4. Simulate the following CPU scheduling algorithms

a) FCFS b) SJF c) Priority d) Round Robin

**fcfs**

**#include <stdio.h>**

**int main() {**

**int at[10], bt[10], wt[10], ft[10], tat[10], i, j, n, temp1, temp2, x = 0;**

**int twt = 0, ttat = 0;**

**float awt1, atat1;**

**printf("Enter number of processes: ");**

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

**printf("Enter arrival times:\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]);**

**}**

**for (i = 0; i < n - 1; 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;**

**}**

**}**

**}**

**printf("Arrival Time\tBurst Time\n");**

**for (i = 0; i < n; i++) {**

**printf("%d\t\t%d\n", at[i], bt[i]);**

**}**

**printf("Calculating Finishing Time, Waiting Time, Turnaround 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];**

**twt += wt[i];**

**ttat += tat[i];**

**printf("Process %d: FT=%d, WT=%d, TAT=%d\n", i + 1, ft[i], wt[i], tat[i]);**

**}**

**awt1 = (float)twt / n;**

**atat1 = (float)ttat / n;**

**printf("Average Waiting Time: %.2f\n", awt1);**

**printf("Average Turnaround Time: %.2f\n", atat1);**

**return 0;**

**}**

**Round Robin**

**#include<stdio.h>  
int main(){  
    int at[10], bt[10], wt[10], st[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("Enter number of processes:\n");  
    scanf("%d", &n);  
    printf("Enter burst times:\n");  
    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]);  
    }  
    for(i = 0; i < n - 1; 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("Arrival time: %d, Burst time: %d\n", at[i], bt[i]);  
    }  
    printf("\nEnter time quantum: ");  
    scanf("%d", &tq);  
    // Round Robin Scheduling  
    while(1) {  
        count = 0;  
        for(i = 0; i < n; i++) {  
            temp = tq;  
            if(st[i] == 0) {  
                count++;  
                continue;  
            }  
            if(st[i] > tq) {  
                st[i] -= tq;  
            } else {  
                temp = st[i];  
                st[i] = 0;  
            }  
            sq += temp;  
            tat[i] = sq - at[i];  
        }  
        if(count == n) {  
            break;  
        }  
    }  
    for(i = 0; i < n; i++) {  
        wt[i] = tat[i] - bt[i];  
        swt += wt[i];  
        stat += tat[i];  
    }  
    awt = (float)swt / n;  
    atat = (float)stat / n;  
    printf("\nProcess No\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\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 waiting time: %f\n", awt);  
    printf("Average turn-around time: %f\n", atat);  
    return 0;**

**Sjf**

***#include<stdio.h>  
#define max 20  
int main(){  
    int bt[max], at[max], ft[max], wt[max], tat[max];  
    int i, j, x = 0, y = 0, z = 0, min = 0, temp1, temp2, k, l;  
    int n;  
    float t, u;  
    printf("Enter number of processes to be executed:\n");  
    scanf("%d", &n);  
    for(i = 1; i <= n; i++) {  
        printf("Enter the burst time for process-%d:\n", i);  
        scanf("%d", &bt[i]);  
        printf("Enter the arrival time 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] = 0;  
        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\n, bt: %d\n, ft: %d\n,wt: %d\n,tat: %d\n",  
                i, at[i], bt[i], ft[i], wt[i], tat[i]);  
    }  
    printf("\nAverage Waiting Time: %.2f", (float)z/n);  
    printf("\nAverage Turnaround Time: %.2f", (float)y/n);  
  
    return 0;  
}***

***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, temp1, temp2, temp3;***

***float avg\_wt = 0, avg\_tat = 0;***

***printf("Enter number of processes to be executed: ");***

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

***for (i = 0; i < n; i++) {***

***printf("Enter the burst time of process-%d: ", i + 1);***

***scanf("%d", &bt[i]);***

***printf("Enter the arrival time for process-%d: ", i + 1);***

***scanf("%d", &at[i]);***

***p[i] = i + 1;***

***}***

***for (i = 0; i < n - 1; i++) {***

***for (j = i + 1; j < n; j++) {***

***if (at[i] > at[j] || (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;***

***}***

***}***

***}***

***int total\_time = 0;***

***for (i = 0; i < n; i++) {***

***if (total\_time < at[i]) {***

***total\_time = at[i];***

***}***

***total\_time += bt[i];***

***ft[i] = total\_time;***

***wt[i] = ft[i] - at[i] - bt[i];***

***tat[i] = bt[i] + wt[i];***

***avg\_wt += wt[i];***

***avg\_tat += tat[i];***

***}***

***printf("\n%-10s %-12s %-10s %-15s %-12s %-14s\n", "Process", "Arrival Time", "Burst Time", "Completion Time", "Waiting Time", "Turnaround Time");***

***for (i = 0; i < n; i++) {***

***printf("%-10d %-12d %-10d %-15d %-12d %-14d\n", p[i], at[i], bt[i], ft[i], wt[i], tat[i]);***

***}***

***printf("\nAverage Waiting Time: %.2f", avg\_wt / n);***

***printf("\nAverage Turnaround Time: %.2f\n", avg\_tat / n);***

***return 0;***

***}***

7. Write a program to solve producer-consumer problem using Semaphores.

**#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <unistd.h>  
#include <signal.h>  
#define MAX 5**

**#define TOTAL 10       
int buffer[MAX];  
int in = 0, out = 0;    
sem\_t empty, full;  
pthread\_mutex\_t mutex;  
int running = 1;**

**void handle\_signal(int sig) {  
    printf("\nReceived signal %d. Cleaning up and exiting...\n", sig);  
    running = 0;**

**}  
void\* producer(void\* param) {  
    int item;  
    for (int i = 0; i < TOTAL && running; i++) {  
        item = rand() % 100;**

**sem\_wait(&empty);**

**pthread\_mutex\_lock(&mutex);  
        buffer[in] = item;**

**printf("Produced: %d at index %d\n", item, in);  
        in = (in + 1) % MAX;    
        pthread\_mutex\_unlock(&mutex);  
        sem\_post(&full);    
        sleep(1);**

**}  
    return NULL;  
}  
void\* consumer(void\* param) {  
    int item;  
    for (int i = 0; i < TOTAL && running; i++) {  
        sem\_wait(&full);**

**pthread\_mutex\_lock(&mutex);  
        item = buffer[out];**

**printf("Consumed: %d from index %d\n", item, out);  
        out = (out + 1) % MAX;   
        pthread\_mutex\_unlock(&mutex);  
        sem\_post(&empty);   
        sleep(1);**

**}  
    return NULL;  
}  
int main() {  
    srand(time(NULL));**

**signal(SIGINT, handle\_signal);    
    pthread\_t prod, cons;  
      sem\_init(&empty, 0, MAX);  
    sem\_init(&full, 0, 0);  
    pthread\_mutex\_init(&mutex, NULL);  
    pthread\_create(&prod, NULL, producer, NULL);  
    pthread\_create(&cons, NULL, consumer, NULL);  
    pthread\_join(prod, NULL);  
    pthread\_join(cons, NULL);  
    sem\_destroy(&empty);  
    sem\_destroy(&full);  
    pthread\_mutex\_destroy(&mutex);  
    printf("Program finished successfully!\n");  
    return 0;  
}**

8. Implement the following memory allocation methods for fixed partition

a) First fit b) Worst fit c) Best fit

a) First fit

***#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] -= 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");  
    }  
}  
int 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 processes: ");  
    scanf("%d", &p);  
    for (i = 0; i < p; i++) {  
        printf("Enter %d process size: ", i);  
        scanf("%d", &processSize[i]);  
    }  
    firstFit(blockSize, bs, processSize, p);  
    return 0;  
}***

c) Best fit

#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] -= 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");  
    }  
}  
int 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 processes: ");  
    scanf("%d", &p);  
    for (i = 0; i < p; i++) {  
        printf("Enter %d process size: ", i);  
        scanf("%d", &processSize[i]);  
    }  
    BestFit(blockSize, bs, processSize, p);  
    return 0;  
}

b) Worst fit

#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] -= 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");  
    }  
}  
int 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 processes: ");  
    scanf("%d", &p);  
    for (i = 0; i < p; i++) {  
        printf("Enter %d process size: ", i);  
        scanf("%d", &processSize[i]);  
    }  
    worstFit(blockSize, bs, processSize, p);  
    return 0;  
}

9. Simulate the following page replacement algorithms

a) FIFO b) LRU c) LFU

#include <stdio.h>  
int main() {  
    int i, j, n, a[50], frame[10], no, k, avail, count = 0;  
    printf("\nEnter the number of pages: ");  
    scanf("%d", &n);  
    printf("\nEnter the page numbers:\n");  
    for (i = 0; i < n; i++)  
        scanf("%d", &a[i]);  
    printf("\nEnter 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 = 0; 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;  
                break;  
            }  
        }  
                if (avail == 0) {  
            frame[j] = a[i];  
            j = (j + 1) % no;    
            count++;  
            for (k = 0; k < no; k++)  
                printf("%d\t", frame[k]);  
        }  
        printf("\n");  
    }  
    printf("\nPage fault count is: %d\n", count);  
    return 0;  
}

b) LRU

#include<stdio.h>  
int findLRU(int time[], int n) {  
    int i, min = time[0], pos = 0;  
    for (i = 1; i < n; ++i) {  
        if (time[i] < min) {  
            min = time[i];  
            pos = i;  
        }  
    }  
    return pos;  
}  
int main() {  
    int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;  
    printf("Enter number of frames: ");  
    scanf("%d", &no\_of\_frames);  
    printf("Enter number of pages: ");  
    scanf("%d", &no\_of\_pages);  
    printf("Enter reference string: ");  
    for (i = 0; i < no\_of\_pages; ++i) {  
        scanf("%d", &pages[i]);  
    }  
    for (i = 0; i < no\_of\_frames; ++i) {  
        frames[i] = -1;  
    }  
        for (i = 0; i < no\_of\_pages; ++i) {  
        flag1 = flag2 = 0;  
               for (j = 0; j < no\_of\_frames; ++j) {  
            if (frames[j] == pages[i]) {  
                counter++;  
                time[j] = counter;

             flag1 = 1;

            flag2 = 1;   
                break;  
            }  
        }  
               if (flag1 == 0) {  
                    for (j = 0; j < no\_of\_frames; ++j) {  
                if (frames[j] == -1) {  
                    counter++;  
                    faults++;  
                    frames[j] = pages[i];  
                    time[j] = counter;  
                    flag2 = 1;   
                    break;  
                }  
            }  
        }  
          if (flag2 == 0) {  
            pos = findLRU(time, no\_of\_frames);

            counter++;  
            faults++;  
            frames[pos] = pages[i];  
            time[pos] = counter;   
        }  
                printf("\n");  
        for (j = 0; j < no\_of\_frames; ++j) {  
            printf("%d\t", frames[j]);  
        }  
    }  
       printf("\n\nTotal page faults = %d\n", faults);  
    return 0;  
}

c) LfU

***#include <stdio.h>***

***int main() {***

***int total\_frames, total\_pages, hit = 0;***

***int frame[10], pages[25], arr[25] = {0}, time[25] = {0};***

***int m, n, temp, flag, k, minimum\_time;***

***printf("Enter the number of pages: ");***

***scanf("%d", &total\_pages);***

***printf("Enter total number of frames: ");***

***scanf("%d", &total\_frames);***

***for (m = 0; m < total\_frames; m++) {***

***frame[m] = -1;***

***}***

***printf("Enter values of reference string:\n");***

***for (m = 0; m < total\_pages; m++) {***

***printf("Enter value no.[%d]: ", m + 1);***

***scanf("%d", &pages[m]);***

***}***

***printf("\n");***

***for (m = 0; m < total\_pages; m++) {***

***arr[pages[m]]++;***

***time[pages[m]] = m;***

***flag = 1;***

***for (n = 0; n < total\_frames; n++) {***

***if (frame[n] == pages[m]) {***

***hit++;***

***flag = 0;***

***break;***

***} else if (frame[n] == -1) {***

***frame[n] = pages[m];***

***flag = 0;***

***break;***

***}***

***}***

***if (flag) {***

***k = frame[0];***

***for (n = 1; n < total\_frames; n++) {***

***if (arr[frame[n]] < arr[k] ||***

***(arr[frame[n]] == arr[k] && time[frame[n]] < time[k])) {***

***k = frame[n];***

***}***

***}***

***for (n = 0; n < total\_frames; n++) {***

***if (frame[n] == k) {***

***arr[frame[n]] = 0;***

***frame[n] = pages[m];***

***break;***

***}***

***}***

***}***

***for (n = 0; n < total\_frames; n++) {***

***if (frame[n] != -1)***

***printf("%d\t", frame[n]);***

***else***

***printf("-\t");***

***}***

***printf("\n");***

***}***

***printf("Page Hit:\t%d\n", hit);***

***printf("Page Fault:\t%d\n", total\_pages - hit);***

***return 0;***

***}***

11. Implement Bankers Algorithm for Dead Lock avoidance and prevention

#include<stdio.h>  
**//#include<conio.h>  
int main(){  
int available[3],work[5],max[5][3],allocation[5][3],need[5][3],safe[5],totalres[5];  
char finish[3];  
int i,j,k,totalloc=0,state,value=0;  
//clrscr();  
printf("Enter instances of each resources:\n");  
for(i=0;i<3;i++){  
    scanf("%d",&totalres[i]);  
}  
printf("Enter maximum no 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]);  
    }  
}  
printf("Enter no of resources allocated to each process:\n");  
for(i=0;i<5;i++){  
    for(j=0;j<3;j++){  
        printf("Enter the resource of %d allocated to process %d:",(j+1),i);  
        scanf("%d",&allocation[i][j]);  
    }  
}  
for(i=0;i<5;i++){  
    for(j=0;j<3;j++){  
        need[i][j]=max[i][j]-allocation[i][j];  
    }  
}  
for(i=0;i<3;i++){  
    finish[i]='f';  
}  
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];  
}  
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("Available resorces:");  
for(i=0;i<3;i++){  
    printf("%d",available[i]);  
}  
printf("\n");  
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==1){  
        for(j=0;j<3;j++){  
            work[j]=work[j]+allocation[i][j];  
        }  
        finish[i]='t';  
        safe[value]=i;  
        ++value;  
    }  
    if(i==4){  
        if(value=5){  
            break;  
        }  
        else{  
            i=-1;  
        }  
    }  
}  
printf("safe states are:\n");  
for(i=0;i<5;i++){  
    printf("p %d",safe[i]);  
}  
return 0;  
}**