 FCFS

#include<stdio.h>  
int main()  
{  
    int at[10] ,bt[10], wt[10], ft[10], tat[10];  
    int i, j, n, temp1, temp2, x=0;  
    int twt=0, ttat=0;  
    float awt1, atat1;  
    printf("Enter number of processes:\n");  
    scanf("%d", &n);  
    printf("Enter arrival time:\n");  
    for(i=0;i<n;i++)  
        scanf("%d", &at[i]);  
    printf("Enter burst times:\n");  
    for(i=0;i<n;i++)  
        scanf("%d", &bt[i]);  
    printf("Sorting arrival times and burst times\n");  
    for(i=0;i<n;i++)  
    {  
        for(j=i+1;j<n;j++)  
        {  
            if(at[i]>at[j])  
            {  
                temp1=at[i];  
                at[i]=at[j];  
                at[j]=temp1;  
                temp2=bt[i];  
                bt[i]=bt[j];  
                bt[j]=temp2;  
            }  
        }  
    }  
    for(i=0;i<n;i++)  
        printf("%d\n\t%d\n", at[i], bt[i]);  
    printf("Calculation of finishing time, waiting time, turn around time\n");  
    for(i=0;i<n;i++)  
    {  
        x=x+bt[i];  
        ft[i]=x;  
        wt[i]=ft[i]-(at[i]+bt[i]);  
        tat[i]=ft[i]-at[i];  
        printf("%5d\t   %5d\t   %5d\n", ft[i], wt[i], tat[i]);  
    }  
    for(i=0;i<n;i++)   
    {  
        twt=twt+wt[i];  
        ttat=ttat+tat[i];  
    }  
    awt1=(float)twt/n;  
    atat1=(float)ttat/n;  
    printf("Average waiting time is:%f\nAverage turn around time is:%f\n",awt1,atat1);  
}  
output:  
Enter number of processes:  
4  
Enter arrival time:  
1 0 2 4  
Enter burst times:  
10 12 8 16  
Sorting arrival times and burst times  
0  
12  
1  
10  
2  
8  
4  
16  
Calculation of finishing time, waiting time, turn around time  
   12       0      12  
   22      11      21  
   30      20      28  
   46      26      42  
Average waiting time is:14.250000  
Average turn around time is:25.750000  
  
  
=== Code Execution Successful ===

                                                        SJF

#include<stdio.h>  
#define max 20  
int main()  
{  
    int bt[max], at[max], ft[max], wt[max], tat[max];  
    int i, j, n, x=0, y=0, z=0, min=0, temp1, temp2, k, l;  
    float t, u;  
    printf("Enter number of processes to be executed:\n");  
    scanf("%d", &n);  
    for(i=1;i<=n;i++)  
    {  
        printf("Enter burst time for process-%d:\n", i);  
        scanf("%d", &bt[i]);  
        printf("Enter arrival for process-%d:\n", i);  
        scanf("%d", &at[i]);  
    }   
    for(i=1;i<n;i++)  
    {  
        for(j=i+1;j<=n;j++)  
        {  
            if(at[i]>at[j])  
            {  
                temp1=at[i];  
                at[i]=at[j];  
                at[j]=temp1;  
                temp2=bt[i];  
                bt[i]=bt[j];  
                bt[j]=temp2;  
            }  
            else if(at[i]==at[j]&&bt[i]>bt[j])  
            {  
                temp1=at[i];  
                at[i]=at[j];  
                at[j]=temp1;  
                temp2=bt[i];  
                bt[i]=bt[j];  
                bt[j]=temp2;  
            }  
        }  
    }  
    for(i=1;i<=n;i++)  
    {  
        min=min+bt[i];  
        for(j=i+1;at[j]<=min;j++)  
        {  
            for(k=j+1;at[k]<=min;k++)  
            {  
                if(bt[k]<bt[j])  
                {  
                    temp1=bt[k];  
                    bt[k]=bt[j];  
                    bt[j]=temp1;  
                    temp2=at[k];  
                    at[k]=at[j];  
                    at[j]=temp2;  
                }  
            }  
        }  
    }  
    for(i=1;i<=n;i++)  
    {  
        x=x+bt[i];  
        ft[i]=x;  
        if(i==1)  
            wt[i]=y;  
        else  
            wt[i]=ft[i-1]-at[i];  
        tat[i]=bt[i]+wt[i];  
    }  
    for(i=1;i<=n;i++)  
    {  
        y=y+tat[i];  
        z=z+wt[i];  
    }  
    for(i=1;i<=n;i++)  
        printf("\nProcess:%d-->at:%d\t bt:%d\t ft:%d\t wt:%d\t tat:%d\t", i, at[i], bt[i], ft[i], wt[i], tat[i]);  
    printf("\nAverage waiting time:%d",z/n);  
    printf("\nAverage tat time:%d",y/n);  
}  
output:  
Enter number of processes to be executed:  
4  
Enter burst time for process-1:  
10  
Enter arrival for process-1:  
1  
Enter burst time for process-2:  
12  
Enter arrival for process-2:  
0  
Enter burst time for process-3:  
18  
Enter arrival for process-3:  
2  
Enter burst time for process-4:  
4  
Enter arrival for process-4:  
4  
  
Process:1-->at:0         bt:12   ft:12   wt:0    tat:12  
Process:2-->at:4         bt:4    ft:16   wt:8    tat:12  
Process:3-->at:1         bt:10   ft:26   wt:15   tat:25  
Process:4-->at:2         bt:18   ft:44   wt:24   tat:42  
Average waiting time:11  
Average tat time:22  
  
...Program finished with exit code 0  
Press ENTER to exit console.

                                            PRIORITY

#include<stdio.h>  
#define max 20  
int main()  
{  
    int bt[max], at[max], ft[max], wt[max], tat[max], p[max];  
    int i, j, n, x=0, y=0, z=0, temp1, temp2, temp3, min=0, k, l;  
    float t, u;  
    printf("Enter number of processes to be executed:\n");  
    scanf("%d", &n);  
    for(i=1;i<=n;i++)  
    {  
        printf("Enter burst time for process-%d:\n", i);  
        scanf("%d", &bt[i]);  
        printf("Enter arrival for process-%d:\n", i);  
        scanf("%d", &at[i]);  
        printf("Enter the priority of process-%d:\n",i);  
        scanf("%d", &p[i]);  
    }  
    for(i=1;i<=n;i++)  
    {  
        for(j=i+1;j<=n;j++)  
        {  
            if(at[i]>at[j])  
            {  
                temp1=at[i];  
                at[i]=at[j];  
                at[j]=temp1;  
                temp2=bt[i];  
                bt[i]=bt[j];  
                bt[j]=temp2;  
                temp3=p[i];  
                p[i]=p[j];  
                p[j]=temp3;  
            }  
            else if(at[i]==at[j]&&p[i]>p[j])  
            {  
                temp1=at[i];  
                at[i]=at[j];  
                at[j]=temp1;  
                temp2=bt[i];  
                bt[i]=bt[j];  
                bt[j]=temp2;  
                temp3=p[i];  
                p[i]=p[j];  
                p[j]=temp3;  
            }  
        }  
    }  
    for(i=1;i<=n;i++)  
    {  
        min=min+bt[i];  
        for(j=i+1;at[j]<=min;j++)  
        {  
            for(k=j+1;at[k]<=min;k++)  
            {  
                if(p[k]<p[j])  
                {  
                    temp1=bt[k];  
                    bt[k]=bt[j];  
                    bt[j]=temp1;  
                    temp2=at[k];  
                    at[k]=at[j];  
                    at[j]=temp2;  
                    temp3=p[i];  
                    p[i]=p[j];  
                    p[j]=temp3;  
                }  
            }  
        }  
    }  
    for(i=1;i<=n;i++)  
    {  
        x=x+bt[i];  
        ft[i]=x;  
        if(i==1)  
            wt[i]=y;  
        else  
            wt[i]=ft[i-1]-at[i];  
        tat[i]=bt[i]+wt[i];  
    }  
    for(i=1;i<=n;i++)  
    {  
        y=y+tat[i];  
        z=z+wt[i];  
    }  
    for(i=1;i<=n;i++)  
        printf("\nProcess:%d-->at:%d\t bt:%d\t ft:%d\t wt:%d\t tat:%d\t", i, at[i], bt[i], ft[i], wt[i], tat[i]);  
    printf("\nAverage waiting time:%d",z/n);  
    printf("\nAverage turn around time:%d",y/n);  
}  
output:  
Enter number of processes to be executed:  
5  
Enter burst time for process-1:  
10  
Enter arrival for process-1:  
3  
Enter the priority of process-1:  
3  
Enter burst time for process-2:  
1  
Enter arrival for process-2:  
1  
Enter the priority of process-2:  
1  
Enter burst time for process-3:  
2  
Enter arrival for process-3:  
0  
Enter the priority of process-3:  
4  
Enter burst time for process-4:  
1  
Enter arrival for process-4:  
2  
Enter the priority of process-4:  
4  
Enter burst time for process-5:  
5  
Enter arrival for process-5:  
4  
Enter the priority of process-5:  
2  
  
Process:1-->at:0         bt:2    ft:2    wt:0    tat:2  
Process:2-->at:1         bt:1    ft:3    wt:1    tat:2  
Process:3-->at:3         bt:10   ft:13   wt:0    tat:10  
Process:4-->at:4         bt:5    ft:18   wt:9    tat:14  
Process:5-->at:2         bt:1    ft:19   wt:16   tat:17  
Average waiting time:5  
Average turn around time:9  
  
...Program finished with exit code 0  
Press ENTER to exit console.

                                          ROUND ROBIN

#include<stdio.h>  
int main()  
{  
    int st[10], at[10], bt[10], wt[10], tat[10], n, tq;  
    int i, count=0, swt=0, stat=0, temp, sq=0, j, temp1, temp2;  
    float awt=0.0, atat=0.0;  
    printf("\nEnter number of processes to be executed:");  
    scanf("%d", &n);  
    printf("\nEnter burst time sequences:");  
    for(i=0;i<n;i++)  
        scanf("%d", &bt[i]);  
    printf("Enter arrival times:\n");  
    for(i=0;i<n;i++)  
        scanf("%d", &at[i]);  
    printf("After sorting arrival times and burst times\n");  
    for(i=0;i<n;i++)  
    {  
        for(j=i+1;j<n;j++)  
        {  
            if(at[i]>at[j])  
            {  
                temp1=at[i];  
                at[i]=at[j];  
                at[j]=temp1;  
                temp2=bt[i];  
                bt[i]=bt[j];  
                bt[j]=temp2;  
            }  
        }  
    }  
    for(i=0;i<n;i++)  
    {  
        st[i]=bt[i];  
        printf("%d\t%d\t\n", at[i], bt[i]);  
    }  
    printf("\nEnter time quantum:");  
    scanf("%d", &tq);  
    while(1)  
    {  
        for(i=0, count=0;i<n;i++)  
        {  
            temp=tq;  
            if(st[i]==0)  
            {  
                count++;  
                continue;  
            }  
            if(st[i]>tq)  
                st[i]=st[i]-tq;  
            else if(st[i]>=0)  
            {  
                temp=st[i];  
                st[i]=0;  
            }  
            sq=sq+temp;  
            tat[i]=sq-at[i];  
        }  
        if(n==count)  
            break;  
    }  
    for(i=0;i<n;i++)  
    {  
        wt[i]=tat[i]-bt[i];  
        swt=swt+wt[i];  
        stat=stat+tat[i];  
    }  
    awt=(float)swt/n;  
    atat=(float)stat/n;  
    printf("PNO\t\tat\t\tbt\t\twt\t\ttat\n");  
    for(i=0;i<n;i++)  
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i+1, at[i], bt[i], wt[i],tat[i]);  
    printf("\nAverage wt is:%f\nAverage tat is:%f\n", awt, atat);  
    return 0;  
}  
output:  
  
Enter number of processes to be executed:3  
  
Enter burst time sequences:25 25 10  
Enter arrival times:  
3  
4  
2  
After sorting arrival times and burst times  
2 10  
3 25  
4 25  
  
Enter time quantum:5  
PNO at bt wt tat  
1 2 10 8 18  
2 3 25 27 52  
3 4 25 31 56  
  
Average wt is:22.000000  
Average tat is:42.000000  
  
  
=== Code Execution Successful ===

                                                       FIRSTFIT

#include <stdio.h>  
#include<stdlib.h>  
void firstFit(int blockSize[], int m, int processSize[], int n)  
{  
    int allocation[n];  
    for(int i=0; i<n; i++)  
        allocation[i]=-1;  
    for(int i=0; i<n; i++)  
    {  
        int firstIdx=-1;  
        for(int j=0; j<m; j++)  
        {  
            if(blockSize[j]>=processSize[i])  
            {  
                firstIdx=j;  
                break;  
            }  
        }  
        if (firstIdx!=-1)  
        {  
            allocation[i]=firstIdx;  
            blockSize[firstIdx]=blockSize[firstIdx]-processSize[i];  
        }  
    }  
    printf("Process No. \tProcess Size \tBlock No. \n");  
    for(int i=0; i<n; i++)  
    {  
        printf("%d\t\t%d\t\t", i, processSize[i]);  
        if(allocation[i]!=-1)  
            printf("%d\n", allocation[i]);  
        else  
            printf("Not Allocated\n");  
    }  
}  
void main()  
{  
    int i, bs, p, blockSize[10], processSize[10];  
    printf("Enter no. of blocks:");  
    scanf("%d", &bs);  
    for(i=0;i<bs;i++)  
    {  
        printf("Enter %d block size:", i);  
        scanf("%d", &blockSize[i]);  
    }  
    printf("Enter no. of process:");  
    scanf("%d", &p);  
    for(i=0;i<p;i++)  
    {  
        printf("Enter %d process size:", i);  
        scanf("%d", &processSize[i]);  
    }  
    firstFit(blockSize, bs, processSize, p);  
}  
output:  
Enter no. of blocks:4  
Enter 0 block size:100  
Enter 1 block size:200  
Enter 2 block size:300  
Enter 3 block size:400  
Enter no. of process:4  
Enter 0 process size:20  
Enter 1 process size:30  
Enter 2 process size:40  
Enter 3 process size:50  
Process No. Process Size Block No.  
0           20           0  
1         30         0  
2         40         0  
3         50        1  
  
  
=== Code Exited With Errors ===

                                                        WORSTFIT

#include <stdio.h>  
#include<stdlib.h>  
void worstFit(int blockSize[],int m,int processSize[],int n){  
    int allocation[n];  
    for(int i=0;i<n;i++)  
      allocation[i]=-1;  
    for(int i=0;i<n;i++){  
        int worstIdx=-1;  
        for(int j=0;j<m;j++){  
            if(blockSize[j]>=processSize[i]){  
                if(worstIdx==-1 || blockSize[j]>blockSize[worstIdx])  
                worstIdx=j;  
            }  
             
        }  
            if(worstIdx!=-1){  
                allocation[i]=worstIdx;  
                blockSize[worstIdx]=blockSize[worstIdx]-processSize[i];  
            }  
        }  
            printf("Process No.\t Process size \tBlock NO.\n");  
            for(int i=0;i<n;i++){  
                printf("%d\t\t%d\t\t",i,processSize[i]);  
                if(allocation[i]!=-1)  
                printf("%d\n",allocation[i]);  
                else  
                printf("not Allocated\n");  
            }  
    }  
    void main(){  
        int i,bs,p,blockSize[10],processSize[10];  
        printf("Enter no of blocks: ");  
        scanf("%d",&bs);  
        for(i=0;i<bs;i++){  
            printf("Enter %d block size: ",i);  
            scanf("%d",&blockSize[i]);  
        }  
        printf("Enter no of process: ");  
        scanf("%d",&p);  
        for(i=0;i<p;i++){  
            printf("Enter %d process size: ",i);  
            scanf("%d",&processSize[i]);  
        }  
        worstFit(blockSize,bs,processSize,p);  
}  
output:  
Enter no of blocks: 3  
Enter 0 block size: 50  
Enter 1 block size: 30  
Enter 2 block size: 70  
Enter no of process: 4  
Enter 0 process size: 20  
Enter 1 process size: 10  
Enter 2 process size: 30  
Enter 3 process size: 50  
Process No. Process size Block NO.  
0 20 2  
1 10 0  
2 30 2  
3 50 not Allocated  
  
  
=== Code Exited With Errors ===

                                         BESTFIT

#include <stdio.h>  
#include <stdlib.h>  
void BestFit(int blockSize[], int m, int processSize[], int n)  
{  
 int allocation[n];  
 for (int i = 0; i < n; i++)  
   allocation[i] = - 1;  
 for (int i = 0; i < n; i++)  
 {  
   int bestIdx = - 1;  
   for (int j = 0; j < m; j++)  
  {  
   if (blockSize[j] >= processSize[i])  
   {  
    if (bestIdx == - 1 || blockSize[j] < blockSize [ bestIdx])  
      bestIdx = j;  
   }  
  }  
  if(bestIdx != - 1)  
  {  
   allocation[i] = bestIdx;  
   blockSize [bestIdx]= blockSize [bestIdx] -processSize[i];  
  }  
 }  
 printf ("Process No. \tProcess Size \tBlock No. \n");  
 for (int i = 0; i < n; i++)  
 {  
  printf("%d\t\t%d\t\t", i, processSize[i]);  
  if (allocation[i] != - 1)  
    printf("%d\n", allocation[i]);  
  else  
    printf("Not Allocated\n");  
  }  
}  
void main()  
{  
 int i,bs,p,blockSize[10], processSize[10];  
 printf("Enter no.of blocks:");  
 scanf("%d",&bs);  
 for(i=0;i<bs;i++)  
 {  
  printf("Enter %d block size:", i);  
  scanf("%d", &blockSize[i]);  
 }  
 printf ("Enter no.of process:");  
 scanf("%d",&p);  
 for(i=0;i<p; i++)  
 {  
  printf("Enter %d process size:",i);  
  scanf("%d", &processSize [i]);  
 }  
 BestFit(blockSize, bs, processSize, p);  
}  
output:  
Enter no.of blocks:4  
Enter 0 block size:250  
Enter 1 block size:500  
Enter 2 block size:300  
Enter 3 block size:600

Enter no.of process:5  
Enter 0 process size:100  
Enter 1 process size:400  
Enter 2 process size:200  
Enter 3 process size:350  
Enter 4 process size:150  
Process No. Process Size Block No.  
0 100 0  
1 400 1  
2 200 2  
3 350 3  
4 150 0  
  
  
=== Code Exited With Errors ===

                                           FIFO

#include<stdio.h>  
int main()  
{  
    int i, j, n, a[50],frame[10], no, k, avail, count=0;  
    printf("\n Enter the number of pages:\n");  
    scanf("%d", &n);  
    printf("\n Enter the page number:\n");  
    for(i=1;i<=n;i++)  
        scanf("%d", &a[i]);  
    printf("\n Enter the number of frames:");  
    scanf("%d", &no);  
    for(i=0;i<no;i++)  
        frame[i]=-1;  
    j=0;  
    printf("\tref string\tpage frames\n");  
    for(i=1;i<=n;i++)  
    {  
        printf("%d\t\t\t", a[i]);  
        avail=0;  
        for(k=0;k<no;k++)  
        if(frame[k]==a[i])  
                avail=1;  
        if(avail==0)  
        {  
            frame[j]=a[i];  
            j=(j+1)%no;  
            count++;  
            // Print the current state of the frames  
            for(k=0;k<no;k++)  
                printf("%d\t", frame[k]);  
        }  
        printf("\n");  
    }  
    // Output the total number of page faults  
    printf("Page Fault is %d", count);  
    return 0;  
}  
output:  
  
Enter the number of pages:  
20  
  
 Enter the page number:  
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1  
  
 Enter the number of frames:3  
ref string page frames  
7 7 -1 -1  
0 7 0 -1  
1 7 0 1  
2 2 0 1  
0  
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  
2  
1 0 1 3  
2 0 1 2  
0  
1  
7 7 1 2  
0 7 0 2  
1 7 0 1  
Page Fault is 15  
  
=== Code Execution Successful ===

                                             BANKERS

#include <stdio.h>  
  
int main() {  
    int available[3], work[3], max[5][3], allocation[5][3], need[5][3], safe[5], totalres[3];  
    char finish[5];  
    int i, j, k, totalloc = 0, state, value = 0;  
  
    // Input total resources  
    printf("Enter instances of each resource:\n");  
    for (i = 0; i < 3; i++) {  
        scanf("%d", &totalres[i]);  
    }  
  
    // Input maximum resources for each process  
    printf("Enter maximum number of resources for each process:\n");  
    for (i = 0; i < 5; i++) {  
        for (j = 0; j < 3; j++) {  
            printf("Enter process-%d resource-%d: ", i, (j + 1));  
            scanf("%d", &max[i][j]);  
        }  
    }  
  
    // Input allocated resources to each process  
    printf("Enter number of resources allocated to each process:\n");  
    for (i = 0; i < 5; i++) {  
        for (j = 0; j < 3; j++) {  
            printf("Enter the resource R%d allocated to process %d: ", (j + 1), i);  
            scanf("%d", &allocation[i][j]);  
        }  
    }  
  
    // Calculate need matrix  
    for (i = 0; i < 5; i++) {  
        for (j = 0; j < 3; j++) {  
            need[i][j] = max[i][j] - allocation[i][j];  
        }  
    }  
  
    // Initialize finish array  
    for (i = 0; i < 5; i++) {  
        finish[i] = 'f';  // all processes are initially unfinished  
    }  
  
    // Calculate available resources  
    for (i = 0; i < 3; i++) {  
        totalloc = 0;  
        for (j = 0; j < 5; j++) {  
            totalloc = totalloc + allocation[j][i];  
        }  
        available[i] = totalres[i] - totalloc;  
        work[i] = available[i];  
    }  
  
    // Display resources  
    printf("Allocated Resources:\n");  
    for (i = 0; i < 5; i++) {  
        for (j = 0; j < 3; j++) {  
            printf("%d ", allocation[i][j]);  
        }  
        printf("\n");  
    }  
  
    printf("Maximum Resources:\n");  
    for (i = 0; i < 5; i++) {  
        for (j = 0; j < 3; j++) {  
            printf("%d ", max[i][j]);  
        }  
        printf("\n");  
    }  
  
    printf("Needed Resources:\n");  
    for (i = 0; i < 5; i++) {  
        for (j = 0; j < 3; j++) {  
            printf("%d ", need[i][j]);  
        }  
        printf("\n");  
    }  
  
    printf("Available Resources: ");  
    for (i = 0; i < 3; i++) {  
        printf("%d ", available[i]);  
    }  
    printf("\n");  
  
    // Safe State Calculation  
    for (i = 0; i < 5; i++) {  
        for (j = 0; j < 3; j++) {  
            if (finish[i] == 'f' && need[i][j] <= work[j]) {  
                state = 1;  
            } else {  
                state = 0;  
                break;  
            }  
        }  
  
        // If state is 1, update work and finish  
        if (state == 1) {  
            for (j = 0; j < 3; j++) {  
                work[j] = work[j] + allocation[i][j];  
            }  
            finish[i] = 't'; // mark process as finished  
            safe[value] = i;  
            value++;  
        }  
  
        // If we haven't finished all processes, try again  
        if (i == 4) {  
            if (value == 5) {  
                break;  
            } else {  
                i = -1; // restart the loop  
            }  
        }  
    }  
  
    // Output the safe state sequence  
    printf("Safe sequence: ");  
    for (i = 0; i < value; i++) {  
        printf("p%d ", safe[i]);  
    }  
    printf("\n");  
  
    return 0;  
}  
output:  
Enter instances of each resource:  
10 5 7  
Enter maximum number of resources for each process:  
Enter process-0 resource-1: 7  
Enter process-0 resource-2: 5  
Enter process-0 resource-3: 3  
Enter process-1 resource-1: 3  
Enter process-1 resource-2: 2  
Enter process-1 resource-3: 2  
Enter process-2 resource-1: 9  
Enter process-2 resource-2: 0  
Enter process-2 resource-3: 2  
Enter process-3 resource-1: 2  
Enter process-3 resource-2: 2  
Enter process-3 resource-3: 2  
Enter process-4 resource-1: 4  
Enter process-4 resource-2: 3  
Enter process-4 resource-3: 3  
Enter number of resources allocated to each process:  
Enter the resource R1 allocated to process 0: 0  
Enter the resource R2 allocated to process 0: 1  
Enter the resource R3 allocated to process 0: 0  
Enter the resource R1 allocated to process 1: 2  
Enter the resource R2 allocated to process 1: 0  
Enter the resource R3 allocated to process 1: 0  
Enter the resource R1 allocated to process 2: 3  
Enter the resource R2 allocated to process 2: 0  
Enter the resource R3 allocated to process 2: 2  
Enter the resource R1 allocated to process 3: 2  
Enter the resource R2 allocated to process 3: 1  
Enter the resource R3 allocated to process 3: 1  
Enter the resource R1 allocated to process 4: 0  
Enter the resource R2 allocated to process 4: 0  
Enter the resource R3 allocated to process 4: 2  
Allocated Resources:  
0 1 0  
2 0 0  
3 0 2  
2 1 1  
0 0 2  
Maximum Resources:  
7 5 3  
3 2 2  
9 0 2  
2 2 2  
4 3 3  
Needed Resources:  
7 4 3  
1 2 2  
6 0 0  
0 1 1  
4 3 1  
Available Resources: 3 3 2  
Safe sequence: p1 p3 p4 p0 p2  
  
  
=== Code Execution Successful ===

                                PRODUCER&CONSUMER

#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <unistd.h>  
#include <signal.h>  
  
#define MAX 5         // Buffer size  
#define TOTAL 10      // Total number of items to produce/consume  
  
// Shared buffer  
int buffer[MAX];  
int in = 0, out = 0;  // Circular buffer index  
  
// Synchronization tools  
sem\_t empty, full;  
pthread\_mutex\_t mutex;  
int running = 1;  // Flag to control loop execution  
  
void handle\_signal(int sig) {  
    printf("\nReceived signal %d. Cleaning up and exiting...\n", sig);  
    running = 0;  // Stop producer and consumer loops  
}  
  
// Producer function  
void\* producer(void\* param) {  
    int item;  
    for (int i = 0; i < TOTAL && running; i++) {  
        item = rand() % 100;  // Generate a random item  
  
        sem\_wait(&empty);  // Wait for an empty slot  
        pthread\_mutex\_lock(&mutex);  
  
        buffer[in] = item;  // Place item in buffer  
        printf("Produced: %d at index %d\n", item, in);  
        in = (in + 1) % MAX;  // Move to next index in circular buffer  
  
        pthread\_mutex\_unlock(&mutex);  
        sem\_post(&full);  // Signal that an item is available  
  
        sleep(1);  // Simulate time taken to produce  
    }  
    return NULL;  
}  
  
// Consumer function  
void\* consumer(void\* param) {  
    int item;  
    for (int i = 0; i < TOTAL && running; i++) {  
        sem\_wait(&full);  // Wait until there is an item  
        pthread\_mutex\_lock(&mutex);  
  
        item = buffer[out];  // Retrieve item  
        printf("Consumed: %d from index %d\n", item, out);  
        out = (out + 1) % MAX;  // Move to next index in circular buffer  
  
        pthread\_mutex\_unlock(&mutex);  
        sem\_post(&empty);  // Signal that a slot is empty  
  
        sleep(1);  // Simulate time taken to consume  
    }  
    return NULL;  
}  
  
int main() {  
    srand(time(NULL));  // Seed random number generator  
    signal(SIGINT, handle\_signal);  // Register signal handler  
  
    pthread\_t prod, cons;  
  
    // Initialize semaphores and mutex  
    sem\_init(&empty, 0, MAX);  
    sem\_init(&full, 0, 0);  
    pthread\_mutex\_init(&mutex, NULL);  
  
    // Create producer and consumer threads  
    pthread\_create(&prod, NULL, producer, NULL);  
    pthread\_create(&cons, NULL, consumer, NULL);  
  
    // Wait for threads to finish  
    pthread\_join(prod, NULL);  
    pthread\_join(cons, NULL);  
  
    // Cleanup resources  
    sem\_destroy(&empty);  
    sem\_destroy(&full);  
    pthread\_mutex\_destroy(&mutex);  
  
    printf("Program finished successfully!\n");  
    return 0;  
}