**LAB EXPERIMENTS**

**21.** #include <stdio.h>

void worstFit(int blocks[], int m, int processes[], int n) {

int allocation[n];

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

allocation[i] = -1;

}

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

int worstIdx = -1;

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

if (blocks[j] >= processes[i]) {

if (worstIdx == -1 || blocks[j] > blocks[worstIdx]) {

worstIdx = j;

}

}

}

if (worstIdx != -1) {

allocation[i] = worstIdx;

blocks[worstIdx] -= processes[i];

}

}

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

if (allocation[i] != -1) {

printf("Process %d allocated to block %d\n", i + 1, allocation[i] + 1);

} else {

printf("Process %d not allocated\n", i + 1);

}

}

}

int main() {

int blocks[] = {100, 500, 200, 300, 600};

int processes[] = {212, 417, 112, 426};

int m = sizeof(blocks) / sizeof(blocks[0]);

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

worstFit(blocks, m, processes, n);

return 0;

}

**OUTPUT:**

Process 1 allocated to block 2

Process 2 allocated to block 5

Process 3 allocated to block 3

Process 4 allocated to block 4

**22.** #include <stdio.h>

void bestFit(int blocks[], int m, int processes[], int n) {

int allocation[n];

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

allocation[i] = -1;

}

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

int bestIdx = -1;

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

if (blocks[j] >= processes[i]) {

if (bestIdx == -1 || blocks[j] < blocks[bestIdx]) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blocks[bestIdx] -= processes[i];

}

}

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

if (allocation[i] != -1) {

printf("Process %d allocated to block %d\n", i + 1, allocation[i] + 1);

} else {

printf("Process %d not allocated\n", i + 1);

}

}

}

int main() {

int blocks[] = {100, 500, 200, 300, 600};

int processes[] = {212, 417, 112, 426};

int m = sizeof(blocks) / sizeof(blocks[0]);

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

bestFit(blocks, m, processes, n);

return 0;

}

**OUTPUT:**

Process 1 allocated to block 3

Process 2 allocated to block 5

Process 3 allocated to block 1

Process 4 allocated to block 4

**23.** #include <stdio.h>

void firstFit(int blocks[], int m, int processes[], int n) {

int allocation[n];

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

allocation[i] = -1;

}

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

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

if (blocks[j] >= processes[i]) {

allocation[i] = j;

blocks[j] -= processes[i];

break;

}

}

}

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

if (allocation[i] != -1) {

printf("Process %d allocated to block %d\n", i + 1, allocation[i] + 1);

} else {

printf("Process %d not allocated\n", i + 1);

}

}

}

int main() {

int blocks[] = {100, 500, 200, 300, 600};

int processes[] = {212, 417, 112, 426};

int m = sizeof(blocks) / sizeof(blocks[0]);

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

firstFit(blocks, m, processes, n);

return 0;

}

**OUTPUT:**

Process 1 allocated to block 2

Process 2 allocated to block 5

Process 3 allocated to block 1

Process 4 allocated to block 4

**24.** #include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd;

char buffer[100];

fd = open("file.txt", O\_CREAT | O\_RDWR, 0644);

if (fd < 0) {

perror("File open error");

return 1;

}

write(fd, "Hello, UNIX!", 12);

lseek(fd, 0, SEEK\_SET);

read(fd, buffer, 12);

buffer[12] = '\0';

printf("Read from file: %s\n", buffer);

close(fd);

return 0;

}

**OUTPUT:** Read from file: Hello, UNIX!

**25.** #include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

int main() {

int fd;

struct stat st;

DIR \*dir;

struct dirent \*entry;

fd = open("file.txt", O\_RDWR);

if (fd < 0) {

perror("open");

return 1;

}

fcntl(fd, F\_SETFL, O\_APPEND); // Set file to append mode

write(fd, "\nAppended line.", 15);

lseek(fd, 0, SEEK\_SET);

fstat(fd, &st);

printf("File size: %ld bytes\n", st.st\_size);

close(fd);

dir = opendir(".");

if (dir == NULL) {

perror("opendir");

return 1;

}

printf("Directory contents:\n");

while ((entry = readdir(dir)) != NULL) {

printf(" %s\n", entry->d\_name);

}

closedir(dir);

return 0;

}

**OUTPUT:**

File size: 28 bytes

Directory contents:

file.txt

**26.** #include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*fp = fopen("demo.txt", "w");

if (!fp) {

perror("Create");

return 1;

}

fprintf(fp, "File management example.\n");

fclose(fp);

fp = fopen("demo.txt", "r");

if (!fp) {

perror("Read");

return 1;

}

char ch;

printf("File content:\n");

while ((ch = fgetc(fp)) != EOF)

putchar(ch);

fclose(fp);

if (remove("demo.txt") == 0)

printf("\nFile deleted successfully.\n");

else

perror("Delete");

return 0;

}

**OUTPUT:**

File content:

File management example.

File deleted successfully.

**27.** #include <stdio.h>

#include <dirent.h>

int main(int argc, char \*argv[]) {

DIR \*d;

struct dirent \*dir;

char \*path = ".";

if (argc > 1)

path = argv[1];

d = opendir(path);

if (!d) {

perror("opendir");

return 1;

}

while ((dir = readdir(d)) != NULL) {

printf("%s ", dir->d\_name);

}

printf("\n");

closedir(d);

return 0;

}

**OUTPUT:**

. .. file.txt ls.c a.out

**28**. #include <stdio.h>

#include <string.h>

int main(int argc, char \*argv[]) {

if (argc != 3) {

printf("Usage: %s <pattern> <file>\n", argv[0]);

return 1;

}

FILE \*fp = fopen(argv[2], "r");

if (!fp) {

perror("fopen");

return 1;

}

char line[256];

while (fgets(line, sizeof(line), fp)) {

if (strstr(line, argv[1])) {

printf("%s", line);

}

}

fclose(fp);

return 0;

}

**OUTPUT:**

This is a sample file.

We are learning grep.

UNIX system programming.

**29.** #include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10;

int buffer[10];

int in = 0, out = 0;

void wait(int \*s) {

while (\*s <= 0);

(\*s)--;

}

void signal(int \*s) {

(\*s)++;

}

void producer(int item) {

wait(&empty);

wait(&mutex);

buffer[in] = item;

in = (in + 1) % 10;

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

signal(&mutex);

signal(&full);

}

void consumer() {

int item;

wait(&full);

wait(&mutex);

item = buffer[out];

out = (out + 1) % 10;

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

signal(&mutex);

signal(&empty);

}

int main() {

for (int i = 0; i < 5; i++) {

producer(i + 1);

}

for (int i = 0; i < 5; i++) {

consumer();

}

return 0;

}

**OUTPUT:**

Produced: 1

Produced: 2

Produced: 3

Produced: 4

Produced: 5

Consumed: 1

Consumed: 2

Consumed: 3

Consumed: 4

Consumed: 5

**30.** #include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

void\* print\_hello(void\* thread\_id) {

long tid = (long) thread\_id;

printf("Hello from thread %ld!\n", tid);

pthread\_exit(NULL); // Exit the thread after completing its task

}

int main() {

pthread\_t thread1, thread2

if (pthread\_create(&thread1, NULL, print\_hello, (void\*) t1)) {

printf("Error creating thread 1\n");

return 1;

}

// Create thread 2

if (pthread\_create(&thread2, NULL, print\_hello, (void\*) t2)) {

printf("Error creating thread 2\n");

return 1;

}

// Join thread 1 to the main thread

if (pthread\_join(thread1, NULL)) {

printf("Error joining thread 1\n");

return 1;

}

// Join thread 2 to the main thread

if (pthread\_join(thread2, NULL)) {

printf("Error joining thread 2\n");

return 1;

}

// Check if thread1 and thread2 are equal

if (pthread\_equal(thread1, thread2)) {

printf("Thread 1 and Thread 2 are the same thread.\n");

} else {

printf("Thread 1 and Thread 2 are different threads.\n");

}

// Exit main thread

pthread\_exit(NULL);

return 0;

}

**OUTPUT:**

Hello from thread 1!

Hello from thread 2!

Thread 1 and Thread 2 are different threads.