Operating System

Day-3:Programmes.

Name:G.Tejaswi

Reg.No:192211748

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

#include <stdio.h>

#define MAX\_PARTITIONS 10

void worstFit(int partitions[], int numPartitions, int processes[], int numProcesses) {

int allocation[MAX\_PARTITIONS] = {-1};

int i, j;

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

int worstIndex = -1;

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

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

if (worstIndex == -1 || partitions[j] > partitions[worstIndex]) {

worstIndex = j;

}

}

}

if (worstIndex != -1) {

allocation[worstIndex] = i;

partitions[worstIndex] -= processes[i];

}

}

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

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

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

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

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

} else {

printf("Not Allocated\n");

}

}

}

int main() {

int partitions[MAX\_PARTITIONS] = {40, 10, 30, 60};

int numPartitions = 4;

int processes[] = {100, 50, 30, 120, 35};

int numProcesses = 5;

printf("Memory Partitions:\n");

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

printf("Partition %d: %d KB\n", i+1, partitions[i]);

}

printf("\nProcesses:\n");

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

printf("Process %d: %d KB\n", i+1, processes[i]);

}

worstFit(partitions, numPartitions, processes, numProcesses);

return 0;

}

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

#include <stdio.h>

#define MEMORY\_SIZE 1000

typedef struct MemoryBlock

{

int size;

int allocated;

} MemoryBlock;

void initializeMemory(MemoryBlock memory[], int size)

{

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

memory[i].size = 0;

memory[i].allocated = 0;

}

}

void displayMemory(MemoryBlock memory[], int size)

{

printf("Memory:\n");

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

printf("Block %d: Size = %d, Allocated = %s\n", i + 1, memory[i].size,

memory[i].allocated ? "Yes" : "No");

}

printf("\n");

}

void bestFit(MemoryBlock memory[], int size, int requestSize)

{

int bestFitIndex = -1;

int minFragmentation = MEMORY\_SIZE + 1;

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

{

if (!memory[i].allocated && memory[i].size >= requestSize)

{

int fragmentation = memory[i].size - requestSize;

if (fragmentation < minFragmentation)

{

bestFitIndex = i;

minFragmentation = fragmentation;

}

}

}

if (bestFitIndex != -1)

{

memory[bestFitIndex].allocated = 1;

printf("Memory allocated successfully in Block %d\n", bestFitIndex + 1);

} else {

printf("Memory allocation failed. No suitable block found.\n");

}

}

int main()

{

MemoryBlock memory[MEMORY\_SIZE];

initializeMemory(memory, MEMORY\_SIZE);

displayMemory(memory, MEMORY\_SIZE);

bestFit(memory, MEMORY\_SIZE, 150);

bestFit(memory, MEMORY\_SIZE, 300);

bestFit(memory, MEMORY\_SIZE, 100);

bestFit(memory, MEMORY\_SIZE, 200);

displayMemory(memory, MEMORY\_SIZE);

return 0;

}

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

#include <stdio.h>

void firstFit(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++) {

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

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

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

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

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

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

if (allocation[i] != -1)

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

else

printf("Not Allocated");

printf("\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]);

firstFit(blockSize, m, processSize, n);

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fileDescriptor;

fileDescriptor = open("example\_file.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

if (fileDescriptor == -1) {

perror("Error opening file");

exit(EXIT\_FAILURE);

}

char data[] = "Hello, this is a demonstration of UNIX system calls for file management.\n";

ssize\_t bytesWritten = write(fileDescriptor, data, sizeof(data) - 1);

if (bytesWritten == -1) {

perror("Error writing to file");

close(fileDescriptor);

exit(EXIT\_FAILURE);

}

printf("Data written to the file.\n");

close(fileDescriptor);

fileDescriptor = open("example\_file.txt", O\_RDONLY);

if (fileDescriptor == -1) {

perror("Error opening file for reading");

exit(EXIT\_FAILURE);

}

char buffer[1024];

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

if (bytesRead == -1) {

perror("Error reading from file");

close(fileDescriptor);

exit(EXIT\_FAILURE);

}

buffer[bytesRead] = '\0';

printf("Data read from the file:\n%s", buffer);

close(fileDescriptor);

return 0;

}

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

readdir).

#include <stdio.h>

#define BLOCK\_SIZE 1

#define FILE\_SIZE 100

int contiguous() {

int i;

int numOperations = 0;

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

numOperations++;

}

return numOperations;

}

int linked() {

int i;

int numOperations = 0;

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

numOperations++;

numOperations++;

}

return numOperations;

}

int indexed() {

int numOperations = 0;

numOperations++;

numOperations += FILE\_SIZE;

return numOperations;

}

int main() {

printf("Contiguous allocation: %d disk I/O operations\n", contiguous());

printf("Linked allocation: %d disk I/O operations\n", linked());

printf("Indexed allocation: %d disk I/O operations\n", indexed());

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

void createFile(const char \*filename, const char \*content)

{

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

if (file == NULL) {

perror("Error creating file");

exit(EXIT\_FAILURE);

}

fprintf(file, "%s", content);

fclose(file);

}

void readFile(const char \*filename)

{

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

if (file == NULL) {

perror("Error opening file for reading");

exit(EXIT\_FAILURE);

}

char buffer[1024];

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

printf("%s", buffer);

}

fclose(file);

}

void appendToFile(const char \*filename, const char \*content) {

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

if (file == NULL) {

perror("Error opening file for appending");

exit(EXIT\_FAILURE);

}

fprintf(file, "%s", content);

fclose(file);

}

int main()

{

const char \*filename = "example\_file.txt";

const char \*initialContent = "This is an example file.\n";

createFile(filename, initialContent);

printf("File created with initial content.\n");

printf("File content:\n");

readFile(filename);

const char \*additionalContent = "Appending more content to the file.\n";

appendToFile(filename, additionalContent);

printf("Content appended to the file.\n");

printf("File content after appending:\n");

readFile(filename);

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

int main() {

DIR \*directory;

struct dirent \*entry;

directory = opendir(".");

if (directory == NULL)

{

perror("Error opening directory");

exit(EXIT\_FAILURE);

}

printf("Files and directories in the current directory:\n");

while ((entry = readdir(directory)) != NULL)

{

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

}

closedir(directory);

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LINE\_LENGTH 256

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

{

if (argc != 2) {

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

exit(EXIT\_FAILURE);

}

char pattern[MAX\_LINE\_LENGTH];

strcpy(pattern, argv[1]);

char line[MAX\_LINE\_LENGTH];

while (fgets(line, sizeof(line), stdin) != NULL)

{

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

printf("%s", line);

}

}

return 0;

}

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>

#include <unistd.h>

#define BUFFER\_SIZE 5

int buffer[BUFFER\_SIZE];

sem\_t mutex, empty, full;

int in = 0, out = 0;

void \*producer(void \*arg)

{

int item;

for (int i = 0; i < 10; i++)

{

item = rand() % 100;

sem\_wait(&empty);

sem\_wait(&mutex);

buffer[in] = item;

printf("Producer produced item: %d at index %d\n", item, in);

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

sem\_post(&mutex);

sem\_post(&full);

sleep(1);

}

pthread\_exit(NULL);

}

void \*consumer(void \*arg)

{

int item;

for (int i = 0; i < 10; i++)

{

sem\_wait(&full);

sem\_wait(&mutex);

item = buffer[out];

printf("Consumer consumed item: %d from index %d\n", item, out);

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

sem\_post(&mutex);

sem\_post(&empty);

sleep(2);

}

pthread\_exit(NULL);

}

int main()

{

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

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

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

pthread\_t producerThread, consumerThread;

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

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

pthread\_join(producerThread, NULL);

pthread\_join(consumerThread, NULL);

sem\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

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

(i) create (ii) join (iii) equal (iv) exit.

#include <stdio.h>

#include <pthread.h>

void \*myThreadFun(void \*vargp) {

printf("Teja\n");

return NULL;

}

int main()

{

pthread\_t thread\_id;

printf("Before Thread\n");

pthread\_create(&thread\_id, NULL, myThreadFun, NULL);

pthread\_join(thread\_id, NULL);

printf("After Thread\n");

if (pthread\_equal(pthread\_self(), thread\_id)) {

printf("The main thread is equal to the created thread.\n");

} else {

printf("The main thread is not equal to the created thread.\n");

}

exit(0);

return 0;

}