**21. Develop a C program to implement worst fit algorithm of memory management**

#include <stdio.h>

// Function to allocate memory to blocks as per worst fit algorithm

void worstFit(int blockSize[], int blocks, int processSize[], int processes) {

// Stores block id of the block allocated to a process

int allocation[processes];

// Initially no block is assigned to any process

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

allocation[i] = -1;

}

// Pick each process and find suitable blocks according to worst fit algorithm

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

// Find the worst fit block for current process

int wstIdx = -1;

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

if (blockSize[j] >= processSize[i]) {

if (wstIdx == -1 || blockSize[j] > blockSize[wstIdx]) {

wstIdx = j;

}

}

}

// If a block was found for current process

if (wstIdx != -1) {

// Allocate block j to process i

allocation[i] = wstIdx;

// Reduce available memory in this block

blockSize[wstIdx] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

printf("%d\t\t%d\t\t", i + 1, processSize[i]);

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

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

} else {

printf("Not Allocated\n");

}

}

}

int main() {

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

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

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

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

worstFit(blockSize, blocks, processSize, processes);

return 0;

}

**Output**

Process No. Process Size Block no.

1 212 5

2 417 2

3 112 5

4 426 Not Allocated

**22 .Construct a C program to implement best fit algorithm of memory management**

#include <stdio.h>

void bestFit(int blockSize[], int m, int processSize[], 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 (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j])

bestIdx = j;

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

printf("Process No.\tBlock No.\n");

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

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

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

int main() {

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

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

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

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

bestFit(blockSize, m, processSize, n);

return 0;

}

**Output**

Process No. Block No.

1 4

2 2

3 3

4 Not Allocated

**23. Construct a C program to implement first fit algorithm of memory management.**

#include <stdio.h>

// Function to allocate memory to blocks as per first fit algorithm

void firstFit(int blockSize[], int blocks, int processSize[], int processes) {

// Stores block id of the block allocated to a process

int allocation[processes];

// Initially no block is assigned to any process

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

allocation[i] = -1;

}

// Pick each process and find the first suitable block according to first fit algorithm

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

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

if (blockSize[j] >= processSize[i]) {

// Allocate block j to process i

allocation[i] = j;

// Reduce available memory in this block

blockSize[j] -= processSize[i];

break; // Move to the next process

}

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

printf("%d\t\t%d\t\t", i + 1, processSize[i]);

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

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

} else {

printf("Not Allocated\n");

}

}

}

int main() {

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

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

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

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

firstFit(blockSize, blocks, processSize, processes);

return 0;

}

**Output**

Process No. Process Size Block no.

1 212 2

2 417 5

3 112 2

4 426 Not Allocated

**24. Design a C program to demonstrate UNIX system calls for file management.**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd;

char buffer[100];

// Open a file for reading and writing

fd = open("example.txt", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

perror("Error opening file");

return 1;

}

// Write to the file

const char \*text = "Hello, UNIX system calls!";

ssize\_t bytesWritten = write(fd, text, sizeof(text));

if (bytesWritten == -1) {

perror("Error writing to file");

close(fd);

return 1;

}

// Move the file pointer to the beginning

if (lseek(fd, 0, SEEK\_SET) == -1) {

perror("Error seeking in file");

close(fd);

return 1;

}

// Read from the file

ssize\_t bytesRead = read(fd, buffer, sizeof(buffer) - 1);

if (bytesRead == -1) {

perror("Error reading file");

close(fd);

return 1;

}

// Null-terminate the buffer and print it

buffer[bytesRead] = '\0';

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

// Close the file

if (close(fd) == -1) {

perror("Error closing file");

return 1;

}

return 0;

}

**Output**

Read from file: Hello, UNIX system calls!

**25. Construct a C program to implement the I/O system calls of UNIX (fcntl, seek, stat, opendir, readdir)**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

int main() {

int fd;

struct stat statbuf;

struct dirent \*entry;

DIR \*dir;

// Open a file for reading and writing

fd = open("example.txt", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

perror("Error opening file");

return 1;

}

// Write to the file

const char \*text = "Hello, UNIX system calls!";

ssize\_t bytesWritten = write(fd, text, sizeof(text));

if (bytesWritten == -1) {

perror("Error writing to file");

close(fd);

return 1;

}

// Move the file pointer to the beginning using lseek

if (lseek(fd, 0, SEEK\_SET) == -1) {

perror("Error seeking in file");

close(fd);

return 1;

}

// Get file status using stat

if (stat("example.txt", &statbuf) == -1) {

perror("Error getting file status");

close(fd);

return 1;

}

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

printf("File permissions: %o\n", statbuf.st\_mode & 0777);

// Use fcntl to get file status flags

int flags = fcntl(fd, F\_GETFL);

if (flags == -1) {

perror("Error getting file flags");

close(fd);

return 1;

}

printf("File flags: %d\n", flags);

// Close the file

if (close(fd) == -1) {

perror("Error closing file");

return 1;

}

// Open a directory

dir = opendir(".");

if (dir == NULL) {

perror("Error opening directory");

return 1;

}

printf("\nDirectory contents:\n");

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

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

}

// Close the directory

if (closedir(dir) == -1) {

perror("Error closing directory");

return 1;

}

return 0;

}

**Output**

File size: 27 bytes

File permissions: 644

File flags: 2

Directory contents:

.

..

example.txt

io\_system\_calls

**26. Construct a C program to implement the file management operations**

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*file;

// Open a file for writing

file = fopen("example.txt", "w");

if (file == NULL) {

printf("Error opening the file for writing.\n");

return 1;

}

// Write to the file

fprintf(file, "Hello, World!\n");

fprintf(file, "This is a C file management example.\n");

fclose(file);

// Open the file for reading

file = fopen("example.txt", "r");

if (file == NULL) {

printf("Error opening the file for reading.\n");

return 1;

}

// Read from the file

char buffer[100];

while (fgets(buffer, sizeof(buffer), file) != NULL) {

printf("%s", buffer);

}

fclose(file);

return 0;

}

**Output**

Hello, World!

This is a C file management example.

**27. Develop a C program for simulating the function of ls UNIX Command**.

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

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

struct dirent \*entry;

DIR \*dp;

const char \*path = "."; // Default to current directory

if (argc > 1) {

path = argv[1]; // Use provided path if available

}

dp = opendir(path);

if (dp == NULL) {

perror("opendir");

return 1;

}

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

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

}

closedir(dp);

return 0;

}

**Output**

Enter file name: hello

Enter the pattern: world

Error opening file: No such file or directory

**28. Write a C program for simulation of GREP UNIX command.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LINE\_LENGTH 1024

void searchFile(const char \*pattern, const char \*filename) {

FILE \*file = fopen(filename, "r");

if (file == NULL) {

perror("Error opening file");

exit(1);

}

char line[MAX\_LINE\_LENGTH];

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

if (strstr(line, pattern) != NULL) {

printf("%s", line);

}

}

fclose(file);

}

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

if (argc != 3) {

fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);

return 1;

}

const char \*pattern = argv[1];

const char \*filename = argv[2];

searchFile(pattern, filename);

return 0;

}

**Output**

Hello, World!

This is a C file management example.

**29.** **Write a C program to simulate the solution of Classical Process Synchronization Problem**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 5

int buffer[BUFFER\_SIZE];

int in = 0, out = 0;

sem\_t empty, full;

pthread\_mutex\_t mutex;

void \*producer(void \*arg) {

int item;

while (1) {

item = rand() % 100;

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

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

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

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void \*consumer(void \*arg) {

int item;

while (1) {

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

item = buffer[out];

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

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

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main() {

pthread\_t prod, cons;

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

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

pthread\_mutex\_init(&mutex, NULL);

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

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

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

sem\_destroy(&empty);

sem\_destroy(&full);

pthread\_mutex\_destroy(&mutex);

return 0;

}

**Output**

Produced: 42

Consumed: 42

Produced: 7

Consumed: 7

**30. Write C programs to demonstrate the following thread related concepts.**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

void\* func(void\* arg) {

printf("Inside the thread\n");

pthread\_exit(NULL);

}

void fun() {

pthread\_t ptid;

pthread\_create(&ptid, NULL, func, NULL);

printf("This line may be printed before thread terminates\n");

if (pthread\_equal(ptid, pthread\_self())) {

printf("Threads are equal\n");

} else {

printf("Threads are not equal\n");

}

pthread\_join(ptid, NULL); // Wait for the thread to finish

printf("This line will be printed after thread ends\n");

}

int main() {

fun();

return 0;

}

**Output**

This line may be printed before thread terminates

Threads are not equal

Inside the thread

This line will be printed after thread ends