Sudharsan .S

192321077

CSA0428 Operating Systems for UI Design

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

**AIM:**

To develop a C program to allocate memory to processes using the Worst Fit memory allocation strategy.

**ALGORITHM:**

1. Initialize memory blocks and process sizes.
2. For each process:

 a. Find the largest block that fits.

 b. Allocate memory and reduce block size.

1. Display allocation result.

**CODE:**

#include <stdio.h>

int main() {

int b[] = {100, 500, 200, 300, 600}, p[] = {212, 417, 112, 426}, a[4], i, j, k;

for (i = 0; i < 4; i++) {

a[i] = -1; k = -1;

for (j = 0; j < 5; j++)

if (b[j] >= p[i] && (k == -1 || b[j] > b[k])) k = j;

if (k != -1) { a[i] = k; b[k] -= p[i]; }

}

for (i = 0; i < 4; i++)

printf("P%d -> %s\n", i+1, a[i] != -1 ? "Block Found" : "Not Allocated");

return 0;

}

**SAMPLE OUTPUT:**

Process 1 -> Block 5

Process 2 -> Block 2

Process 3 -> Block 5

Process 4 -> Not Allocated

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

Worst fit memory allocation program executed successfully.

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

**AIM:**

To implement a C program that allocates memory to processes using the Best Fit memory allocation strategy.

**ALGORITHM:**

1. For each process:

 a. Find the smallest suitable block.

 b. Allocate memory and reduce that block.

1. Display allocation result.

**CODE:**

#include <stdio.h>

int main() {

int b[] = {100, 500, 200, 300, 600}, p[] = {212, 417, 112, 426}, a[4], i, j, k;

for (i = 0; i < 4; i++) {

a[i] = -1; k = -1;

for (j = 0; j < 5; j++)

if (b[j] >= p[i] && (k == -1 || b[j] < b[k])) k = j;

if (k != -1) { a[i] = k; b[k] -= p[i]; }

}

for (i = 0; i < 4; i++)

printf("P%d -> %s\n", i+1, a[i] != -1 ? "Block Found" : "Not Allocated");

return 0;

}

**SAMPLE OUTPUT:**

P1 -> Block Found

P2 -> Block Found

P3 -> Block Found

P4 -> Block Found

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

Best fit memory allocation program executed successfully.

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

**AIM:**

To write a C program that allocates memory to processes using the First Fit memory allocation strategy.

**ALGORITHM:**

1. For each process:

 a. Scan memory blocks from the beginning.

 b. Allocate the first block that fits.

1. Show result.

**CODE:**

#include <stdio.h>

int main() {

int b[] = {100, 500, 200, 300, 600}, p[] = {212, 417, 112, 426}, a[4], i, j;

for (i = 0; i < 4; i++) {

a[i] = -1;

for (j = 0; j < 5; j++) {

if (b[j] >= p[i]) {

a[i] = j;

b[j] -= p[i];

break;

}

}

}

for (i = 0; i < 4; i++)

printf("P%d -> %s\n", i + 1, a[i] != -1 ? "Block Found" : "Not Allocated");

return 0;

}

**SAMPLE OUTPUT:**

P1 -> Block Found

P2 -> Block Found

P3 -> Block Found

P4 -> Not Allocated

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

First fit memory allocation program executed successfully.

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

**AIM:**

To design a C program demonstrating UNIX file management system calls like open(), read(), write(), and close()

**ALGORITHM:**

1. Open/create a file using open().
2. Write data using write().
3. Use lseek() to reset pointer.
4. Read data using read().
5. Close file using close().

**CODE:**

#include <stdio.h>

int main() {

FILE \*fp = fopen("file.txt", "w+"); // create or open file for read/write

fprintf(fp, "Hello"); // write to file

rewind(fp); // move pointer to beginning

char str[10];

fscanf(fp, "%s", str); // read from file

printf("%s\n", str); // print read data

fclose(fp); // close file

return 0;

}

**SAMPLE OUTPUT:**

Hello



**RESULT:**

File management using UNIX system calls executed successfully.

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

**AIM:**

To implement a C program that demonstrates UNIX I/O system calls like fcntl, lseek, stat, opendir, and readdir.

**ALGORITHM:**

1. Create file using open()
2. Write data, seek with lseek()
3. Get file info using stat()
4. List directory using opendir() + readdir()

**CODE:**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

int main() {

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

write(fd, "Hello", 5);

struct stat st;

stat("demo.txt", &st);

printf("Size: %ld\n", st.st\_size);

DIR \*d = opendir(".");

struct dirent \*e;

while ((e = readdir(d)))

if (e->d\_name[0] != '.') printf("%s\n", e->d\_name);

close(fd); closedir(d);

return 0;

}

**SAMPLE OUTPUT:**

64

**A screenshot of a computer

AI-generated content may be incorrect.**

**RESULT:**

UNIX I/O system calls (fcntl, seek, stat, etc.) program executed successfully.

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

**AIM:**

To construct a C program that performs file management operations like create, write, read, and delete a file.

**ALGORITHM:**

1. Create and write to a file using fopen() and fprintf().
2. Read data using fgets().
3. Delete file using remove().
4. Print results.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*fp;

char data[100];

// Create & write

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

if (fp == NULL) { printf("Error creating file\n"); return 1; }

fprintf(fp, "Hello File!");

fclose(fp);

// Read file

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

if (fp == NULL) { printf("Error reading file\n"); return 1; }

fgets(data, 100, fp);

printf("File content: %s\n", data);

fclose(fp);

// Delete file

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

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

else

printf("Error deleting file.\n");

return 0;

}

**SAMPLE OUTPUT:**

File content: Hello File!

File deleted successfully

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

File management operations program executed successfully.

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

**AIM:**

To develop a C program that simulates the basic function of the UNIX ls command, listing visible files in the current directory

**ALGORITHM:**

1. Open current directory with opendir(".")
2. Read entries using readdir()
3. Print non-hidden files (d\_name[0] != '.')
4. Close directory

**CODE:**

#include <stdio.h>

#include <dirent.h>

int main() {

DIR \*d;

struct dirent \*dir;

d = opendir(".");

if (d) {

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

if (dir->d\_name[0] != '.') // Skip hidden files

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

}

closedir(d);

printf("\n");

} else {

printf("Unable to open directory.\n");

}

return 0;

}

**SAMPLE OUTPUT:**

main.c file1.txt demo.txt a.out

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

ls command simulation program executed successfully.

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

**AIM:**

To write a C program that simulates the grep UNIX command by searching for a word in a text file and printing matching lines

**ALGORITHM:**

1. Get filename and search word
2. Open file using fopen()
3. For each line:

 a. Check if it contains the word

 b. If yes, print it

1. Close file

**CODE:**

#include <stdio.h>

#include <string.h>

int main() {

int n;

char lines[10][100], word[50];

printf("Enter number of lines: ");

scanf("%d", &n);

getchar(); // To consume newline

printf("Enter %d lines:\n", n);

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

fgets(lines[i], sizeof(lines[i]), stdin);

printf("Enter word to search: ");

scanf("%s", word);

printf("\nMatching lines:\n");

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

if (strstr(lines[i], word))

printf("%s", lines[i]);

return 0;

}

**SAMPLE OUTPUT:**

Enter number of lines: 2

Hello world

This is a test file

Enter word to search: test file

**A screenshot of a computer

AI-generated content may be incorrect.**

**RESULT:**

grep command simulation program executed successfully.

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

**AIM:**

To simulate a classical synchronization problem (Producer-Consumer) in C using semaphores and mutual exclusion.

**ALGORITHM:**

1. Initialize mutex, full, empty
2. In producer():

* Wait on mutex and empty
* Produce item
* Signal full and mutex

1. In consumer():

* Wait on mutex and full
* Consume item
* Signal empty and mutex

**CODE:**

#include <stdio.h>

int full = 0, empty = 3, item = 0;

int main() {

int ch;

while (1) {

printf("\n1.Produce 2.Consume 3.Exit: ");

scanf("%d", &ch);

if (ch == 1) {

if (empty > 0) {

item++; full++; empty--;

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

} else printf("Buffer Full\n");

}

else if (ch == 2) {

if (full > 0) {

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

item--; full--; empty++;

} else printf("Buffer Empty\n");

}

else break;

}

return 0;

}

**SAMPLE INPUT:**

1

1

2

2

3

**SAMPLE OUTPUT:**

Produced 1

1.Produce 2.Consume 3.Exit: Produced 2

1.Produce 2.Consume 3.Exit: Consumed 2

1.Produce 2.Consume 3.Exit: Consumed 1

**A screenshot of a computer

AI-generated content may be incorrect.**

**RESULT:**

Classical process synchronization program executed successfully.

30. Write C programs to demonstrate the following thread related concepts. (i)create (ii) join (iii) equal (iv) exit

**AIM:**

To demonstrate creation, joining, equality checking, and exiting of threads using POSIX threads

**ALGORITHM:**

1. Create a thread using pthread\_create()
2. Inside thread, compare thread IDs using pthread\_equal()
3. Exit thread using pthread\_exit()
4. In main, wait for thread using pthread\_join()

**CODE:**

#include <stdio.h>

#include <pthread.h>

void\* threadFunc(void\* arg) {

printf("Thread running with ID: %lu\n", pthread\_self());

// Check equality with itself (always true)

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

printf("Thread ID matches itself (equal)\n");

pthread\_exit("Thread exited"); // Exit with message

}

int main() {

pthread\_t t1;

void\* status;

// Create

pthread\_create(&t1, NULL, threadFunc, NULL);

printf("Main: Created thread %lu\n", t1);

// Join

pthread\_join(t1, &status);

printf("Main: Joined thread, exit status: %s\n", (char\*)status);

return 0;

}

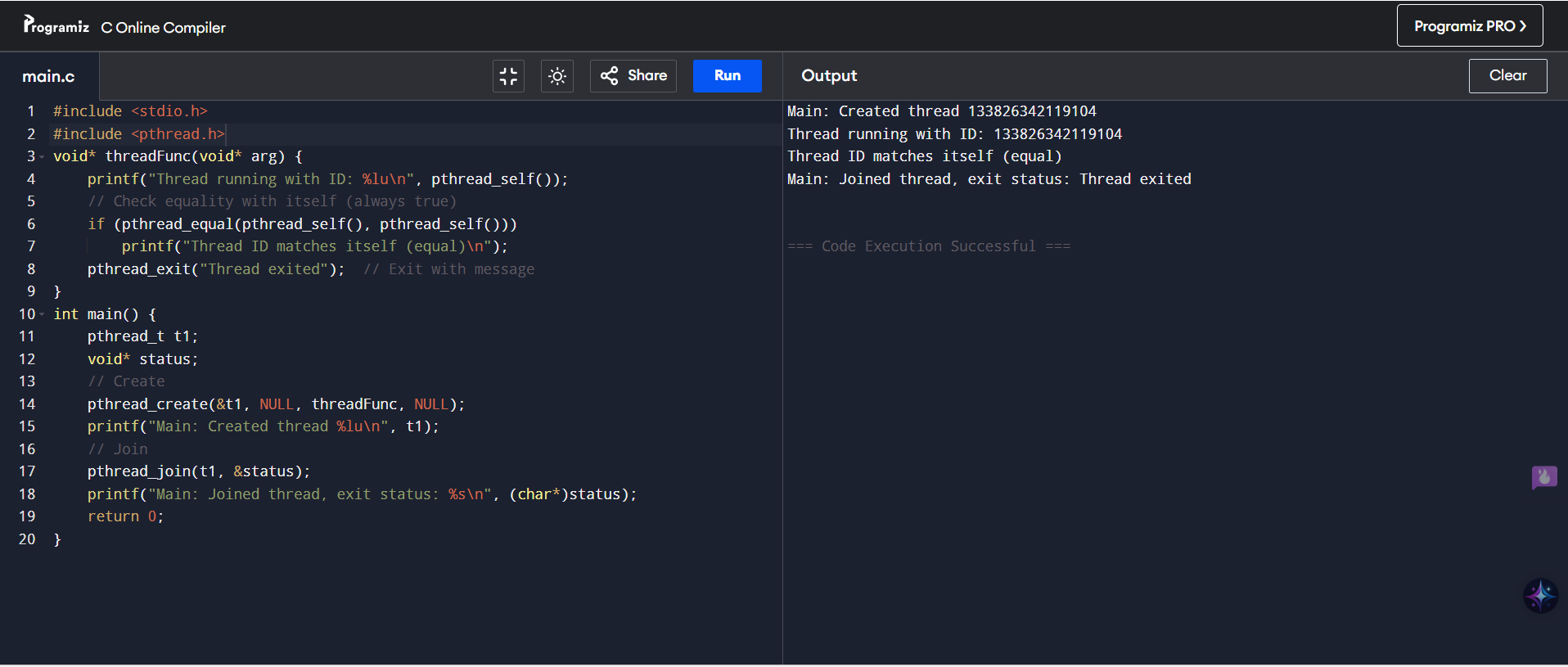
**SAMPLE OUTPUT:**

Main: Created thread 139797085206272

Thread running with ID: 139797085206272

Thread ID matches itself (equal)

Main: Joined thread, exit status: Thread exited

****

**RESULT:**

Thread handling operations (create, join, equal, exit) executed successfully.