System call fork

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

main()

{

pid\_t pid;

int x = 5;

pid = fork();

x++;

if (pid < 0)

{

printf("Process creation error");

exit(-1);

}

else if (pid == 0)

{

printf("Child process:");

printf("\nProcess id is %d", getpid());

printf("\nValue of x is %d", x);

printf("\nProcess id of parent is %d\n", getppid());

}

else

{

printf("\nParent process:");

printf("\nProcess id is %d", getpid());

printf("\nValue of x is %d", x);

printf("\nProcess id of shell is %d\n", getppid());

}

}

Wait

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

main()

{

int i, status;

pid\_t pid;

pid = fork();

if (pid < 0)

{

printf("\nProcess creation failure\n"); exit(-1);

}

else if(pid > 0)

{

wait(NULL);

printf ("\nParent starts\nEven Nos: "); for (i=2;i<=10;i+=2)

printf ("%3d",i);

printf ("\nParent ends\n");

}

else if (pid == 0)

{

printf ("Child starts\nOdd Nos: ");

for (i=1;i<10;i+=2)

printf ("%3d",i);

printf ("\nChild ends\n");

}

}

Fcfs

#include <stdio.h>

int main()

{

int n, bt[20], wt[20], tat[20], avwt = 0, avtat = 0, i, j;

printf("Enter total number of processes (maximum 20): ");

scanf("%d", &n);

printf("Enter Process Burst Time\n");

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

{

printf("P[%d]: ", i + 1);

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

}

wt[0] = 0;

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

{

wt[i] = 0;

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

{

wt[i] += bt[j];

}

}

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

{

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

avwt += wt[i];

avtat += tat[i];

printf("P[%d]\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);

}

avwt /= n;

avtat /= n;

printf("\nAverage Waiting Time: %d\n", avwt);

printf("Average Turnaround Time: %d\n", avtat);

return 0;

}

SJF

#include <stdio.h>

int main()

{

int i, j, n, p[5], wt[5], bt[5], tat[5];

int te, t;

float twt = 0, ttat = 0, awt, atat;

printf("Enter the number of processes: \t");

scanf("%d", &n);

printf("Enter the process numbers:\n");

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

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

}

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

printf("Enter the burst time of process %d:\t", p[i]);

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

}

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

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

if (bt[i] < bt[j]) {

t = bt[i];

bt[i] = bt[j];

bt[j] = t;

te = p[i];

p[i] = p[j];

p[j] = te;

}

}

}

wt[0] = 0;

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

wt[i] = wt[i - 1] + bt[i - 1];

}

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

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

}

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

twt += wt[i];

ttat += tat[i];

}

awt = twt / n;

atat = ttat / n;

printf("\nProcess\tWaiting Time\tTurnaround Time\n");

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

printf("P%d\t%d\t\t%d\n", p[i], wt[i], tat[i]);

}

printf("The average waiting time is %f\n", awt);

printf("The average turnaround time is %f\n", atat);

return 0;

}

Round robin

#include <stdio.h>

void roundRobin(int bt[], int n, int quantum) {

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

int rem\_bt[n];

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

rem\_bt[i] = bt[i];

}

int t = 0;

int done;

do {

done = 1;

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

if (rem\_bt[i] > 0) {

done = 0;

if (rem\_bt[i] > quantum) {

t += quantum;

rem\_bt[i] -= quantum;

} else {

t += rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

} while (!done);

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

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

total\_wt += wt[i];

total\_tat += tat[i];

}

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);

}

printf("Average waiting time: %.2f\n", (float)total\_wt / n);

printf("Average turnaround time: %.2f\n", (float)total\_tat / n);

}

int main() {

int bt[] = {10, 5, 8, 12};

int n = sizeof(bt) / sizeof(bt[0]);

int quantum = 2;

roundRobin(bt, n, quantum);

return 0;

}

Priority

#include<stdio.h>

int main()

{

int i,j=0,b,n,p[5],wt[5],bt[5],pr[5],tat[5];

int temp,te,t;

float twt,ttat,awt,atat;

printf("Enter the no of processes:\t");

scanf("%d",&n);

printf("Enter the process numbers:\n");

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

{

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

}

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

{

printf("Enter the priority of %d process\t",i);

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

printf("Enter the burst time of %d process\t",i);

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

}

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

{

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

{

if(pr[i]<pr[j])

{

temp=pr[i];

pr[i]=pr[j];

pr[j]=temp;

t=bt[i];

bt[i]=bt[j];

bt[j]=t;

te=p[i];

p[i]=p[j];

p[j]=te;

}

}

}

wt[-1]=0;

bt[-1]=0;

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

{

wt[i]=wt[i-1]+bt[i-1];

}

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

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

twt=0;

ttat=0;

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

{

twt=twt+wt[i];

ttat=ttat+tat[i];

}

atat=ttat/n;

awt=twt/n;

printf("\nProcess\twt\t tat\n");

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

{

printf("\nP%d\t%d\t%d\n",p[i],wt[i],tat[i]); }

printf("The average wt is %f\n",awt);

printf("The average tat is %f\n",atat);

return 1;

}

IPC Pipes

#include<stdio.h>

#include<string.h>

int main()

{

 int fd[2],child;

 char a[10];

printf("Enter the string to enter into the pipe:");

scanf("%s",a);

pipe(fd);

child=fork();

if(!child)

{

 close(fd[0]);

 write(fd[1],a,strlen(a));

 wait(0);

}

else

{

 close(fd[1]);

 read(fd[0],a,10);

 printf("\n The String retrieved from the pipe is: %s\n",a);

}

return 0;

}

SHARED MEMORY

#include<stdio.h>

#include<sys/shm.h>

#include<sys/ipc.h>

#define size 32

int main()

{ int shmid; char \*s[100],\*str;

printf("\nipc message passing using shared memory sender");

shmid=shmget(60,size,IPC\_CREAT|0666);

str=shmat(shmid,0,0);

printf("\nenter the message to be sent");

gets(s);

strcpy(str,s);

printf("\nyour mesage has been sent");

return 0; }

**RECEIVER**

#include<stdio.h>

#include<sys/shm.h>

#include<sys/ipc.h>

#define size 32

int main()

{

printf("\nipc message passing using shared memory-receiver");

int shmid;

char \*str;

shmid=shmget(60,size,IPC\_CREAT|0666);

str=shmat(shmid,0,0);

printf("\nreceived message is....");

puts(str);

return 0; }

Semaphore

#include<stdio.h>

void main()

{

int buffer[10], bufsize, in, out, produce, consume, choice=0;

in = 0;

out = 0;

bufsize = 10;

while(choice !=3)

{

printf(“\n1. Produce \t 2. Consume \t3. Exit”);

printf(“\nEnter your choice: ”);

scanf(“%d”, &choice);

switch(choice) {

case 1: if((in+1)%bufsize==out)

printf(“\nBuffer is Full”);

else

{

printf(“\nEnter the value: “);

scanf(“%d”, &produce);

buffer[in] = produce;

in = (in+1)%bufsize;

}

Break;

case 2: if(in == out)

printf(“\nBuffer is Empty”);

else

{

consume = buffer[out];

printf(“\nThe consumed value is %d”, consume);

out = (out+1)%bufsize;

}

break;

} } }

Deadlock Avoidance

#include<stdio.h>

#include<stdlib.h>

int alloc[10][10];

int max[10][10];

int cneed[10][10];

int a[10];

int total[10];

void main()

{

int r,p,i,j,k,flag,f,count,x,flag1;

count=0;

flag1=0;

printf("\nEnter number of resources:");

scanf("%d",&r);

f=r;

printf("\nEnter total memory of given resources sequentially:");

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

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

printf("\nEnter number of processes:");

scanf("%d",&p);

printf("\nEnter Allocated and Maxneed memory…"); for(i=0;i<p;i++)

{

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

{

printf("\nFor Process %d=",i);

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

//calculating current need

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

}

}

printf("\nSequence of execution is…\n"); k=0;

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

{

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

{

a[k]+=alloc[i][j];

}

a[k]=total[k]-a[k];

k++;

}

while(count!=p)

{

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

{

flag=0;

if(cneed[i][f]!=1)

{

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

{

total[j]=a[j]-cneed[i][j];

}

for(k=0;k<r;k++)

{

if(total[k]<0)

{

flag=1;

}

}

}

if((flag==0)&&(cneed[i][f]!=1))

break;

}

x=i;

cneed[x][f]=1;

printf("P%d->",x);

count++;

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

{

 total[i]+=max[x][i];

a[i]=total[i];

}

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

{

if((cneed[i][0]<a[0])&&(cneed[i][f]==0))

flag1=1;

}

if(flag1==0)

break;

}

if(flag1==0)

printf("\nUnsafe…");

else

printf("\nSafe…");

getchar();

}

Deadloack detection

#include <stdio.h>

int current[5][5], maximum\_claim[5][5], available[5];

int allocation[5] = {0, 0, 0, 0, 0};

int maxres[5], running[5], safe = 0;

int counter = 0, i, j, exec, resources, processes, k = 1;

int main() {

printf("\nEnter number of processes: ");

scanf("%d", &processes);

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

running[i] = 1;

counter++;

}

printf("\nEnter number of resources: ");

scanf("%d", &resources);

printf("\nEnter Claim Vector: ");

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

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

}

printf("\nEnter Allocated Resource Table:\n");

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

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

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

}

}

printf("\nEnter Maximum Claim Table:\n");

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

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

scanf("%d", &maximum\_claim[i][j]);

}

}

printf("\nThe Claim Vector is: ");

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

printf("\t%d", maxres[i]);

}

printf("\nThe Allocated Resource Table:\n");

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

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

printf("\t%d", current[i][j]);

}

printf("\n");

}

printf("\nThe Maximum Claim Table:\n");

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

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

printf("\t%d", maximum\_claim[i][j]);

}

printf("\n");

}

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

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

allocation[j] += current[i][j];

}

}

printf("\nAllocated resources:");

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

printf("\t%d", allocation[i]);

}

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

available[i] = maxres[i] - allocation[i];

}

printf("\nAvailable resources:");

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

printf("\t%d", available[i]);

}

printf("\n");

while (counter != 0) {

safe = 0;

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

if (running[i]) {

exec = 1;

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

if (maximum\_claim[i][j] - current[i][j] > available[j]) {

exec = 0;

break;

}

}

if (exec) {

printf("\nProcess %d is executing\n", i + 1);

running[i] = 0;

counter--;

safe = 1;

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

available[j] += current[i][j];

}

break;

}

}

}

if (!safe) {

printf("\nThe processes are in unsafe state.\n");

break;

} else {

printf("\nThe process is in safe state");

printf("\nAvailable vector:");

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

printf("\t%d", available[i]);

}

printf("\n");

}

}

return 0;

}

Threading

#include <stdio.h>

#include <string.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

pthread\_t tid[2];

void\* doSomeThing(void \*arg) {

unsigned long i = 0;

pthread\_t id = pthread\_self();

if (pthread\_equal(id, tid[0])) {

printf("\nFirst thread processing\n");

} else {

printf("\nSecond thread processing\n");

}

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

return NULL;

}

int main(void) {

int i = 0;

int err;

while (i < 2) {

err = pthread\_create(&(tid[i]), NULL, &doSomeThing, NULL);

if (err != 0) {

printf("\nCan't create thread: [%s]", strerror(err));

} else {

printf("\nThread created successfully\n");

}

i++;

}

sleep(5);

return 0;

}

Paging technique

#include<stdio.h>

#define MAX 50

int main()

{

int page[MAX],i,n,f,ps,off,pno;

int choice=0;

printf("\nEnter the no of peges in memory: ");

scanf("%d",&n);

printf("\nEnter page size: ");

scanf("%d",&ps);

printf("\nEnter no of frames: ");

scanf("%d",&f);

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

page[i]=-1;

printf("\nEnter the page table\n");

printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");

printf("\npageno\tframeno\n-------\t-------");

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

{

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

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

}

do

{

printf("\n\nEnter the logical address(i.e,page no & offset):");

scanf("%d%d",&pno,&off);

if(page[pno]==-1)

printf("\n\nThe required page is not available in any of frames");

else

printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off); printf("\nDo you want to continue(1/0)?:");

scanf("%d",&choice);

}while(choice==1);

return 1;

}

Memory allocation – first—best—worst

#include <stdio.h>

int main() {

int ch, n, fs, i, j, k, l, z, z1, z2, t1, bt[5], p[5], sfs[5], t[5], t2[5];

printf("Enter the number of processes:\t");

scanf("%d", &n);

printf("Enter the process numbers and their sizes:\n");

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

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

}

printf("\nEnter the number of free partitions:\t");

scanf("%d", &fs);

printf("\nEnter the size of each free partition:\n");

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

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

}

printf("Enter your choice:\n1. First Fit\n2. Best Fit\n3. Worst Fit\n");

scanf("%d", &ch);

printf("\nProcessNo\tPro.Size\tAll.Seg.No\n");

switch (ch) {

case 1:

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

for (j = 1; j <= fs; j++) {

if (bt[i] <= sfs[j]) {

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

sfs[j] -= bt[i];

break;

}

}

}

break;

case 2:

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

for (j = 1; j <= fs; j++) t[j] = sfs[j] - bt[i];

for (j = 1; j <= fs; j++) t2[j] = t[j];

for (k = 1; k < fs; k++)

for (l = k + 1; l <= fs; l++)

if (t2[k] > t2[l]) { t1 = t2[k]; t2[k] = t2[l]; t2[l] = t1; }

for (j = 1, z = 0; j <= fs; j++)

if (t2[j] >= 0) { z1 = t2[j]; break; }

for (k = 1, z2 = 0; k <= fs; k++)

if (z1 == t[k] && !z2++) {

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

sfs[k] -= bt[i];

break;

}

}

break;

case 3:

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

for (j = 1; j <= fs; j++) t[j] = sfs[j] - bt[i];

for (j = 1; j <= fs; j++) t2[j] = t[j];

for (k = 1; k < fs; k++)

for (l = k + 1; l <= fs; l++)

if (t2[k] < t2[l]) { t1 = t2[k]; t2[k] = t2[l]; t2[l] = t1; }

for (j = 1, z = 0; j <= fs; j++)

if (t2[j] >= 0) { z1 = t2[j]; break; }

for (k = 1, z2 = 0; k <= fs; k++)

if (z1 == t[k] && !z2++) {

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

sfs[k] -= bt[i];

break;

}

}

break;

default:

printf("Enter a correct choice between 1 and 3\n");

}

return 0;

}

Page replacement \_ fifo

#include <stdio.h>

int main() {

int i, j, n, a[50], frame[10], no, k, avail, count = 0;

printf("\nENTER THE NUMBER OF PAGES:\n");

scanf("%d", &n);

printf("\nENTER THE PAGE NUMBER :\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\t page frames\n");

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

printf("%d\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++;

for(k = 0; k < no; k++)

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

}

printf("\n");

}

printf("Page Fault Is %d\n", count);

return 0;

}

Lru

#include <stdio.h>

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 no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10];

int 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]);

}

// Initialize frames and time arrays

for (i = 0; i < no\_of\_frames; ++i) {

frames[i] = -1;

}

// Process each page

for (i = 0; i < no\_of\_pages; ++i) {

flag1 = flag2 = 0;

// Check if the page is already in one of the frames

for (j = 0; j < no\_of\_frames; ++j) {

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

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

// If page is not in any of the frames

if (flag1 == 0) {

// Check for empty frame

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 there are no empty frames, use LRU replacement

if (flag2 == 0) {

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

// Print the current state of frames

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;

}

LFU

#include <stdio.h>

int main() {

int frames[10], temp[10], pages[10];

int total\_pages, m, n, position, k, l, total\_frames;

int a = 0, b = 0, page\_fault = 0;

printf("\nEnter Total Number of Frames:\t");

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

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

frames[m] = -1;

}

printf("Enter Total Number of Pages:\t");

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

printf("Enter Values for Reference String:\n");

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

printf("Value No.[%d]:\t", m + 1);

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

}

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

a = 0;

b = 0;

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

if (frames[m] == pages[n]) {

a = 1;

b = 1;

break;

}

}

if (a == 0) {

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

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

frames[m] = pages[n];

b = 1;

break;

}

}

}

if (b == 0) {

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

temp[m] = 0;

}

for (k = n + 1, l = 1; l <= total\_frames - 1; l++, k++) {

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

if (frames[m] == pages[k]) {

temp[m] = 1;

}

}

}

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

if (temp[m] == 0) {

position = m;

break;

}

}

frames[position] = pages[n];

page\_fault++;

}

printf("\n");

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

printf("%d\t", frames[m]);

}

}

printf("\nTotal Number of Page Faults:\t%d\n", page\_fault);

return 0;

}

Sequential file allocation

#include <stdio.h>

#include <stdlib.h>

void recurse(int files[]) {

int flag = 0, startBlock, len, j, k, ch;

printf("Enter the starting block and the length of the files: ");

scanf("%d%d", &startBlock, &len);

for (j = startBlock; j < (startBlock + len); j++) {

if (files[j] == 0)

flag++;

}

if (len == flag) {

for (k = startBlock; k < (startBlock + len); k++) {

if (files[k] == 0) {

files[k] = 1;

printf("%d\t%d\n", k, files[k]);

}

}

if (k == (startBlock + len))

printf("The file is allocated to the disk\n");

} else {

printf("The file is not allocated to the disk\n");

}

printf("Do you want to enter more files?\n");

printf("Press 1 for YES, 0 for NO: ");

scanf("%d", &ch);

if (ch == 1)

recurse(files);

else

exit(0);

return;

}

int main() {

int files[50];

int i;

for(i = 0; i < 50; i++)

files[i] = 0;

printf("Files Allocated are:\n");

recurse(files);

return 0;

}

Linked file allocation

#include <stdio.h>

#include <stdlib.h> // Include the necessary header for the system function

#include <string.h>

struct fileTable {

    char name[20];

    int sb, nob;

} ft[30];

void clearScreen() {

#ifdef \_WIN32

    system("cls");

#else

    system("clear");

#endif

}

int main() {

    int i, j, n;

    char s[20];

    clearScreen();

    printf("Enter number of files: ");

    scanf("%d", &n);

    for (i = 0; i < n; i++) {

        printf("\nEnter file name %d: ", i + 1);

        scanf("%s", ft[i].name);

        printf("Enter starting block of file %d: ", i + 1);

        scanf("%d", &ft[i].sb);

        printf("Enter number of blocks in file %d: ", i + 1);

        scanf("%d", &ft[i].nob);

    }

    printf("\nEnter the file name to be searched: ");

    scanf("%s", s);

    for (i = 0; i < n; i++) {

        if (strcmp(s, ft[i].name) == 0) {

            break;

        }

    }

    if (i == n) {

        printf("\nFile Not Found\n");

    } else {

        printf("\nFILE NAME\tSTART BLOCK\tNO OF BLOCKS\tBLOCKS OCCUPIED\n");

        printf("%s\t\t%d\t\t%d\t\t", ft[i].name, ft[i].sb, ft[i].nob);

        for (j = 0; j < ft[i].nob; j++) {

            printf("%d", ft[i].sb + j);

            if (j < ft[i].nob - 1) {

                printf(", ");

            }

        }

        printf("\n");

    }

    printf("Press Enter to continue...");

    getchar();

    getchar();

    return 0;

}

Indexed file allocation

#include <stdio.h>

#include <stdlib.h>

void main() {

    int f[50], index[50], i, n, st, len, j, c, k, ind, count = 0;

    for (i = 0; i < 50; i++)

        f[i] = 0;

x:

    printf("Enter the index block: ");

    scanf("%d", &ind);

    if (f[ind] != 1) {

        printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);

        scanf("%d", &n);

    } else {

        printf("%d index is already allocated \n", ind);

        goto x;

    }

y:

    count = 0;

    for (i = 0; i < n; i++) {

        scanf("%d", &index[i]);

        if (f[index[i]] == 0)

            count++;

    }

    if (count == n) {

        for (j = 0; j < n; j++)

            f[index[j]] = 1;

        printf("Allocated\n");

        printf("File Indexed\n");

        for (k = 0; k < n; k++)

            printf("%d-------->%d : %d\n", ind, index[k], f[index[k]]);

    } else {

        printf("File in the index is already allocated \n");

        printf("Enter another file indexed");

        goto y;

    }

    printf("Do you want to enter more file (Yes - 1/No - 0)");

    scanf("%d", &c);

    if (c == 1)

        goto x;

    else

        exit(0);

}