Program 1

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid, child\_pid;

int status;

pid = fork();

if (pid < 0) {

perror("fork failed");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

printf("Child process: PID = %d, PPID = %d\n", getpid(), getppid());

execlp("ls", "ls", NULL);

perror("execlp failed");

exit(EXIT\_FAILURE);

} else {

printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);

child\_pid = wait(&status);

if (child\_pid == -1) {

perror("wait failed");

exit(EXIT\_FAILURE);

}

if (WIFEXITED(status)) {

printf("Child process exited with status %d\n", WEXITSTATUS(status));

} else {

printf("Child process did not exit normally\n");

}

}

return 0;

}

Output

braham@braham:~/Desktop/program$ gcc prg1.c -o prg1

braham@braham:~/Desktop/program$ ./prg1

Parent process: PID = 4203, Child PID = 4204

Child process: PID = 4204, PPID = 4203

prg1 prg1.c

Child process exited with status 0

Program 2

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int id;

int burst\_time;

int priority;

} Process;

void fcfs\_scheduling(int n, int burst\_times[]) {

int waiting\_time[n], turnaround\_time[n];

waiting\_time[0] = 0;

turnaround\_time[0] = burst\_times[0];

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

waiting\_time[i] = waiting\_time[i - 1] + burst\_times[i - 1];

turnaround\_time[i] = waiting\_time[i] + burst\_times[i];

}

printf("FCFS Scheduling\n");

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

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

printf("P%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst\_times[i], waiting\_time[i], turnaround\_time[i]);

}

}

int compare\_sjf(const void \*a, const void \*b) {

return ((Process \*)a)->burst\_time - ((Process \*)b)->burst\_time;

}

void sjf\_scheduling(int n, Process processes[]) {

int waiting\_time[n], turnaround\_time[n];

qsort(processes, n, sizeof(Process), compare\_sjf);

waiting\_time[0] = 0;

turnaround\_time[0] = processes[0].burst\_time;

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

waiting\_time[i] = waiting\_time[i - 1] + processes[i - 1].burst\_time;

turnaround\_time[i] = waiting\_time[i] + processes[i].burst\_time;

}

printf("SJF Scheduling\n");

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

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

printf("P%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst\_time, waiting\_time[i], turnaround\_time[i]);

}

}

void round\_robin\_scheduling(int n, int burst\_times[], int quantum) {

int remaining\_times[n], waiting\_time[n], turnaround\_time[n];

int t = 0;

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

remaining\_times[i] = burst\_times[i];

}

while (1) {

int done = 1;

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

if (remaining\_times[i] > 0) {

done = 0;

if (remaining\_times[i] > quantum) {

t += quantum;

remaining\_times[i] -= quantum;

} else {

t += remaining\_times[i];

waiting\_time[i] = t - burst\_times[i];

remaining\_times[i] = 0;

}

}

}

if (done) {

break;

}

}

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

turnaround\_time[i] = burst\_times[i] + waiting\_time[i];

}

printf("Round Robin Scheduling\n");

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

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

printf("P%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst\_times[i], waiting\_time[i], turnaround\_time[i]);

}

}

int compare\_priority(const void \*a, const void \*b) {

return ((Process \*)a)->priority - ((Process \*)b)->priority;

}

void priority\_scheduling(int n, Process processes[]) {

int waiting\_time[n], turnaround\_time[n];

qsort(processes, n, sizeof(Process), compare\_priority);

waiting\_time[0] = 0;

turnaround\_time[0] = processes[0].burst\_time;

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

waiting\_time[i] = waiting\_time[i - 1] + processes[i - 1].burst\_time;

turnaround\_time[i] = waiting\_time[i] + processes[i].burst\_time;

}

printf("Priority Scheduling\n");

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

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

printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst\_time, processes[i].priority, waiting\_time[i], turnaround\_time[i]);

}

}

int main() {

int n, quantum;

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

scanf("%d", &n);

int burst\_times[n];

Process processes[n];

printf("Enter burst times for each process:\n");

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

printf("Burst Time for P%d: ", i + 1);

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

processes[i].id = i + 1;

processes[i].burst\_time = burst\_times[i];

}

printf("Enter the quantum time for Round Robin (0 to skip): ");

scanf("%d", &quantum);

if (quantum > 0) {

round\_robin\_scheduling(n, burst\_times, quantum);

}

printf("Enter priorities for each process:\n");

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

printf("Priority for P%d: ", i + 1);

scanf("%d", &processes[i].priority);

}

fcfs\_scheduling(n, burst\_times);

sjf\_scheduling(n, processes);

priority\_scheduling(n, processes);

return 0;

} braham@braham:~/Desktop/program$ gcc prg2.c -o prg2

braham@braham:~/Desktop/program$ ./prg2

Enter the number of processes: 4

Enter burst times for each process:

Burst Time for P1: 10

Burst Time for P2: 5

Burst Time for P3: 8

Burst Time for P4: 12

Enter the quantum time for Round Robin (0 to skip): 4

Round Robin Scheduling

Process ID Burst Time Waiting Time Turnaround Time

P1 10 21 31

P2 5 16 21

P3 8 17 25

P4 12 23 35

Enter priorities for each process:

Priority for P1: 2

Priority for P2: 1

Priority for P3: 4

Priority for P4: 3

FCFS Scheduling

Process ID Burst Time Waiting Time Turnaround Time

P1 10 0 10

P2 5 10 15

P3 8 15 23

P4 12 23 35

SJF Scheduling

Process ID Burst Time Waiting Time Turnaround Time

P2 5 0 5

P3 8 5 13

P1 10 13 23

P4 12 23 35

Priority Scheduling

Process ID Burst Time Priority Waiting Time Turnaround Time

P2 5 1 0 5

P1 10 2 5 15

P4 12 3 15 27

P3 8 4 27 35

Program 3

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int in = 0;

int out = 0;

sem\_t empty;

sem\_t full;

sem\_t mutex;

void\* producer(void\* arg) {

while (1) {

sleep(rand() % 3);

int item = rand() % 100;

sem\_wait(&empty);

sem\_wait(&mutex);

buffer[in] = item;

in = (in + 1) % BUFFER\_SIZE;

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

sem\_post(&mutex);

sem\_post(&full);

}

return NULL;

}

void\* consumer(void\* arg) {

while (1) {

sleep(rand() % 3);

sem\_wait(&full);

sem\_wait(&mutex);

int item = buffer[out];

out = (out + 1) % BUFFER\_SIZE;

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

sem\_post(&mutex);

sem\_post(&empty);

}

return NULL;

}

int main() {

pthread\_t producer\_thread, consumer\_thread;

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

sem\_init(&mutex, 0, 1);

pthread\_create(&producer\_thread, NULL, producer, NULL);

pthread\_create(&consumer\_thread, NULL, consumer, NULL);

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

sem\_destroy(&empty);

sem\_destroy(&full);

sem\_destroy(&mutex);

return 0;

} braham@braham:~/Desktop/program$ gcc prg3.c -o prg3

braham@braham:~/Desktop/program$ ./prg3

Produced: 77

Consumed: 77

Produced: 35

Consumed: 35

Produced: 49

Consumed: 49

Produced: 27

Consumed: 27

Produced: 63

Consumed: 63

Produced: 26

Consumed: 26

Produced: 11

Consumed: 11

Produced: 29

Consumed: 29

Produced: 62

Consumed: 62

Produced: 35

Consumed: 35

Produced: 22

Program 4

Writer.c(this program will create the FIFO(if it does not already exist)and write a message it

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <string.h>

#define FIFO\_NAME "/tmp/myfifo"

#define BUFFER\_SIZE 256

int main() {

int fd;

const char \*message = "Hello from writer process!";

if (mkfifo(FIFO\_NAME, 0666) == -1) {

perror("mkfifo");

exit(EXIT\_FAILURE);

}

printf("Writer: FIFO created\n");

fd = open(FIFO\_NAME, O\_WRONLY);

if (fd == -1) {

perror("open");

exit(EXIT\_FAILURE);

}

printf("Writer: FIFO opened for writing\n");

size\_t message\_length = strlen(message);

if (write(fd, message, message\_length) == -1) {

perror("write");

close(fd);

exit(EXIT\_FAILURE);

}

printf("Writer: Message written to FIFO\n");

close(fd);

return 0;

}

Reader.c(this program will open the FIFO and read the message from the writer

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#define FIFO\_NAME "/tmp/myfifo"

#define BUFFER\_SIZE 256

int main() {

int fd;

char buffer[BUFFER\_SIZE];

ssize\_t bytesRead;

fd = open(FIFO\_NAME, O\_RDONLY);

if (fd == -1) {

perror("open");

exit(EXIT\_FAILURE);

}

bytesRead = read(fd, buffer, BUFFER\_SIZE - 1);

if (bytesRead == -1) {

perror("read");

close(fd);

exit(EXIT\_FAILURE);

}

buffer[bytesRead] = '\0';

printf("Reader: Message received: %s\n", buffer);

close(fd);

if (unlink(FIFO\_NAME) == -1) {

perror("unlink");

exit(EXIT\_FAILURE);

}

return 0;

}

Compile the reader and writer programs one by one using the following command

braham@braham:~/Desktop/program$ gcc writer.c -o writer

braham@braham:~/Desktop/program$ gcc reader.c -o reader

* open two terminals
* in the first terminal
* ./writer
* In the second terminal
* ./reader
* Click both the terminal to see the output
* braham@braham:~/Desktop/program$ ./writer
* Writer: FIFO created
* Writer: FIFO opened for writing
* Writer: Message written to FIFO
* braham@braham:~/Desktop/program$ ./reader
* Reader: Message received: Hello from writer process!

Program 5

#include <stdio.h>

#include <stdbool.h>

#define MAX 10

#define RESOURCES 3

#define PROCESSES 5

void calculateNeed(int need[PROCESSES][RESOURCES], int max[PROCESSES][RESOURCES], int allot[PROCESSES][RESOURCES]);

bool isSafe(int processes[], int avail[], int max[][RESOURCES], int allot[][RESOURCES]);

int main() {

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

int resources[] = {10, 5, 7};

int allot[PROCESSES][RESOURCES] = {

{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

int max[PROCESSES][RESOURCES] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

int avail[RESOURCES];

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

avail[i] = resources[i];

for (int j = 0; j < PROCESSES; j++) {

avail[i] -= allot[j][i];

}

}

isSafe(processes, avail, max, allot);

return 0;

}

void calculateNeed(int need[PROCESSES][RESOURCES], int max[PROCESSES][RESOURCES], int allot[PROCESSES][RESOURCES]) {

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

for (int j = 0; j < RESOURCES; j++) {

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

}

}

}

bool isSafe(int processes[], int avail[], int max[][RESOURCES], int allot[][RESOURCES]) {

int need[PROCESSES][RESOURCES];

calculateNeed(need, max, allot);

bool finish[PROCESSES] = {0};

int safeSeq[PROCESSES];

int work[RESOURCES];

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

work[i] = avail[i];

int count = 0;

while (count < PROCESSES) {

bool found = false;

for (int p = 0; p < PROCESSES; p++) {

if (!finish[p]) {

int j;

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

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

break;

if (j == RESOURCES) {

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

work[k] += allot[p][k];

safeSeq[count++] = p;

finish[p] = 1;

found = true;

}

}

}

if (!found) {

printf("System is not in a safe state\n");

return false;

}

}

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

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

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

printf("\n");

return true;

}

Out put

braham@braham:~/Desktop/program$ gcc banker.c -o banker

braham@braham:~/Desktop/program$ ./banker

System is in a safe state.

Safe sequence is: 1 3 4 0 2

Program 6

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MEMORY\_SIZE 1000

typedef struct {

int start;

int size;

bool isFree;

} Block;

Block memory[MEMORY\_SIZE];

void initializeMemory() {

for (int i = 0; i < MEMORY\_SIZE; i++) {

memory[i].start = i;

memory[i].size = 1;

memory[i].isFree = true;

}

}

void displayMemory() {

printf("Memory Status:\n");

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (!memory[i].isFree) {

printf("Block from %d to %d is occupied\n", memory[i].start, memory[i].start + memory[i].size - 1);

}

}

}

void allocateMemory(int size, char method) {

int bestIdx = -1, worstIdx = -1, firstIdx = -1;

int bestSize = MEMORY\_SIZE + 1, worstSize = -1;

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i].isFree) {

int j = i;

int currentSize = 0;

while (j < MEMORY\_SIZE && memory[j].isFree) {

currentSize++;

j++;

}

if (currentSize >= size) {

if (method == 'b' && currentSize < bestSize) {

bestSize = currentSize;

bestIdx = i;

} else if (method == 'w' && currentSize > worstSize) {

worstSize = currentSize;

worstIdx = i;

} else if (method == 'f' && firstIdx == -1) {

firstIdx = i;

}

}

i = j - 1;

}

}

int idx = (method == 'b') ? bestIdx : (method == 'w') ? worstIdx : firstIdx;

if (idx != -1) {

int allocated = 0;

// Allocate the memory

for (int i = idx; i < MEMORY\_SIZE && allocated < size; i++) {

if (memory[i].isFree) {

memory[i].isFree = false;

allocated++;

}

}

printf("Allocated %d units using %s fit starting at %d\n", size, (method == 'b') ? "Best" : (method == 'w') ? "Worst" : "First", idx);

} else {

printf("Failed to allocate %d units of memory\n", size);

}

}

void deallocateMemory(int start, int size) {

for (int i = start; i < start + size && i < MEMORY\_SIZE; i++) {

memory[i].isFree = true;

}

printf("Deallocated %d units of memory starting from %d\n", size, start);

}

int main() {

char method;

int size, start;

initializeMemory();

while (1) {

printf("\nMemory Allocation Simulator\n");

printf("1. Allocate Memory (First fit: 'f', Best fit: 'b', Worst fit: 'w')\n");

printf("2. Deallocate Memory\n");

printf("3. Display Memory\n");

printf("4. Exit\n");

printf("Enter your choice: ");

int choice;

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter allocation size: ");

scanf("%d", &size);

printf("Enter fit method (f for First fit, b for Best fit, w for Worst fit): ");

scanf(" %c", &method);

allocateMemory(size, method);

break;

case 2:

printf("Enter deallocation start and size: ");

scanf("%d %d", &start, &size);

deallocateMemory(start, size);

break;

case 3:

displayMemory();

break;

case 4:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

Output

braham@braham:~/Desktop/program$ gcc memory.c -o memory

braham@braham:~/Desktop/program$ ./memory

Memory Allocation Simulator

1. Allocate Memory (First fit: 'f', Best fit: 'b', Worst fit: 'w')

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 1

Enter allocation size: 50

Enter fit method (f for First fit, b for Best fit, w for Worst fit): f

Allocated 50 units using First fit starting at 0

Memory Allocation Simulator

1. Allocate Memory (First fit: 'f', Best fit: 'b', Worst fit: 'w')

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 1

Enter allocation size: 30

Enter fit method (f for First fit, b for Best fit, w for Worst fit): b

Allocated 30 units using Best fit starting at 50

Memory Allocation Simulator

1. Allocate Memory (First fit: 'f', Best fit: 'b', Worst fit: 'w')

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 1

Enter allocation size: 100

Enter fit method (f for First fit, b for Best fit, w for Worst fit): w

Allocated 100 units using Worst fit starting at 80

Memory Allocation Simulator

1. Allocate Memory (First fit: 'f', Best fit: 'b', Worst fit: 'w')

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 2

Enter deallocation start and size: 0 50

Deallocated 50 units of memory starting from 0

Memory Allocation Simulator

1. Allocate Memory (First fit: 'f', Best fit: 'b', Worst fit: 'w')

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 3

Memory Status:

Block from 50 to 50 is occupied

Block from 51 to 51 is occupied

Block from 52 to 52 is occupied

Block from 53 to 53 is occupied

Block from 54 to 54 is occupied

Block from 55 to 55 is occupied

Block from 56 to 56 is occupied

Block from 57 to 57 is occupied

Block from 58 to 58 is occupied

Block from 59 to 59 is occupied

Block from 60 to 60 is occupied

Block from 61 to 61 is occupied

Block from 62 to 62 is occupied

Block from 63 to 63 is occupied

Block from 64 to 64 is occupied

Block from 65 to 65 is occupied

Memory Allocation Simulator

1. Allocate Memory (First fit: 'f', Best fit: 'b', Worst fit: 'w')

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 4

Program 7

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FRAMES 10

#define MAX\_PAGES 20

void fifoPageReplacement(int pages[], int num\_pages, int num\_frames) {

int frame[num\_frames];

int page\_faults = 0, index = 0, i, j, k, flag;

for (i = 0; i < num\_frames; i++) {

frame[i] = -1;

}

printf("\nFIFO Page Replacement Algorithm\n");

for (i = 0; i < num\_pages; i++) {

flag = 0;

for (j = 0; j < num\_frames; j++) {

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

flag = 1;

break;

}

}

if (flag == 0) {

frame[index] = pages[i];

index = (index + 1) % num\_frames;

page\_faults++;

printf("Page Fault: ");

for (k = 0; k < num\_frames; k++) {

if (frame[k] != -1) {

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

}

}

printf("\n");

}

}

printf("Total Page Faults: %d\n", page\_faults);

}

void lruPageReplacement(int pages[], int num\_pages, int num\_frames) {

int frame[num\_frames];

int last\_used[num\_frames];

int page\_faults = 0, index, i, j, k, flag, lru\_index;

for (i = 0; i < num\_frames; i++) {

frame[i] = -1;

last\_used[i] = 0;

}

printf("\nLRU Page Replacement Algorithm\n");

for (i = 0; i < num\_pages; i++) {

flag = 0;

for (j = 0; j < num\_frames; j++) {

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

flag = 1;

last\_used[j] = i;

break;

}

}

if (flag == 0) {

lru\_index = 0;

for (j = 1; j < num\_frames; j++) {

if (last\_used[j] < last\_used[lru\_index]) {

lru\_index = j;

}

}

frame[lru\_index] = pages[i];

last\_used[lru\_index] = i;

page\_faults++;

printf("Page Fault: ");

for (k = 0; k < num\_frames; k++) {

if (frame[k] != -1) {

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

}

}

printf("\n");

}

}

printf("Total Page Faults: %d\n", page\_faults);

}

int main() {

int pages[MAX\_PAGES];

int num\_pages, num\_frames;

int i, choice;

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

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

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

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

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

for (i = 0; i < num\_pages; i++) {

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

}

printf("\nChoose Page Replacement Algorithm:\n");

printf("1. FIFO\n");

printf("2. LRU\n");

printf("Enter your choice (1 or 2): ");

scanf("%d", &choice);

switch (choice) {

case 1:

fifoPageReplacement(pages, num\_pages, num\_frames);

break;

case 2:

lruPageReplacement(pages, num\_pages, num\_frames);

break;

default:

printf("Invalid choice.\n");

break;

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

braham@braham:~/Desktop/program$ gcc page.c -o page

braham@braham:~/Desktop/program$ ./page

Enter the number of pages: 10

Enter the number of frames: 4

Enter the page reference string:

7 0 1 2 0 3 0 4 2 3

Choose Page Replacement Algorithm:

1. FIFO

2. LRU

Enter your choice (1 or 2): 1

FIFO Page Replacement Algorithm

Page Fault: 7

Page Fault: 7 0

Page Fault: 7 0 1

Page Fault: 7 0 1 2

Page Fault: 3 0 1 2

Page Fault: 3 4 1 2

Total Page Faults: 6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

braham@braham:~/Desktop/program$ gcc page.c -o page

braham@braham:~/Desktop/program$ ./page

Enter the number of pages: 10

Enter the number of frames: 3

Enter the page reference string:

7 0 1 2 0 3 0 4 2 3

Choose Page Replacement Algorithm:

1. FIFO

2. LRU

Enter your choice (1 or 2): 2

LRU Page Replacement Algorithm

Page Fault: 7

Page Fault: 0

Page Fault: 0 1

Page Fault: 0 1 2

Page Fault: 0 3 2

Page Fault: 0 3 4

Page Fault: 0 2 4

Page Fault: 3 2 4

Total Page Faults: 8

Program 8

#include <stdio.h>

#include <string.h>

#define MAX\_FILES 100

#define MAX\_DIRECTORIES 10

#define MAX\_FILES\_PER\_DIR 10

#define MAX\_FILENAME\_LENGTH 50

#define MAX\_DIRECTORY\_NAME 50

typedef struct {

char name[MAX\_FILENAME\_LENGTH];

char content[1000];

} File;

typedef struct {

char name[MAX\_DIRECTORY\_NAME];

File files[MAX\_FILES\_PER\_DIR];

int fileCount;

} Directory;

void listFiles(File files[], int fileCount);

void createFile(File files[], int \*fileCount);

void readFile(File files[], int fileCount);

void listDirectories(Directory directories[], int dirCount);

void listFilesInDirectory(Directory directories[], int dirCount);

void createFileInDirectory(Directory directories[], int dirCount);

void readFileFromDirectory(Directory directories[], int dirCount);

int main() {

File singleLevelFiles[MAX\_FILES];

Directory twoLevelDirectories[MAX\_DIRECTORIES];

int singleLevelFileCount = 0;

int twoLevelDirCount = 0;

int choice;

while (1) {

printf("\nChoose Directory Simulation\n");

printf("1. Single Level Directory\n");

printf("2. Two Level Directory\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar();

switch (choice) {

case 1:

while (1) {

printf("\nSingle Level Directory Menu\n");

printf("1. List Files\n");

printf("2. Create File\n");

printf("3. Read File\n");

printf("4. Go Back\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar();

switch (choice) {

case 1:

listFiles(singleLevelFiles, singleLevelFileCount);

break;

case 2:

createFile(singleLevelFiles, &singleLevelFileCount);

break;

case 3:

readFile(singleLevelFiles, singleLevelFileCount);

break;

case 4:

break;

default:

printf("Invalid choice. Try again.\n");

}

if (choice == 4) break;

}

break;

case 2:

while (1) {

printf("\nTwo Level Directory Menu\n");

printf("1. List Directories\n");

printf("2. List Files in Directory\n");

printf("3. Create File in Directory\n");

printf("4. Read File from Directory\n");

printf("5. Go Back\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar();

switch (choice) {

case 1:

listDirectories(twoLevelDirectories, twoLevelDirCount);

break;

case 2:

listFilesInDirectory(twoLevelDirectories, twoLevelDirCount);

break;

case 3:

createFileInDirectory(twoLevelDirectories, twoLevelDirCount);

break;

case 4:

readFileFromDirectory(twoLevelDirectories, twoLevelDirCount);

break;

case 5:

break;

default:

printf("Invalid choice. Try again.\n");

}

if (choice == 5) break;

}

break;

case 3:

return 0;

default:

printf("Invalid choice. Try again.\n");

}

}

}

void listFiles(File files[], int fileCount) {

if (fileCount == 0) {

printf("No files to list.\n");

return;

}

printf("Files in directory:\n");

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

printf("%d. %s\n", i + 1, files[i].name);

}

}

void createFile(File files[], int \*fileCount) {

if (\*fileCount >= MAX\_FILES) {

printf("Directory is full. Cannot create more files.\n");

return;

}

File newFile;

printf("Enter file name: ");

fgets(newFile.name, MAX\_FILENAME\_LENGTH, stdin);

newFile.name[strcspn(newFile.name, "\n")] = '\0';

printf("Enter file content: ");

fgets(newFile.content, sizeof(newFile.content), stdin);

newFile.content[strcspn(newFile.content, "\n")] = '\0';

files[\*fileCount] = newFile;

(\*fileCount)++;

printf("File created successfully.\n");

}

void readFile(File files[], int fileCount) {

int index;

listFiles(files, fileCount);

printf("Enter the file number to read: ");

scanf("%d", &index);

getchar();

if (index < 1 || index > fileCount) {

printf("Invalid file number.\n");

return;

}

printf("Content of file '%s':\n%s\n", files[index - 1].name, files[index - 1].content);

}

void listDirectories(Directory directories[], int dirCount) {

if (dirCount == 0) {

printf("No directories to list.\n");

return;

}

printf("Directories:\n");

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

printf("%d. %s\n", i + 1, directories[i].name);

}

}

void listFilesInDirectory(Directory directories[], int dirCount) {

int dirIndex;

listDirectories(directories, dirCount);

printf("Enter the directory number to list files: ");

scanf("%d", &dirIndex);

getchar();

if (dirIndex < 1 || dirIndex > dirCount) {

printf("Invalid directory number.\n");

return;

}

Directory dir = directories[dirIndex - 1];

if (dir.fileCount == 0) {

printf("No files in this directory.\n");

return;

}

printf("Files in directory '%s':\n", dir.name);

for (int i = 0; i < dir.fileCount; ++i) {

printf("%d. %s\n", i + 1, dir.files[i].name);

}

}

void createFileInDirectory(Directory directories[], int dirCount) {

int dirIndex;

listDirectories(directories, dirCount);

printf("Enter the directory number to create a file: ");

scanf("%d", &dirIndex);

getchar();

if (dirIndex < 1 || dirIndex > dirCount) {

printf("Invalid directory number.\n");

return;

}

Directory \*dir = &directories[dirIndex - 1];

if (dir->fileCount >= MAX\_FILES\_PER\_DIR) {

printf("Directory is full. Cannot create more files.\n");

return;

}

File newFile;

printf("Enter file name: ");

fgets(newFile.name, MAX\_FILENAME\_LENGTH, stdin);

newFile.name[strcspn(newFile.name, "\n")] = '\0';

printf("Enter file content: ");

fgets(newFile.content, sizeof(newFile.content), stdin);

newFile.content[strcspn(newFile.content, "\n")] = '\0';

dir->files[dir->fileCount] = newFile;

dir->fileCount++;

printf("File created successfully in directory '%s'.\n", dir->name);

}

void readFileFromDirectory(Directory directories[], int dirCount) {

int dirIndex, fileIndex;

listDirectories(directories, dirCount);

printf("Enter the directory number to read a file: ");

scanf("%d", &dirIndex);

getchar();

if (dirIndex < 1 || dirIndex > dirCount) {

printf("Invalid directory number.\n");

return;

}

Directory dir = directories[dirIndex - 1];

if (dir.fileCount == 0) {

printf("No files in this directory.\n");

return;

}

listFilesInDirectory(directories, dirCount);

printf("Enter the file number to read: ");

scanf("%d", &fileIndex);

getchar();

if (fileIndex < 1 || fileIndex > dir.fileCount) {

printf("Invalid file number.\n");

return;

}

printf("Content of file '%s':\n%s\n", dir.files[fileIndex - 1].name, dir.files[fileIndex - 1].content);

}

Out put

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

braham@braham:~/Desktop/program$ gcc directory.c -o directory

braham@braham:~/Desktop/program$ ./directory

Choose Directory Simulation

1. Single Level Directory

2. Two Level Directory

3. Exit

Enter your choice: 1

Single Level Directory Menu

1. List Files

2. Create File

3. Read File

4. Go Back

Enter your choice: 2

Enter file name: file1.txt

Enter file content: Hello, this is file 1.

File created successfully.

Single Level Directory Menu

1. List Files

2. Create File

3. Read File

4. Go Back

Enter your choice: 1

Files in directory:

1. file1.txt

Single Level Directory Menu

1. List Files

2. Create File

3. Read File

4. Go Back

Enter your choice: 3

Enter the file number to read: 1

Content of file 'file1.txt':

Hello, this is file 1.

Single Level Directory Menu

1. List Files

2. Create File

3. Read File

4. Go Back

Enter your choice: 4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

braham@braham:~/Desktop/program$ gcc directory.c -o directory

braham@braham:~/Desktop/program$ ./directory

Choose Directory Simulation

1. Single Level Directory

2. Two Level Directory

3. Exit

Enter your choice: 2

Two Level Directory Menu

1. List Directories

2. List Files in Directory

3. Create File in Directory

4. Read File from Directory

5. Go Back

Enter your choice: 1

Directories:

1. dir1

Two Level Directory Menu

1. List Directories

2. List Files in Directory

3. Create File in Directory

4. Read File from Directory

5. Go Back

Enter your choice: 2

Enter the directory number to list files: 1

Files in directory 'dir1':

1. file1.txt

Two Level Directory Menu

1. List Directories

2. List Files in Directory

3. Create File in Directory

4. Read File from Directory

5. Go Back

Enter your choice: 3

Enter the directory number to create a file: 1

Enter file name: file2.txt

Enter file content: Hello, this is file 2 in dir1.

File created successfully in directory 'dir1'.

Two Level Directory Menu

1. List Directories

2. List Files in Directory

3. Create File in Directory

4. Read File from Directory

5. Go Back

Enter your choice: 2

Enter the directory number to list files: 1

Files in directory 'dir1':

1. file1.txt

2. file2.txt

Two Level Directory Menu

1. List Directories

2. List Files in Directory

3. Create File in Directory

4. Read File from Directory

5. Go Back

Enter your choice: 4

Enter the directory number to read a file: 1

Enter the file number to read: 2

Content of file 'file2.txt':

Hello, this is file 2 in dir1.

Two Level Directory Menu

1. List Directories

2. List Files in Directory

3. Create File in Directory

4. Read File from Directory

5. Go Back

Enter your choice: 5

Program 9

#include <stdio.h>

#include <stdlib.h>

#define MAX\_BLOCKS 10

typedef struct Block {

int blockNumber;

int nextBlock;

} Block;

Block disk[MAX\_BLOCKS];

void initializeDisk() {

for (int i = 0; i < MAX\_BLOCKS; i++) {

disk[i].blockNumber = i;

disk[i].nextBlock = -1;

}

}

void allocateBlocks(int startBlock, int numBlocks) {

if (numBlocks <= 0) {

printf("Invalid number of blocks.\n");

return;

}

int current = startBlock;

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

if (current >= MAX\_BLOCKS || disk[current].nextBlock != -1) {

printf("Not enough contiguous blocks available.\n");

return;

}

if (i < numBlocks - 1) {

int next = current + 1;

if (next < MAX\_BLOCKS && disk[next].nextBlock == -1) {

disk[current].nextBlock = next;

current = next;

} else {

printf("Not enough contiguous blocks available.\n");

return;

}

} else {

disk[current].nextBlock = -1;

}

}

printf("Blocks allocated successfully.\n");

}

void displayAllocation() {

printf("Block Allocation:\n");

for (int i = 0; i < MAX\_BLOCKS; i++) {

printf("Block %d: Next Block = %d\n", disk[i].blockNumber, disk[i].nextBlock);

}

}

int main() {

int startBlock, numBlocks;

initializeDisk();

printf("Enter starting block number for allocation: ");

scanf("%d", &startBlock);

printf("Enter number of blocks to allocate: ");

scanf("%d", &numBlocks);

if (startBlock >= MAX\_BLOCKS || startBlock < 0) {

printf("Invalid starting block number.\n");

return 1;

}

allocateBlocks(startBlock, numBlocks);

displayAllocation();

return 0;

}

Output

braham@braham:~/Desktop/program$ gcc file.c -o file

braham@braham:~/Desktop/program$ ./file

Enter starting block number for allocation: 2

Enter number of blocks to allocate: 4

Blocks allocated successfully.

Block Allocation:

Block 0: Next Block = -1

Block 1: Next Block = -1

Block 2: Next Block = 3

Block 3: Next Block = 4

Block 4: Next Block = 5

Block 5: Next Block = -1

Block 6: Next Block = -1

Block 7: Next Block = -1

Block 8: Next Block = -1

Block 9: Next Block = -1

Program 10

#include <stdio.h>

#include <stdlib.h>

#define MAX\_REQUESTS 100

void sort(int arr[], int n) {

int i, j, temp;

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

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

if (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

void scanAlgorithm(int requests[], int n, int head, int direction, int disk\_size) {

int seek\_sequence[MAX\_REQUESTS];

int index = 0, i;

sort(requests, n);

if (direction == 1) {

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

if (requests[i] >= head) {

seek\_sequence[index++] = requests[i];

}

}

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

if (requests[i] < head) {

seek\_sequence[index++] = requests[i];

}

}

}

else {

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

if (requests[i] <= head) {

seek\_sequence[index++] = requests[i];

}

}

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

if (requests[i] > head) {

seek\_sequence[index++] = requests[i];

}

}

}

printf("Seek Sequence: ");

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

printf("%d ", seek\_sequence[i]);

}

printf("\n");

}

int main() {

int requests[MAX\_REQUESTS];

int n, head, direction, disk\_size;

printf("Enter the number of disk requests: ");

scanf("%d", &n);

if (n > MAX\_REQUESTS) {

printf("Number of requests exceeds the maximum limit of %d.\n", MAX\_REQUESTS);

return 1;

}

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

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

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

}

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

scanf("%d", &head);

printf("Enter the direction of head movement (1 for right, 0 for left): ");

scanf("%d", &direction);

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

scanf("%d", &disk\_size);

scanAlgorithm(requests, n, head, direction, disk\_size);

return 0;

}

Output

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

braham@braham:~/Desktop/program$ gcc disk.c -o disk

braham@braham:~/Desktop/program$ ./disk

Enter the number of disk requests: 8

Enter the disk requests:

98 183 37 122 14 124 65 67

Enter the initial head position: 50

Enter the direction of head movement (1 for right, 0 for left): 1

Enter the size of the disk: 200

Seek Sequence: 65 67 98 122 124 183 37 14

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

braham@braham:~/Desktop/program$ gcc disk.c -o disk

braham@braham:~/Desktop/program$ ./disk

Enter the number of disk requests: 8

Enter the disk requests:

98 183 37 122 14 124 65 67

Enter the initial head position: 50

Enter the direction of head movement (1 for right, 0 for left): 0

Enter the size of the disk: 200

Seek Sequence: 37 14 65 67 98 122 124 183