1. Simulate the following non-preemptive CPU scheduling algorithms to find turnaround and waiting time: (a) FCFS (b) SJF (c) RR (preemptive) (d) Priority
2. Simulate the following file organization techniques: (i) Single-level directory (ii) Two-level directory (iii) Hierarchical directory

*// Simulate single-level directory structure.*

*#include*<stdio.h>

*#include*<conio.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);

        }

    }

}

1. Implement Banker’s algorithm for deadlock avoidance.
2. Simulate the following disk scheduling algorithms: (i) FCFS (ii) SCAN (iii) C-SCAN
3. Implement the Producer-Consumer problem using semaphores.
4. Simulate the working of Dining Philosophers’ problem.
5. Implement Pass 1 of a 2 Pass Assembler.
6. Implement Pass 2 of a 2 Pass Assembler.
7. Implement a Single Pass Assembler.
8. Implement a 2 Pass Macro Assembler.