**SYSTEM SOFTWARE LAB RECORD**

**Done by:**

Srividya Krishnakumar

CS5A

55

**TABLE OF CONTENTS:**

|  |  |  |
| --- | --- | --- |
| **Sl. No** | **Title** | **Page No.** |
| 1 | CPU Scheduling | 4 |
| 2 | File Organization | 10 |
| 3 | Banker’s Algorithm | 21 |
| 4 | Disk Scheduling | 24 |
| 5 | Producer-Consumer Problem | 27 |
| 6 | Dining Philosopher’s Problem | 30 |
| 7 | Pass 1 of a Two-Pass Assembler | 34 |
| 8 | Pass 2 of a Two-Pass Assembler | 36 |
| 9 | Single Pass Assembler | 38 |
| 10 | Two-Pass Macro Processor | 41 |
| 11 | Absolute Loader | 44 |
| 12 | Symbol Table with Hashing | 45 |

**PART A**

1. **Simulate the following non-preemptive CPU scheduling algorithms to find turnaround and waiting time: (a) FCFS (b) SJF (c) RR (preemptive) (d) Priority**

**// FCFS**

#include<stdio.h>

void findWaitingTime(int processes[],int n,int bt[],int wt[],int at[])

{

int service\_time[n];

service\_time[0]=0;

wt[0]=0;

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

{

service\_time[i]=service\_time[i-1]+bt[i-1];

wt[i]=service\_time[i]-at[i];

if (wt[i]<0)

{

wt[i]=0;

}

}

}

void findTurnAroundTime(int processes[],int n,int bt[],int wt[],int tat[])

{

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

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

}

void findavgTime(int processes[],int n,int bt[],int at[])

{

int wt[n],tat[n];

findWaitingTime(processes,n,bt,wt,at);

findTurnAroundTime(processes,n,bt,wt,tat);

printf("\nProcesses Burst Time Arrival Time Waiting Time Turn-Around Time Completion Time");

int total\_wt=0,total\_tat=0;

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

{

total\_wt=total\_wt+wt[i];

total\_tat=total\_tat+tat[i];

int compl\_time=tat[i]+at[i];

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

}

printf("\nAverage waiting time=%f",(float)total\_wt/(float)n);

printf("\nAverage turn around time=%f",(float)total\_tat/(float)n);

}

void main()

{

int processes[20],n,burst\_time[20],arrival\_time[20],i;

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

scanf("%d",&n);

printf("\nEnter Burst Times(in ms) of Processes:");

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

{

printf("\nProcess %d:",i+1);

scanf("%d",&burst\_time[i]);

}

printf("\nEnter Arrival Times(in ms) of Processes:");

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

{

printf("\nProcess %d:",i+1);

scanf("%d",&arrival\_time[i]);

}

findavgTime(processes,n,burst\_time,arrival\_time);

}

**// SJF**

#include<stdio.h>

void main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total\_tat=0,total\_wt=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter the number of process:");

scanf("%d",&n);

printf("\nEnter Burst Times(in ms) for Processes:");

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

{

printf("\nProcess %d:",i+1);

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

p[i]=i+1;

}

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total\_wt+=wt[i];

}

avg\_wt=(float)total\_wt/n;

printf("\nProcess Burst Time Waiting Time Turn Around Time");

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

{

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

total\_tat+=tat[i];

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

}

avg\_tat=(float)total\_tat/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

**// Round Robin (pre-emptive)**

#include<stdio.h>

void main()

{

int count,j,n,time,remain,flag=0,time\_quantum,wait\_time=0,turnaround\_time=0,at[20],bt[20],rt[20];

printf("Enter the number of Processes:");

scanf("%d",&n);

remain=n;

printf("\nEnter Arrival Times(in ms) for Processes:");

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

{

printf("\nProcess %d:",count+1);

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

}

printf("\nEnter Burst Times(in ms) for Processes:");

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

{

printf("\nProcess %d:",count+1);

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

rt[count]=bt[count];

}

printf("\nEnter the Time Quantum:");

scanf("%d",&time\_quantum);

printf("\nProcesses Turn Around Time Waiting Time");

for(time=0,count=0;remain!=0;)

{

if ((rt[count]<=time\_quantum)&&(rt[count]>0))

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if (rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if ((rt[count]==0)&&(flag==1))

{

remain--;

printf("\n%d\t\t%d\t\t%d",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if (count==n-1)

{

count=0;

}

else if(at[count+1]<=time)

{

count++;

}

else

{

count=0;

}

}

printf("\nAverage Waiting Time=%f",(float)(wait\_time\*1.0/n));

printf("\nAverage Turn Around Time:%f",(float)(turnaround\_time\*1.0/n));

}

**// Priority**

#include<stdio.h>

void main()

{

int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total\_wt=0, total\_tat=0, pos, temp;

float avg\_wt, avg\_tat;

printf("Enter the number of Processes:");

scanf("%d", &n);

printf("\nEnter the Burst Times(in ms) for Processes:");

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

printf("\nProcess %d:",i+1);

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

p[i]=i+1;

}

printf("\nEnter the Priority values for Processes:");

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

printf("\nProcess %d:",i+1);

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

}

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

pos=i;

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

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

pos=j;

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

wt[i]=0;

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

wt[i]+=bt[j];

total\_wt+=wt[i];

}

avg\_wt=total\_wt/n;

printf("\nProcesses Burst Time Waiting Time Turn Around Time");

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

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

total\_tat+=tat[i];

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

}

avg\_tat=total\_tat/n;

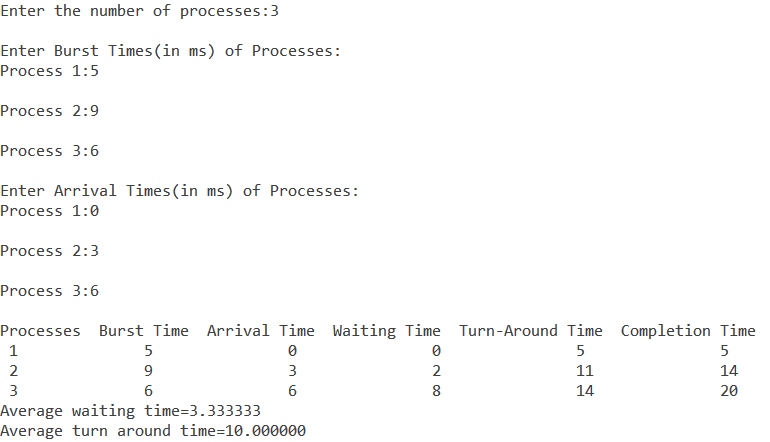
printf("\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turn Around Time=%f",avg\_tat);

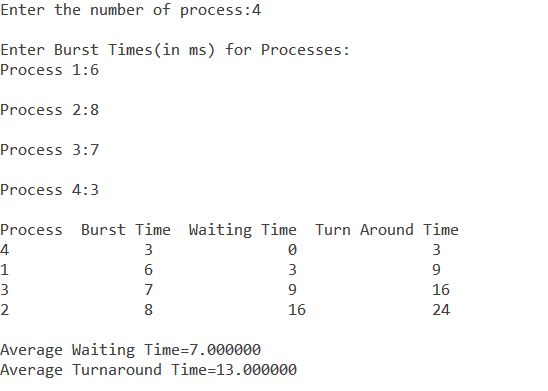
}

**Output:**

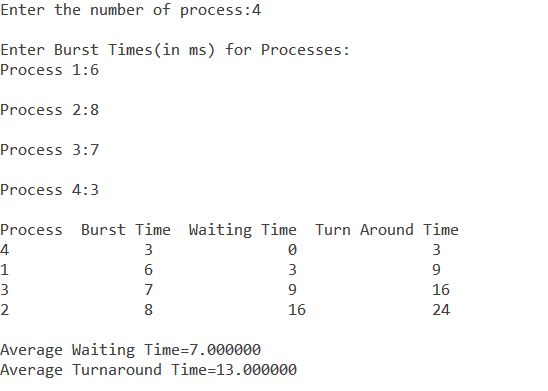
1. FCFS



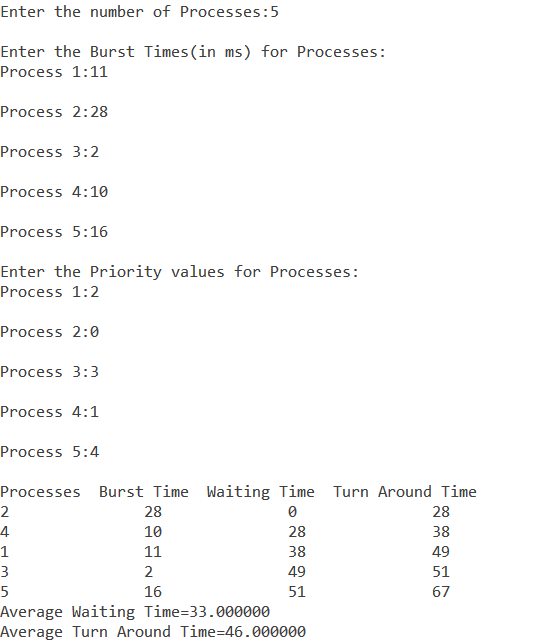
1. SJF



1. Round-Robin (pre-emptive)



1. Priority



1. **Simulate the following file organization techniques: (i) Single-level directory (ii) Two-level directory (iii) Hierarchical directory**

***// Single-level directory***

*#include*<stdio.h>

*#include*<stdlib.h>

*#include*<string.h>

void main()

{

    int i, fCount=0, ch;

    char dName[10], fName[10][10], name[10];

    printf("Enter the directory name: ");

    scanf("%s", dName);

*while* (1) {

        printf("\n1. Create file. \n2. Delete file. \n

3. Search file. \n4. Display files. \n5. Exit. \nENTER CHOICE: ");

        scanf("%d", &ch);

*switch* (ch)

        {

*case* 1: *// Create file*

*if* (fCount < 10) {

                printf("Enter filename: ");

                scanf("%s", name);

*for* (i=0; i<fCount; i++) {

*if* (!strcmp(name, fName[i]))

*break*;

                }

*if* (i==fCount) {

                    strcpy(fName[fCount++], name);

                    printf("File created\n");

                } *else* {

                    printf("File %s already exists!\n", name);

                }

            } *else* {

                printf("Directory full!\n");

            }

*break*;

*case* 2: *// Delete file*

*if* (fCount) {

                printf("Enter the name of the file: ");

                scanf("%s", name);

*for* (i = 0; i < fCount; i++) {

*if* (!strcmp(name, fName[i])) {

                        printf("Deleting file %s\n", name);

                        strcpy(fName[i], fName[--fCount]);

*break*;

                    }

                }

*if* (i == fCount)

                    printf("File %s not found!\n", name);

            } *else* {

                printf("Directory empty!\n");

            }

*break*;

*case* 3: *// Search file*

            printf("Enter the name of the file: ");

            scanf("%s", name);

*for* (i = 0; i < fCount; i++) {

*if* (!strcmp(name, fName[i])) {

                    printf("File %s found!\n", name);

*break*;

                }

            }

*if* (i == fCount)

                printf("File %s not found!\n", name);

*break*;

*case* 4: *// Display files*

            printf("\nFiles in directory %s: \n", dName);

*for* (i = 0; i < fCount; i++)

                printf("%s\n", fName[i]);

*break*;

*default*:

            exit(0);

        }

    }

}

***// Two-level directory.***

*#include*<stdio.h>

*#include*<stdlib.h>

*#include*<string.h>

typedef struct {

    char dName[10], fName[10][10];

    int fCount; *// no. of files*

} directory;

void main()

{

    directory dir[10];

    int i, ch, dCount=0, k;

    char f[30], d[30];

*while* (1)

    {

        printf("\n1. Create Directory. \n2. Create File. \n3. Delete File. \n4. Search file. \n5. Display. \n6. Exit. \nENTER CHOICE: ");

        scanf("%d", &ch);

*switch* (ch)

        {

*case* 1: *// create directory*

            printf("Enter the name of the directory: ");

            scanf("%s", dir[dCount].dName);

            dir[dCount].fCount = 0;

            dCount++;

            printf("Directory created\n");

*break*;

*case* 2:  *// create file*

            printf("Enter the name of the directory: ");

            scanf("%s", d);

*for* (i=0; i<dCount; i++) {

*if* (!strcmp(d, dir[i].dName)) {

                    printf("Enter the name of the file: ");

                    scanf("%s", f);

*for* (k=0; k<dir[i].fCount; k++) {

*if* (!strcmp(f, dir[i].fName[k]))

*break*;

                    }

*if* (k==dir[i].fCount) {

                        strcpy(dir[i].fName[dir[i].fCount++], f);

                        printf("File created\n");

                    } *else* {

                        printf("File %s already exists!\n\n", f);

                    }

*break*;

                }

            }

*if* (i==dCount)

                printf("Directory %s not found!\n", d);

*break*;

*case* 3: *// delete file*

            printf("Enter the name of the directory: ");

            scanf("%s", d);

*for* (i = 0; i < dCount; i++) {

*if* (!strcmp(d, dir[i].dName)) {

*if* (dir[i].fCount) {

                        printf("Enter the name of the file: ");

                        scanf("%s", f);

*for* (k = 0; k < dir[i].fCount; k++) {

*if* (!strcmp(f, dir[i].fName[k])) {

                                printf("Deleted file: %s\n", f);

                                dir[i].fCount--;

                                strcpy(dir[i].fName[k], dir[i].fName[dir[i].fCount]);

*goto* jmp;

                            }

                        }

                        printf("File %s not found!\n", f);

*goto* jmp;

                    } *else* {

                        printf("Directory empty!\n");

*goto* jmp;

                    }

                }

            }

            printf("Directory %s not found!\n", d);

            jmp: *break*;

*case* 4: *// search*

            printf("Enter directory name: ");

            scanf("%s", d);

*for* (i=0; i<dCount; i++) {

*if* (!strcmp(d, dir[i].dName)) {

*if* (dir[i].fCount) {

                        printf("Enter name of the file: ");

                        scanf("%s", f);

*for* (k=0; k<dir[i].fCount; k++) {

*if* (!strcmp(f, dir[i].fName[k])) {

                                printf("File %s found in directory: %s\n", f, dir[i].dName);

*goto* jmps;

                            }

                        }

                        printf("File %s not found!\n", f);

*goto* jmps;

                    } *else* {

                        printf("Directory empty!");

*goto* jmps;

                    }

                }

            }

            printf("Directory %s not found!\n", d);

            jmps: *break*;

*case* 5: *// display*

*if* (!dCount)

                printf("No directories!\n");

*else* {

*for* (i=0; i<dCount; i++) {

                    printf("DIRECTORY: %s\n", dir[i].dName);

*if* (dir[i].fCount) {

*for* (k=0; k<dir[i].fCount; k++)

                            printf("%s\n", dir[i].fName[k]);

                        printf("\n");

                    } *else* {

                        printf("Empty!\n\n");

                    }

                }

            }

*break*;

*default*:

            exit(0);

        }

    }

}

***// Heirarchical Directory***

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <stdbool.h>

*#include* <string.h>

struct node {

    char name[128];

    bool isDir;

    struct node \*p; *// parent*

    struct node \*c[100]; *// children*

    int i; *// no of children*

} \* head, \*curr;

void ls() {

    int i;

*if* (!curr->i) {

        printf("Directory Empty!\n");

*return*;

    }

*for* (i = 0; i < curr->i; i++) {

*if* (curr->c[i]->isDir)

            printf("\*%s\*  ", curr->c[i]->name);

*else*

            printf("%s  ", curr->c[i]->name);

    }

    printf("\n");

}

void touch(bool d) {

    char \*type = d ? "directory" : "file";

    printf("Enter %s name: ", type);

    char fname[128];

    scanf("%s", fname);

    struct node \*temp = (struct node \*)malloc(sizeof(struct node));

    strcpy(temp->name, fname);

    temp->isDir = d;

    temp->p = curr;

    curr->c[curr->i] = temp;

    curr->i += 1; *// increment the no. of children*

}

void cd() *// relative path - from current directory*

{

    int i;

    printf("Enter directory name: ");

    char dname[128];

    scanf("%s", dname);

*for* (i = 0; i < curr->i; i++) {

*if* (!strcmp(curr->c[i]->name, dname) && curr->c[i]->isDir) {

            curr = curr->c[i];

            printf("Changed directory to: %s. \n", curr->name);

*return*;

        }

    }

    printf("Directory not present.\n");

}

void cdup() {

*if* (curr->p == NULL) {

        printf("You are at the root directory\n");

*return*;

    }

    curr = curr->p;

    printf("Changed directory to: %s. \n", curr->name);

}

void rm(bool d) {

    char \*type = d ? "directory" : "file";

    printf("Enter name of %s to delete: ", type);

    char name[128];

    scanf("%s", name);

    int i;

*for* (i = 0; i < curr->i; i++) {

*if* (!strcmp(curr->c[i]->name, name) && ((d && curr->c[i]->isDir) || (!d && curr->c[i]->isDir == false))) {

            int t = i;

*while* (t < (curr->i) - 1) {

                curr->c[t] = curr->c[t + 1];

                t++;

            }

            curr->i -= 1;

            printf("Successfully deleted.\n");

*return*;

        }

    }

    printf("Not found\n");

}

void main() {

    int in;

    head = (struct node \*)malloc(sizeof(struct node));

    strcpy(head->name, "root");

    head->isDir = true;

    head->p = NULL;

    head->i = 0;

    curr = head;

*while* (true) {

        printf("\n1. List directory. \n2. Change directory. \n3. Go to parent directory. \n4. Add new file. \n5. Delete file. \n6. Create new directory. \n7. Delete directory. \n8. Print working directory. \n9. Exit. \nENTER CHOICE: ", curr->name);

        scanf("%d", &in);

*switch* (in) {

*case* 1:

                ls();

*break*;

*case* 2:

                cd();

*break*;

*case* 3:

                cdup();

*break*;

*case* 4:

                touch(false);

*break*;

*case* 5:

                rm(false);

*break*;

*case* 6:

                touch(true);

*break*;

*case* 7:

                rm(true);

*break*;

*case* 8:

                printf("%s\n", curr->name);

*break*;

*default*:

                exit(0);

        }

    }

}

**Output:**

1. Single-level directory

Enter the directory name: sk

1. Create file.

2. Delete file.

3. Search file.

4. Display files.

5. Exit.

ENTER CHOICE: 1

Enter filename: f1

File created

1. Create file.

2. Delete file.

3. Search file.

4. Display files.

5. Exit.

ENTER CHOICE: 1

Enter filename: f2

File created

1. Create file.

2. Delete file.

3. Search file.

4. Display files.

5. Exit.

ENTER CHOICE: 4

Files in directory sk:

f1

f2

1. Create file.

2. Delete file.

3. Search file.

4. Display files.

5. Exit.

ENTER CHOICE: 3

Enter the name of the file: f2

File f2 found!

1. Create file.

2. Delete file.

3. Search file.

4. Display files.

5. Exit.

ENTER CHOICE: 4

Files in directory sk:

f1

f2

1. Create file.

2. Delete file.

3. Search file.

4. Display files.

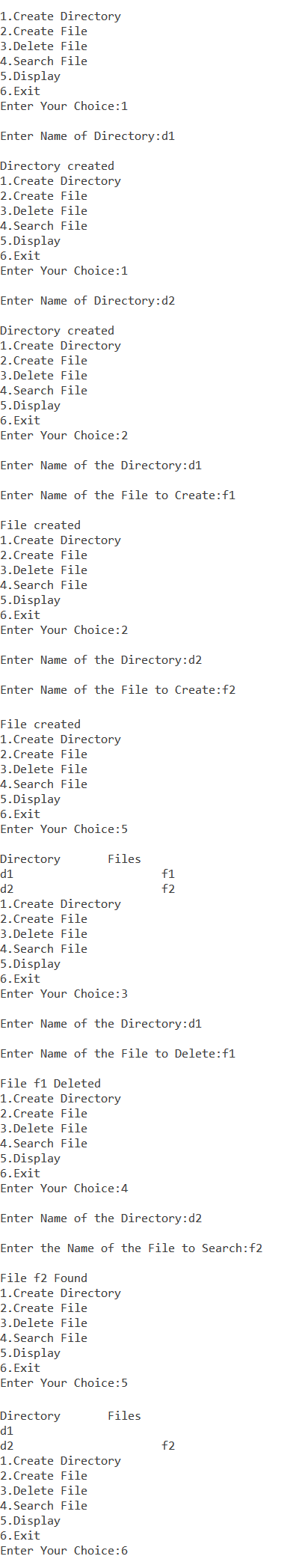
5. Exit.

ENTER CHOICE: 2

Enter the name of the file: f2

Deleting file f2

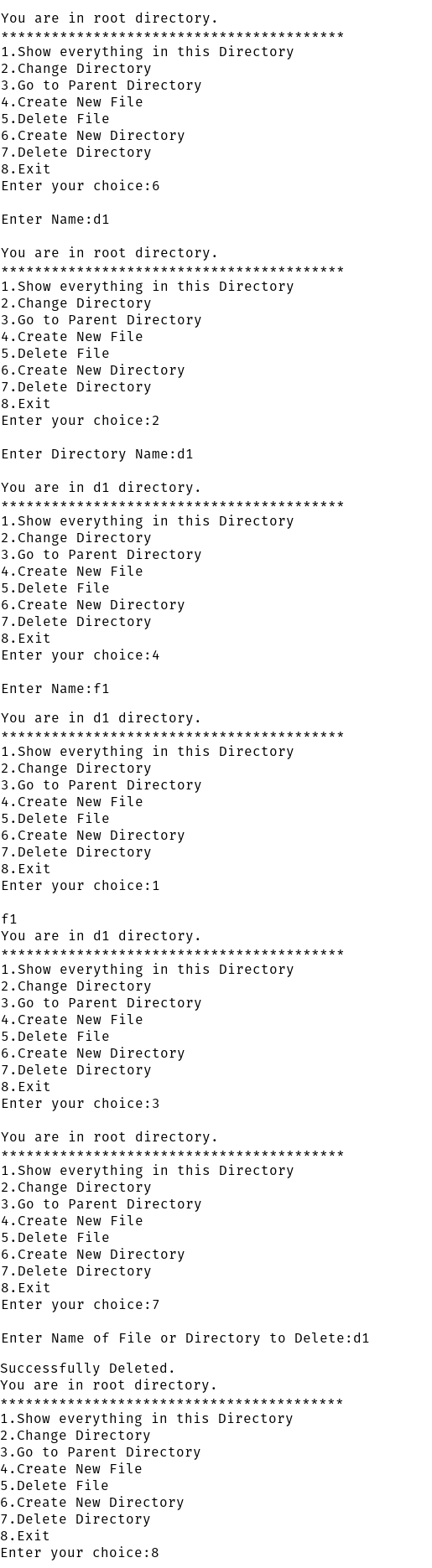
1. Two-level directory



A screenshot of a map

Description automatically generated

1. Hierarchical directory



A close up of text on a white background

Description automatically generated

1. **Implement Banker’s algorithm for deadlock avoidance.**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

void main() {

int i, j, m, n, count=0, exec; // count is the no. of processes that have completed

bool safe = false;

printf("Enter the no. of processes: ");

scanf("%d", &n);

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

scanf("%d", &m);

int avlbl[m], max[n][m], alloc[n][m], need[n][m], work[m], finish[n];

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

finish[i] = 0; // Initially, none of the processes have finished

printf("Enter Available Resources: \n");

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

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

work[i] = avlbl[i]; // Initialize Work=Available

}

printf("Enter Max. Resources: \n");

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

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

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

printf("Enter Allocation: \n");

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

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

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

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

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

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

printf("\nNeed matrix: \n");

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

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

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

}

printf("\n");

}

while (count < n) {

safe = false;

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

if (!finish[i]) { // process hasn't terminated

exec = 1;

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

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

exec = 0; // process can't execute

break;

}

}

if (exec) {

printf("\nP%d is executing\n", i);

finish[i] = 1;

count++;

safe = true;

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

work[j] += alloc[i][j]; // release the resource after execution

break;

}

}

}

if (!safe) {

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

exit(0);

}

printf("Work: \n");

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

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

printf("\n");

}

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

}

**Output:**

Enter the no. of processes: 5

Enter the no. of resource types: 3

Enter Available Resources:

3 3 2

Enter Max. Resources:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter Allocation:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Need matrix:

7 4 3

1 2 2

6 0 0

0 1 1

4 3 1

P1 is executing

Work:

5 3 2

P3 is executing

Work:

7 4 3

P0 is executing

Work:

7 5 3

P2 is executing

Work:

10 5 5

P4 is executing

Work:

10 5 7

The processes are in safe state.

1. **Simulate the following disk scheduling algorithms: (i) FCFS (ii) SCAN (iii) C-SCAN**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

int head, n, max, req[25], p;

void fcfs() {

int movts=0, i, cur=head;

printf("Head Movements: \n");

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

if (i != n-1) printf("%d --> ", req[i]);

else printf("%d\n", req[i]);

movts += abs(req[i]-cur);

cur = req[i];

}

printf("No. of cylinder movements: %d\n", movts);

}

void sortRequests() {

bool found = false;

int i, j, temp, s;

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

s=i;

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

if (req[j] < req[s])

s=j;

temp = req[i];

req[i] = req[s];

req[s] = temp;

if (!found && req[i] >= head) {

found = true;

p = i;

}

}

}

void scan() {

int movts=0, i, cur=head;

printf("Head Movements: \n");

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

printf("%d --> ", req[i]);

movts += abs(req[i]-cur);

cur = req[i];

if (i==n-1) {

movts += abs(max-cur); // add the movt to the end of the disk

cur = max; // move to the end of the disk

}

}

for (i=p-1; i>=0; i--) {

if (i) printf("%d --> ", req[i]);

else printf("%d\n", req[i]);

movts += abs(req[i]-cur);

cur = req[i];

}

printf("No. of cylinder movements: %d\n", movts);

}

void cScan() {

int movts=0, i, cur=head, j=p;

printf("Head Movements: \n");

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

if (i != n-1) printf("%d --> ", req[j]);

else printf("%d\n", req[j]);

movts += abs(req[j]-cur);

cur = req[j];

if (j==n-1) {

movts += abs(max-cur) + max; // add the movt to the end of the disk and from the end of the disk to the beginning

cur = 0; // move to the end of the disk

}

j=(j+1)%n;

}

printf("No. of cylinder movements: %d\n", movts);

}

void main()

{

int i;

char ch[2];

do {

printf("Enter the upper limit of cylinders: ");

scanf("%d", &max);

printf("Enter the disk head position: ");

scanf("%d", &head);

printf("Enter the no. of requests: ");

scanf("%d", &n);

printf("Enter the requests: \n");

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

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

printf("\nFCFS: \n");

fcfs();

sortRequests();

printf("\nSCAN: \n");

scan();

printf("\nC-SCAN: \n");

cScan();

printf("\nDo you want to continue? Y/N: ");

scanf("%s", ch);

} while (ch[0]=='y' || ch[0]=='Y');

}

**Output:**

Enter the upper limit of cylinders: 199

Enter the disk head position: 100

Enter the no. of requests: 5

Enter the requests:

23 89 132 42 187

FCFS:

Head Movements:

23 --> 89 --> 132 --> 42 --> 187

No. of cylinder movements: 421

SCAN:

Head Movements:

132 --> 187 --> 89 --> 42 --> 23

No. of cylinder movements: 275

C-SCAN:

Head Movements:

132 --> 187 --> 23 --> 42 --> 89

No. of cylinder movements: 387

Do you want to continue? Y/N: n

1. **Implement the Producer-Consumer problem using semaphores.**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

int mutex=1, n, full=0, empty, buffer[25], temp=0, f=-1, r=-1;

void wait(int \*s) {

(\*s)--;

}

void signal(int \*s) {

(\*s)++;

}

void producer() {

int x;

wait (&mutex);

signal(&full);

wait(&empty);

// produce an item

printf("Enter the item to be produced: ");

scanf("%d", &x);

// place the item in buffer

if (f==-1) f++;

r = (r+1)%n;

buffer[r] = x;

printf("Produced item: %d\n\n", x);

signal(&mutex);

}

void consumer() {

wait(&mutex);

wait(&full);

signal(&empty);

//remove an item from buffer

int x = buffer[f];

if (f==r) f=r=-1;

else f = (f+1)%n;

signal(&mutex);

// consume the item

printf("Consumed item: %d\n\n", x);

}

void main()

{

int ch;

printf("Enter the size of the buffer: ");

scanf("%d", &n);

empty=n;

while (true)

{

printf("1. Producer. \n2. Consumer. \n3. Exit. \nENTER CHOICE: ");

scanf("%d", &ch);

switch(ch) {

case 1:

if (empty)

producer();

else

printf("Buffer full!\n\n");

break;

case 2:

if (full)

consumer();

else

printf("Buffer empty!\n\n");

break;

default: exit(0);

}

}

}

**Output:**

Enter the size of the buffer: 3

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 1

Enter the item to be produced: 5

Produced item: 5

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 1

Enter the item to be produced: 17

Produced item: 17

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 2

Consumed item: 5

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 1

Enter the item to be produced: 13

Produced item: 13

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 1

Enter the item to be produced: 12

Produced item: 12

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 1

Buffer full!

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 2

Consumed item: 17

1. Producer.

2. Consumer.

3. Exit.

ENTER CHOICE: 3

1. **Simulate the working of Dining Philosophers’ problem.**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N]; // semaphore for each philosopher

void test(int phnum) {

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

state[phnum] = EATING; // state of eating

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect during takefork

// used to wake up hungry philosophers during putfork

sem\_post(&S[phnum]);

}

}

void take\_fork(int phnum) { // take up chopsticks

sem\_wait(&mutex);

state[phnum] = HUNGRY; // state that hungry

printf("Philosopher %d is Hungry\n", phnum + 1);

test(phnum); // eat if neighbours are not eating

sem\_post(&mutex);

sem\_wait(&S[phnum]); // if unable to eat wait to be signalled

sleep(1);

}

void put\_fork(int phnum) { // put down chopsticks

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philospher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N]; // create 5 threads

sem\_init(&mutex, 0, 1); // initialize the semaphores

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

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++) { // create philosopher processes

pthread\_create(&thread\_id[i], NULL, philospher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

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

pthread\_join(thread\_id[i], NULL);

}

**Output:**

Philosopher 1 is thinking

Philosopher 2 is thinking

Philosopher 3 is thinking

Philosopher 4 is thinking

Philosopher 5 is thinking

Philosopher 2 is Hungry

Philosopher 3 is Hungry

Philosopher 5 is Hungry

Philosopher 4 is Hungry

Philosopher 4 takes fork 3 and 4

Philosopher 4 is Eating

Philosopher 1 is Hungry

Philosopher 1 takes fork 5 and 1

Philosopher 1 is Eating

Philosopher 4 putting fork 3 and 4 down

Philosopher 4 is thinking

Philosopher 3 takes fork 2 and 3

Philosopher 3 is Eating

Philosopher 1 putting fork 5 and 1 down

Philosopher 1 is thinking

Philosopher 5 takes fork 4 and 5

Philosopher 5 is Eating

Philosopher 4 is Hungry

Philosopher 3 putting fork 2 and 3 down

Philosopher 3 is thinking

**PART B**

1. **Implement Pass 1 of a Two-Pass Assembler.**

#include<stdio.h>

#include<string.h>

void main()

{

FILE \*f1, \*f2, \*f3, \*f4, \*flen;

int lc, sa, op1, o, len; // locctr, starting addr, operand, machine code, length of byte string

char m1[20], la[20], op[20], otp[20]; // mnemonic, label, opcode, opcode in optab

f1 = fopen("input.txt", "r");

f3 = fopen("symtab.txt", "w");

f4 = fopen("out1.txt", "w");

fscanf(f1, "%s %s %d", la, m1, &op1);

if(strcmp(m1, "START")==0) {

sa = op1;

lc = sa;

printf("-\t%s\t%s\t%d\n", la, m1, op1);

fprintf(f4, "-\t%s\t%s\t%d\n", la, m1, op1);

}

else

lc = 0;

fscanf(f1, "%s %s", la, m1);

while(!feof(f1)) {

fscanf(f1, "%s", op);

printf("\n%d\t%s\t%s\t%s\n", lc, la, m1, op);

fprintf(f4, "%d\t%s\t%s\t%s\n", lc, la, m1, op);

if(strcmp(la, "-")!=0)

fprintf(f3, "\n%d\t%s\n", lc, la);

f2 = fopen("optab.txt", "r");

fscanf(f2, "%s %d", otp, &o);

while(!feof(f2)) { // check if mnemonic opcode is there in optab

if(strcmp(m1, otp)==0) {

lc += 3;

break;

}

fscanf(f2, "%s %d", otp, &o);

}

fclose(f2);

if(strcmp(m1, "WORD")==0)

lc += 3;

else if(strcmp(m1, "RESW")==0) {

op1 = atoi(op);

lc += (3\*op1);

}

else if(strcmp(m1, "BYTE")==0) {

if(op[0]=='X') // hex value

lc += 1;

else { // char const

len = strlen(op)-2;

lc += len;

}

}

else if(strcmp(m1, "RESB")==0) {

op1 = atoi(op);

lc += op1;

}

fscanf(f1, "%s%s", la, m1);

}

if(strcmp(m1, "END")==0) {

printf("Program length: %d\n\n", lc-sa);

flen = fopen("length.txt", "w");

fprintf(flen, "%d\n", lc-sa);

fclose(flen);

}

fclose(f1);

fclose(f3);

fclose(f4);

}

input.txt

copy START 1000

- LDA ALPHA

- ADD ONE

- SUB TWO

- STA BETA

ALPHA BYTE C'HOWDY

ONE RESB 2

TWO WORD 5

BETA RESW 1

- END -

symtab.txt

1012 ALPHA

1017 ONE

1019 TWO

1022 BETA

out1.txt

- copy START 1000

1000 - LDA ALPHA

1003 - ADD ONE

1006 - SUB TWO

1009 - STA BETA

1012 ALPHA BYTE C'HOWDY

1017 ONE RESB 2

1019 TWO WORD 5

1022 BETA RESW 1

1025 - END -

1. **Implement Pass 2 of a Two-Pass Assembler.**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

void main()

{

FILE \*fint, \*ftab, \*flen, \*fsym, \*fout;

int op1[10], txtlen, txtlen1, i, j = 0, len;

char add[5], symadd[5], op[5], start[10], temp[30], line[20], label[20], mne[10], operand[10], symtab[10], opmne[10];

fint = fopen("out1.txt", "r");

flen = fopen("length.txt", "r");

ftab = fopen("optab.txt", "r");

fsym = fopen("symtab.txt", "r");

fout = fopen("output.txt", "w");

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

if(strcmp(mne, "START")==0) {

strcpy(start, operand);

fscanf(flen, "%d", &len);

fclose(flen);

}

printf("H^%s^%6s^%06d\nT^00%s^", label, start, len, start);

fprintf(fout, "H^%s^%s^%d\nT^00%s^", label, start, len, start);

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

while(strcmp(mne, "END")!=0) {

fscanf(ftab, "%s%s", opmne, op);

while(!feof(ftab)) {

if(strcmp(mne, opmne)==0) {

fclose(ftab);

fscanf(fsym, "%s%s", symadd, symtab);

while(!feof(fsym)) {

if(strcmp(operand, symtab)==0) {

printf("%s%s^", op, symadd);

fprintf(fout, "%s%s^", op, symadd);

break;

}

else

fscanf(fsym, "%s%s", symadd, symtab);

}

break;

}

else

fscanf(ftab, "%s%s", opmne, op);

}

if((strcmp(mne, "BYTE")==0)||(strcmp(mne, "WORD")==0)) {

if(strcmp(mne, "WORD")==0) {

printf("0000%s^", operand);

fprintf(fout, "0000%s^", operand);

}

else {

len = strlen(operand);

for(i = 2;i<len;i++) {

printf("%d", operand[i]);

fprintf(fout, "%d", operand[i]);

}

printf("^");

fprintf(fout, "^");

}

}

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

ftab = fopen("optab.txt", "r");

fseek(ftab, SEEK\_SET, 0);

}

printf("\nE^00%s\n\n", start);

fprintf(fout, "\nE^00%s\n", start);

fclose(fint);

fclose(ftab);

fclose(fsym);

fclose(fout);

}

length.txt

25

optab.txt

LDA 00

STA 23

ADD 01

SUB 05

output.txt

H^copy^1000^25

T^001000^001012^011017^051019^231022^7279876889^00005^

E^001000

1. **Implement a Single Pass Assembler.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void main()

{

FILE \*f1, \*f2, \*f3, \*f4, \*f5;

int lc, sa, i = 0, j = 0, m[10], pgmlen, len, k, len1, l = 0;

char name[10], opnd[10], la[10], mne[10], s1[10], mne1[10], opnd1[10];

char lcs[10], ms[10];

char sym[10], symaddr[10], obj1[10], obj2[10], s2[10], q[10], s3[10];

f1 = fopen("input.txt", "r");

f2 = fopen("optab.txt", "r");

f3 = fopen("symtab.txt", "w+");

f4 = fopen("symtab1.txt", "w+");

f5 = fopen("output.txt", "w+");

fscanf(f1, "%s%s%s", la, mne, opnd);

if (strcmp(mne, "START") == 0) {

sa = atoi(opnd);

strcpy(name, la);

lc = sa;

}

strcpy(s1, "\*");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "END") != 0) {

if (strcmp(la, "-") == 0) {

fscanf(f2, "%s%s", mne1, opnd1);

while (!feof(f2)) {

if (strcmp(mne1, mne) == 0) {

m[i] = lc + 1;

fprintf(f3, "%s\t%s\n", opnd, s1);

fprintf(f5, "%s\t0000\n", opnd1);

lc = lc + 3;

i = i + 1;

break;

}

else

fscanf(f2, "%s%s", mne1, opnd1);

}

}

else {

fseek(f3, SEEK\_SET, 0);

fscanf(f3, "%s%s", sym, symaddr);

while (!feof(f3)) {

if (strcmp(sym, la) == 0) {

sprintf(lcs, "%d", lc);

fprintf(f4, "%s\t%s\n", la, lcs);

sprintf(ms, "%d", m[j]);

j = j + 1;

fprintf(f5, "%s\t%s\n", ms, lcs);

i = i + 1;

break;

}

else

fscanf(f3, "%s%s", sym, symaddr);

}

if (strcmp(mne, "RESW") == 0)

lc = lc + 3 \* atoi(opnd);

else if (strcmp(mne, "BYTE") == 0) {

strcpy(s2, "-");

len = strlen(opnd);

lc = lc + len - 2;

for (k = 2; k < len; k++) {

q[l] = opnd[k];

l = l + 1;

}

fprintf(f5, "%s\t%s\n", q, s2);

break;

}

else if (strcmp(mne, "RESB") == 0)

lc = lc + atoi(opnd);

else if (strcmp(mne, "WORD") == 0) {

strcpy(s3, "#");

lc = lc + 3;

fprintf(f5, "%s\t%s\n", opnd, s3);

break;

}

}

fseek(f2, SEEK\_SET, 0);

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fseek(f5, SEEK\_SET, 0);

pgmlen = lc - sa;

printf("H^%s^%d^0%x\n", name, sa, pgmlen);

printf("T^");

printf("00%d^0%x", sa, pgmlen);

fscanf(f5, "%s%s", obj1, obj2);

while (!feof(f5)) {

if (strcmp(obj2, "0000") == 0)

printf("^%s%s", obj1, obj2);

else if (strcmp(obj2, "-") == 0) {

printf("^");

len1 = strlen(obj1);

for (k = 0; k < len1; k++)

printf("%d", obj1[k]);

}

else if (strcmp(obj2, "#") == 0) {

printf("^");

printf("%s", obj1);

}

fscanf(f5, "%s%s", obj1, obj2);

}

fseek(f5, SEEK\_SET, 0);

fscanf(f5, "%s%s", obj1, obj2);

while (!feof(f5)) {

if (strcmp(obj2, "0000") != 0) {

if (strcmp(obj2, "-") != 0) {

if (strcmp(obj2, "#") != 0) {

printf("\n");

printf("T^%s^02^%s", obj1, obj2);

}

}

}

fscanf(f5, "%s%s", obj1, obj2);

}

printf("\nE^00%d\n", sa);

}

input.txt

COPY START 1000

- LDA ALPHA

- STA BETA

ALPHA RESW 1

BETA RESW 1

- END -

optab.txt

LDA 00

STA 23

LDCH 15

STCH 18

symtab.txt

ALPHA \*

BETA \*

symtab1.txt

ALPHA 1006

BETA 1009

output.txt

00 0000

23 0000

1001 1006

1004 1009

1. **Implement a Two-Pass Macro Processor.**

**// PASS 1**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

FILE \*f1, \*f2, \*f3;

char mne[20], opnd[20], la[20];

f1 = fopen("inp.txt", "r");

f2 = fopen("namtab.txt", "w+");

f3 = fopen("argtab.txt", "w+");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "MEND") != 0) {

if (strcmp(mne, "MACRO") == 0) {

fprintf(f2, "%s\n", la);

fprintf(f3, "%s\t%s\n", la, opnd);

}

else

fprintf(f3, "%s\t%s\n", mne, opnd);

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fprintf(f3, "%s", mne);

fclose(f1);

fclose(f2);

fclose(f3);

printf("Pass 1 is completed\n");

}

**// PASS 2**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

FILE \*f1, \*f2, \*f3, \*f4, \*f5;

int i, len;

char mne[20], opnd[20], la[20], name[20], mne1[20], opnd1[20], arg[20];

f1 = fopen("inp.txt", "r");

f2 = fopen("namtab.txt", "r");

f3 = fopen("argtab.txt", "r");

f4 = fopen("atab2.txt", "w+");

f5 = fopen("op2.txt", "w");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "END") != 0) {

if (strcmp(mne, "MACRO") == 0) {

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "MEND") != 0)

fscanf(f1, "%s%s%s", la, mne, opnd);

}

else {

fscanf(f2, "%s", name);

if (strcmp(mne, name) == 0) {

len = strlen(opnd);

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

if (opnd[i] != ',')

fprintf(f4, "%c", opnd[i]);

else

fprintf(f4, "\n");

}

fseek(f2, SEEK\_SET, 0);

fseek(f4, SEEK\_SET, 0);

fscanf(f3, "%s%s", mne1, opnd1);

fprintf(f5, ".\t%s\t%s\n", mne1, opnd);

fscanf(f3, "%s%s", mne1, opnd1);

while (strcmp(mne1, "MEND") != 0) {

if ((opnd1[0] == '&')) {

fscanf(f4, "%s", arg);

fprintf(f5, "-\t%s\t%s\n", mne1, arg);

}

else

fprintf(f5, "-\t%s\t%s\n", mne1, opnd1);

fscanf(f3, "%s%s", mne1, opnd1);

}

}

else

fprintf(f5, "%s\t%s\t%s\n", la, mne, opnd);

}

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fprintf(f5, "%s\t%s\t%s\n", la, mne, opnd);

fclose(f1);

fclose(f2);

fclose(f3);

fclose(f4);

fclose(f5);

printf("Pass 2 completed\n");

}

**Pass 1:**

inp.txt

EX1 MACRO &A,&B

- LDA &A

- STA &B

- MEND -

SAMPLE START 1000

- EX1 N1,N2

N1 RESW 1

N2 RESW 1

- END -

namtab.txt

EX1

argtab.txt

EX1 &A,&B

LDA &A

STA &B

MEND

**Pass 2:**

atab2.txt

N1

N2

op2.txt

SAMPLE START 1000

. EX1 N1,N2

- LDA N1

- STA N2

N1 RESW 1

N2 RESW 1

- END -

1. **Implement an Absolute Loader.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void main()

{

FILE \*fp;

int addr, staddri;

char line[50], staddr[10];

fp = fopen("object\_code.txt", "r");

fscanf(fp, "%s", line);

while (!feof(fp)) {

fscanf(fp, "%s", line);

if (line[0] == 'T') {

int i = 0, j = 0;

for (i = 2, j = 0; i < 8; i++, j++)

staddr[j] = line[i];

staddr[j] = '\0';

staddri = atoi(staddr);

i = 12;

while (line[i] != '$') {

if (line[i] != '^') {

printf("00%d %c%c\n", staddri, line[i], line[i + 1]);

staddri++;

i += 2;

}

else

i++;

}

}

else if (line[0] == 'E')

break;

}

}

object\_code.txt

H^SAMPLE^001000^0035

T^001000^0C^001003^071009$

T^002000^03^111111$

E^001000

**Output:**

001000 00

001001 10

001002 03

001003 07

001004 10

001005 09

002000 11

002001 11

002002 11

1. **Implement a Symbol Table with Suitable Hashing.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define LENGTH 7

struct hashTable {

char label[10];

int addr;

} ht[LENGTH];

void addLabel()

{

int addr;

char label[10];

printf("Enter label name: ");

scanf("%s", label);

printf("Enter label address: ");

scanf("%d", &addr);

int loc = addr % LENGTH;

if (ht[loc].addr == -1)

{

ht[loc].addr = addr;

strcpy(ht[loc].label, label);

}

else

printf("Hashtable slot occupied\n");

}

void display()

{

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

if (ht[i].addr != -1)

printf("%d %s\n", ht[i].addr, ht[i].label);

else

printf("0 0\n");

}

void search()

{

char label[10];

int i, set=0, s;

printf("Enter label name: ");

scanf("%s", label);

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

if (ht[i].addr) {

if (!strcmp(ht[i].label, la)) {

set=1;

s = ht[i].addr;

}

}

}

if (set)

printf("Label is present!\n");

else printf("Label is not present!\n");

}

void main()

{

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

ht[i].addr = -1;

strcpy(ht[i].label, "");

}

int c = 0;

while (c < 3) {

printf("1. Add label. \2. View hashtable. \nENTER CHOICE: ");

scanf("%d", &c);

switch (c) {

case 1:

addLabel();

break;

case 2:

display();

break;

default: exit(0);

}

}

}

**Output:**

1. Add label.

2. View hashtable.

3. Search for label.

ENTER CHOICE: 1

Enter label name: loop

Enter label address: 1275

1. Add label.

2. View hashtable.

3. Search for label.

ENTER CHOICE: 1

Enter label name: clear

Enter label address: 6475

1. Add label.

2. View hashtable.

3. Search for label.

ENTER CHOICE: 1

Enter label name: rdlp

Enter label address: 2467

1. Add label.

2. View hashtable.

3. Search for label.

ENTER CHOICE: 2

6475 clear

1275 loop

0 0

2467 rdlp

0 0

0 0

0 0

1. Add label.

2. View hashtable.

3. Search for label.

ENTER CHOICE: 3

Enter label name: loop

Label is present!

1. Add label.

2. View hashtable.

3. Search for label.

ENTER CHOICE: 4