(int));



   optimal(ref,q,size,req);


   return 0;
}
[22-11-2024 07:56] Soham: optimal
[22-11-2024 07:56] Soham: #include<stdio.h>

void firstFit(int process[],int n,int block[],int size_block)
{
   int counter,remaining;
```

```
    printf("process   process size   block   remaining_block_size\n");
    for (int i = 0; i < n; i++)
    {
        counter =0;
        for (int j = 0; j < size_block; j++)
        {
            if(process[i]<block[j])
            {
                remaining = block[j]-process[i];
                printf("%d       %d           %d     %d\n",i+1,process[i],block[j],remaining);
                block[j] = block[j]-process[i];
                break;
            }
            else
            {
                counter++;
            }
        }
        if(counter == size_block)
        {
            printf("%d       %d           not allocated\n",i+1,process[i]);
        }

    }

}

void nextFit(int process[],int n,int block[],int size_block)
{
    int counter,remaining;
    int pointer = 0;
```

```
    printf("process   process size   block   remaining_block_size\n");
    for (int i = 0; i < n; i++)
    {
        counter =pointer;
        for (int j = pointer; j < size_block; j++)
        {
            if(process[i]<=block[j])
            {
                remaining = block[j]-process[i];
                printf("%d          %d              %d      %d\n",i+1,process[i],block[j],remaining);
                block[j] = block[j]-process[i];
                pointer = j;
                if(pointer == size_block-1)
                {
                    pointer = 0;
                }
                break;
            }
            else
            {
                counter++;
            }
        }
        if(counter == size_block)
        {
            printf("%d          %d              not allocated\n",i+1,process[i]);
        }

    }

}
```

```
void bestFit(int process[],int n,int block[],int size_block)
{
   int counter,remaining,min,a;
   printf("process   process size   block   remaining_block_size\n");
   for (int i = 0; i < n; i++)
   {
      counter =0;
      min = 999;
      for (int j = 0; j < size_block; j++)
      {
         if(min > (block[j] - process[i] )&& (block[j] - process[i])>=0)
         {
            min = block[j] - process[i];
            a = j;
         }
         else
         {
            counter++;
         }
      }
      if(counter == size_block)
      {
         printf("%d       %d           not allocated\n",i+1,process[i]);
      }
      else
      {
         remaining = block[a]-process[i];
         printf("%d       %d           %d     %d\n",i+1,process[i],block[a],remaining);
         block[a] = block[a]-process[i];
```

```
        }


    }

}

void worstFit(int process[],int n,int block[],int size_block)
{
    int counter,remaining,max,a;
    printf("process   process size   block   remaining_block_size\n");
    for (int i = 0; i < n; i++)
    {
        counter =0;
        max = 0;
        for (int j = 0; j < size_block; j++)
        {
            if(max < (block[j] - process[i] ))
            {
                max = block[j] - process[i];
                a = j;
            }
            else
            {
                counter++;
            }
        }
        if(counter == size_block)
        {
            printf("%d       %d           not allocated\n",i+1,process[i]);
        }
        else
```

```
        {
            remaining = block[a]-process[i];
            printf("%d       %d          %d     %d\n",i+1,process[i],block[a],remaining);
            block[a] = block[a]-process[i];

        }

    }

}



int main()
{
    int size_block;
    printf("Give the number of the blocks which are vacant:");
    scanf("%d",&size_block);

    int block[size_block];
    printf("Give the block size for:\n");
    for (int i = 0; i < size_block; i++)
    {
        printf("Block %d:", i+1);
        scanf("%d",&block[i]);
    }

    int n;
    printf("Give the number of processes:");
    scanf("%d",&n);
```

```
    int process[n];
    printf("give the process size\n");
    for (int i = 0; i < n; i++)
    {
        printf("Process %d:",i+1);
        scanf("%d",&process[i]);
    }

    //  firstFit(process,n,block,size_block);
    // nextFit(process,n,block,size_block);
    // bestFit(process,n,block,size_block);
    worstFit(process,n,block,size_block);

    return 0;
}
```

[22-11-2024 07:56] Soham: placement stratergies