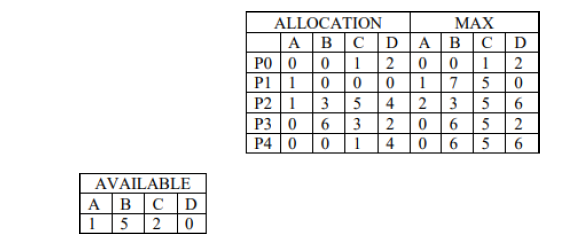
1. Write a C program to simulate Banker’s algorithm for the purpose of deadlock avoidance. Consider the following snapshot of system, A, B, C and D are the resource type.



a) Calculate and display the content of need matrix?

b) Is the system in safe state? If display the safe sequence.

c) If a request from process P arrives for (0, 4, 2, 0) can it be granted immediately by keeping the system in safe state. Print a message.

-> solution

#include<stdio.h>

#define MAX 10

int m,n,total[MAX],avail[MAX],alloc[MAX][MAX],

max[MAX][MAX],need[MAX][MAX],work[MAX],finish[MAX],

seq[MAX],request[MAX];

/\*n = number of processes m = number of resource types total[] = total resources of each type avail[] = available resources right now alloc[][] = resources allocated to each process max[][] = max resources each process may need need[][] = remaining resources each process needs work[] = work vector used during Banker's check finish[] = tracks which process has finished seq[] = stores safe sequence request[] = request vector for resource request \*/

void accept()

{

int i,j;

printf("Enter no.of process:");

scanf("%d",&n);

printf("Enter no.of resource types:");

scanf("%d",&m);

printf("Enter total no.of resources of each resource type:\n");

for(i=0;i<m;i++)

{

printf("%c:",65+i);

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

}

printf("Enter no.of allocated resources of each resource type by each process:\n");

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

{

printf("P%d:\n",i);

for(j=0;j<m;j++)

{

printf("%c:",65+j);

scanf("%d",&alloc[i][j]);

}

}

printf("Enter no.of maximum resources of each resource type by each process:\n");

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

{

printf("P%d:\n",i);

for(j=0;j<m;j++)

{

printf("%c:",65+j);

scanf("%d",&max[i][j]);

}

}

}

void calc\_avail()

{

int i,j,s;

for(j=0;j<m;j++)

{

s=0;

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

s+=alloc[i][j];

avail[j] = total[j] - s;

}

}

void calc\_need()

{

int i,j;

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

for(j=0;j<m;j++)

need[i][j]=max[i][j]-alloc[i][j];

}

void print()

{

int i,j;

printf("\tAllocation\tMax\tNeed\n\t");

for(i=0;i<3;i++)

{

for(j=0;j<m;j++)

printf("%3c",65+j);

printf("\t");

}

printf("\n");

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

{

printf("P%d\t",i);

for(j=0;j<m;j++)

printf("%3d",alloc[i][j]);

printf("\t");

for(j=0;j<m;j++)

printf("%3d",max[i][j]);

printf("\t");

for(j=0;j<m;j++)

printf("%3d",need[i][j]);

printf("\n");

}

printf("Available\n");

for(j=0;j<m;j++)

printf("%3c",65+j);

printf("\n");

for(j=0;j<m;j++)

printf("%3d",avail[j]);

printf("\n");

}

int check(int s)

{

int i,j;

i = s;

do

{

if(!finish[i])

{

for(j=0;j<m;j++)

{

if(need[i][j]>work[j])

break;

}

if(j==m) return i;

}

i=(i+1)%n;

}while(i!=s);

return -1;

}

void banker()

{

int i,j,k=0;

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

finish[i]=0;

for(j=0;j<m;j++)

work[j] = avail[j];

i=0;

while((i=check(i))!=-1)

{

printf("Process P%d resource granted.\n",i);

finish[i] = 1;

for(j=0;j<m;j++)

work[j] += alloc[i][j];

printf("finish(");

for(j=0;j<n;j++)

printf("%d,",finish[j]);

printf("\b)\nwork(");

for(j=0;j<m;j++)

printf("%d,",work[j]);

printf("\b)\n");

seq[k++]=i;

i=(i+1)%n;

}

if(k==n)

{

printf("System is in safe state.\n");

printf("Safe sequence:");

for(j=0;j<n;j++)

printf("P%d ",seq[j]);

}

else

{

printf("System is not in safe state.");

}

printf("\n");

}

int main()

{

int i,j,pno;

accept();

calc\_avail();

calc\_need();

print();

banker();

printf("Enter process no:");

scanf("%d",&pno);

printf("Enter resource request of process P%d\n",pno);

for(j=0;j<m;j++)

{

printf("%c:",65+j);

scanf("%d",&request[j]);

}

for(j=0;j<m;j++)

{

if(request[j]>need[pno][j])

break;

}

if(j==m)

{

for(j=0;j<m;j++)

{

if(request[j]>avail[j])

break;

}

if(j==m)

{

for(j=0;j<m;j++)

{

avail[j]-=request[j];

alloc[pno][j]+=request[j];

need[pno][j]-=request[j];

print();

banker();

}

}

else

printf("Process P%d must wait.\n",pno);

}

else

printf("Process P%d has exceeded its maximum claim\n",pno);

return 0;

}

2. Partially implement the Menu driven Banker's algorithm for accepting Allocation, Max from user.

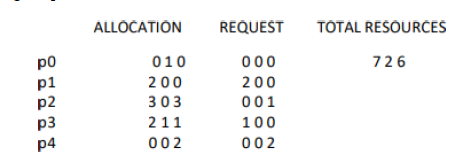
a) Accept Available

b) Display Allocation, Max

c) Find Need and display It,

d) Display Available Consider the system with 3 resources types A,B, and C with 7,2,6 instances respectively.

Consider the following snapshot:



Answer the following questions:

a) Display the contents of Available array?

b) Is there any deadlock? Print the message

-> solution

#include<stdio.h>

#define N 10 // Max processes

#define M 3 // Number of resource types fixed as A,B,C

int n; // Number of processes

int total[M] = {7, 2, 6}; // Total resources: A=7, B=2, C=6

int alloc[N][M], max[N][M], need[N][M], avail[M];

void accept()

{

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Allocation matrix (process wise):\n");

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

{

printf("P%d:\n",i);

for(int j=0; j<M; j++)

{

printf("%c: ",65+j);

scanf("%d",&alloc[i][j]);

}

}

printf("Enter Max matrix (process wise):\n");

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

{

printf("P%d:\n",i);

for(int j=0; j<M; j++)

{

printf("%c: ",65+j);

scanf("%d",&max[i][j]);

}

}

}

void calc\_need()

{

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

for(int j=0; j<M; j++)

need[i][j] = max[i][j] - alloc[i][j];

}

void calc\_avail()

{

int sum[M] = {0};

for(int j=0; j<M; j++)

{

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

sum[j] += alloc[i][j];

avail[j] = total[j] - sum[j];

}

}

void display\_alloc\_max()

{

printf("\nProcess\t Allocation\t Max\n\t");

for(int j=0; j<M; j++)

printf("%c ",65+j);

printf("\t");

for(int j=0; j<M; j++)

printf("%c ",65+j);

printf("\n");

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

{

printf("P%d\t",i);

for(int j=0; j<M; j++)

printf("%d ",alloc[i][j]);

printf("\t");

for(int j=0; j<M; j++)

printf("%d ",max[i][j]);

printf("\n");

}

}

void display\_need()

{

printf("\nNeed matrix:\nProcess\t");

for(int j=0; j<M; j++)

printf("%c ",65+j);

printf("\n");

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

{

printf("P%d\t",i);

for(int j=0; j<M; j++)

printf("%d ",need[i][j]);

printf("\n");

}

}

void display\_avail()

{

printf("\nAvailable resources:\n");

for(int j=0; j<M; j++)

printf("%c ",65+j);

printf("\n");

for(int j=0; j<M; j++)

printf("%d ",avail[j]);

printf("\n");

}

void check\_deadlock()

{

int work[M], finish[N]={0}, safe\_seq[N], count=0;

for(int j=0; j<M; j++)

work[j] = avail[j];

while(count < n)

{

int found=0;

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

{

if(!finish[i])

{

int j;

for(j=0; j<M; j++)

if(need[i][j] > work[j])

break;

if(j==M)

{

for(int k=0; k<M; k++)

work[k] += alloc[i][k];

finish[i]=1;

safe\_seq[count++] = i;

found=1;

}

}

}

if(!found) break;

}

if(count==n)

{

printf("\nNo deadlock. System is in safe state.\nSafe sequence: ");

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

printf("P%d ",safe\_seq[i]);

printf("\n");

}

else

printf("\nDeadlock detected! System is not in safe state.\n");

}

int main()

{

int ch;

do{

printf("\n--- Menu ---\n");

printf("1. Accept Allocation and Max\n");

printf("2. Display Allocation and Max\n");

printf("3. Find Need and display\n");

printf("4. Display Available\n");

printf("5. Check for deadlock\n");

printf("6. Exit\n");

printf("Enter choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: accept(); break;

case 2: display\_alloc\_max(); break;

case 3: calc\_need(); display\_need(); break;

case 4: calc\_avail(); display\_avail(); break;

case 5: calc\_need(); calc\_avail(); check\_deadlock(); break;

case 6: break;

default: printf("Invalid choice!\n");

}

}while(ch!=6);

return 0;

}

3. Write the simulation program using SJF (non-preemptive). The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times .

-> solution

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

typedef struct SJF

{

char name[20];

int at,bt,ct,tbt,wt,tat;

}SJF;

SJF\* s;

int n,currenttime,cnt;

float totalwt,totaltat,avgwt,avgtat;

void accept()

{

int i;

printf("Enter no of processes ");

scanf("%d",&n);

s=(SJF\*)malloc(sizeof(SJF)\*n);

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

{

printf("Enter process name ");

scanf("%s",&s[i].name);

printf("Enter arrival time ");

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

printf("Enter burst time ");

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

s[i].tbt=s[i].bt;

}

}

void sort()

{

SJF t;

int i,j;

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

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

if(s[i].at>s[j].at)

{

t=s[i];

s[i]=s[j];

s[j]=t;

}

}

int getindex()

{

int i,loc=-1,min=999;

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

if(s[i].tbt!=0 && s[i].at<=currenttime)

{

if(s[i].bt<min)

{

min=s[i].bt;

loc=i;

}

}

return loc;

}

void sechedule()

{

int i;

sort();

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

while(1)

{

i=getindex();

if(i==-1)

{

printf("|%d idel",currenttime);

currenttime=s[cnt].at;

printf("%d",currenttime);

}

else

{

printf("|%d %s ",currenttime,s[i].name);

s[i].wt=currenttime-s[i].at;

s[i].tat=s[i].wt+s[i].bt;

s[i].tbt=0;

currenttime+=s[i].bt;

s[i].ct=currenttime;

printf("%d",currenttime);

totalwt+=s[i].wt;

totaltat+=s[i].tat;

cnt++;

if(cnt==n)

break;

}

}

avgwt=totalwt/n;

avgtat=totaltat/n;

}

void display()

{

int i;

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

printf("\nName\tArrival Time\tBurst Time\tWaiting Time\tTurn Around Time\n");

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

printf("%s\t%d\t\t%d\t\t%d\t\t%d\n",s[i].name,s[i].at,s[i].bt,s[i].wt,s[i].tat);

printf("\nTotal waiting time %f",totalwt);

printf("\nTotal Turn Around time %f",totaltat);

printf("\nAverage waiting time %f",avgwt);

printf("\nAverage Turn Around time %f\n",avgtat);

}

int main()

{

accept();

sechedule();

display();

}

4)

Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String: 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

1. Implement FIFO

-> solution

#include <stdio.h>

#define MAX 100

int main() {

int referenceString[MAX], frames[MAX], n, refLen;

int i, j, k, pageFaults = 0, found, pos = 0;

// Input number of frames

printf("Enter number of memory frames: ");

scanf("%d", &n);

// Input length of reference string

printf("Enter the length of the reference string: ");

scanf("%d", &refLen);

// Input reference string

printf("Enter the reference string (space-separated):\n");

for(i = 0; i < refLen; i++) {

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

}

// Initialize frames to -1 (empty)

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

frames[i] = -1;

}

printf("\nPage Replacement Process (FIFO):\n");

// FIFO implementation

for(i = 0; i < refLen; i++) {

found = 0;

// Check if page is already in frame

for(j = 0; j < n; j++) {

if(frames[j] == referenceString[i]) {

found = 1;

break;

}

}

// If page not found => Page Fault

if(!found) {

frames[pos] = referenceString[i];

pos = (pos + 1) % n; // Move pointer circularly

pageFaults++;

// Display current frame status

printf("Step %2d: ", i + 1);

for(k = 0; k < n; k++) {

if(frames[k] != -1)

printf("%d ", frames[k]);

else

printf("- ");

}

printf(" <- Page Fault\n");

} else {

// Display if page already in frame

printf("Step %2d: ", i + 1);

for(k = 0; k < n; k++) {

if(frames[k] != -1)

printf("%d ", frames[k]);

else

printf("- ");

}

printf(" <- No Page Fault\n");

}

}

// Output total number of page faults

printf("\nTotal Page Faults = %d\n", pageFaults);

return 0;

}

5) Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.

Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2

1. Implement LRU

-> solution

#include <stdio.h>

#define MAX 100

int findLRU(int time[], int n) {

int i, minimum = time[0], pos = 0;

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

if(time[i] < minimum) {

minimum = time[i];

pos = i;

}

}

return pos;

}

int main() {

int frames[MAX], time[MAX];

int referenceString[MAX];

int n, refLen;

int i, j, pos, pageFaults = 0, count = 0, found;

// Input number of frames

printf("Enter number of memory frames: ");

scanf("%d", &n);

// Input reference string length

printf("Enter the length of the reference string: ");

scanf("%d", &refLen);

// Input reference string

printf("Enter the reference string:\n");

for(i = 0; i < refLen; i++) {

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

}

// Initialize frames

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

frames[i] = -1;

time[i] = 0;

}

printf("\nPage Replacement Process (LRU):\n");

for(i = 0; i < refLen; ++i) {

found = 0;

// Check if page is already in frame

for(j = 0; j < n; ++j) {

if(frames[j] == referenceString[i]) {

count++;

time[j] = count;

found = 1;

break;

}

}

// If page not found => Page Fault

if(!found) {

// If frame has empty slot

for(j = 0; j < n; ++j) {

if(frames[j] == -1) {

count++;

pageFaults++;

frames[j] = referenceString[i];

time[j] = count;

found = 1;

break;

}

}

}

// If no empty slot, replace LRU page

if(!found) {

pos = findLRU(time, n);

count++;

pageFaults++;

frames[pos] = referenceString[i];

time[pos] = count;

}

// Display frame contents

printf("Step %2d: ", i + 1);

for(j = 0; j < n; ++j) {

if(frames[j] != -1)

printf("%d ", frames[j]);

else

printf("- ");

}

if(found)

printf(" <- No Page Fault\n");

else

printf(" <- Page Fault\n");

}

printf("\nTotal Page Faults = %d\n", pageFaults);

return 0;

}

6 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames.

Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

Implement OPT

-> solution

#include <stdio.h>

#define MAX 100

// Function to find the page to replace (which is used farthest in the future)

int predict(int referenceString[], int frames[], int refLen, int index, int n) {

int i, j, farthest = index, pos = -1, found;

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

found = 0;

for(j = index; j < refLen; j++) {

if(frames[i] == referenceString[j]) {

if(j > farthest) {

farthest = j;

pos = i;

}

found = 1;

break;

}

}

// If the page is never used again

if(!found)

return i;

}

// If all pages will be used again, replace the farthest one

return (pos == -1) ? 0 : pos;

}

int main() {

int referenceString[MAX], frames[MAX];

int refLen = 0, n = 3;

int i, j, k, pageFaults = 0, found;

// Input reference string

printf("Enter the length of the reference string: ");

scanf("%d", &refLen);

printf("Enter the reference string (space-separated):\n");

for(i = 0; i < refLen; i++) {

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

}

// Initialize frames to -1 (empty)

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

frames[i] = -1;

}

printf("\nPage Replacement Process (OPT):\n");

for(i = 0; i < refLen; i++) {

found = 0;

// Check if page is already in frame

for(j = 0; j < n; j++) {

if(frames[j] == referenceString[i]) {

found = 1;

break;

}

}

// Page Fault

if(!found) {

int pos = -1;

// If there is empty frame

for(j = 0; j < n; j++) {

if(frames[j] == -1) {

pos = j;

break;

}

}

// If no empty frame, find optimal page to replace

if(pos == -1)

pos = predict(referenceString, frames, refLen, i + 1, n);

frames[pos] = referenceString[i];

pageFaults++;

}

// Display current frame status

printf("Step %2d: ", i + 1);

for(k = 0; k < n; k++) {

if(frames[k] != -1)

printf("%d ", frames[k]);

else

printf("- ");

}

if(found)

printf(" <- No Page Fault\n");

else

printf(" <- Page Fault\n");

}

printf("\nTotal Page Faults = %d\n", pageFaults);

return 0;

}