**LAB Task 1:**

Create a directory named project  
Navigate into the project directory  
Create two subdirectories, docs, and scripts  
Navigate to the docs directory and create a text file named readme.txt  
Write some content into the readme.txt file  
Display the content of readme.txt to verify the text was written  
Move the readme.txt file from docs to scripts  
Navigate to the scripts directory and check if readme.txt was successfully moved

**Text:**

root@ubuntu:~# cd ..

root@ubuntu:/# ls

bin dev initrd.img media opt run sys var

boot etc lib MInt proc sbin tmp vmlinuz

cdrom home lost+found mnt root srv usr

root@ubuntu:/# cd

root@ubuntu:~# cd Desktop

root@ubuntu:~/Desktop# mkdir project

root@ubuntu:~/Desktop# cd project

root@ubuntu:~/Desktop/project# mkdir docs

root@ubuntu:~/Desktop/project# mkdir script

root@ubuntu:~/Desktop/project# cd docs

root@ubuntu:~/Desktop/project/docs# touch readme.txt

root@ubuntu:~/Desktop/project/docs# echo "This is my readme file">readme.txt

root@ubuntu:~/Desktop/project/docs# cat readme.txt

This is my readme file

root@ubuntu:~/Desktop/project/docs# mv readme.txt ../script/

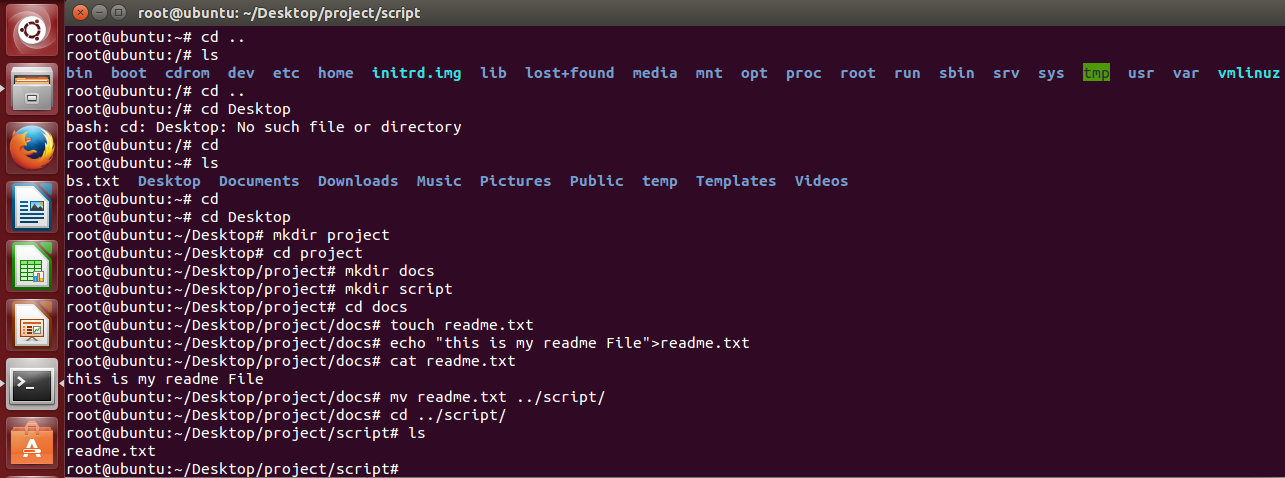
root@ubuntu:~/Desktop/project/docs# cd ../script/

root@ubuntu:~/Desktop/project/script# ls

readme.txt

root@ubuntu:~/Desktop/project/script#

**Screenshot:**

****

**LAB TASK 2:**

**Simple Calculator in C language Using Linux Operating System**

**LINUX Commands:**

root@ubuntu:~/Desktop# gedit Calculator.c

root@ubuntu:~/Desktop# gcc -o Calculator Calculator.c

root@ubuntu:~/Desktop# ./Calculator

Enter two integers: 4

2

Enter choice: 1. Addition 2. Subtraction 3. Multiplication 4. Division

1

Result: 6

Do you want to continue? (Y/N): y

Enter two integers: 8

2

Enter choice: 1. Addition 2. Subtraction 3. Multiplication 4. Division

3

Result: 16

Do you want to continue? (Y/N):

**C Language Code:**

#include<stdio.h>

int main() {

int a, b, choice;

char exit;

do {

printf("Enter two integers: ");

scanf("%d %d", &a, &b);

printf("Enter choice: 1. Addition 2. Subtraction 3. Multiplication 4. Division\n");

scanf("%d", &choice);

// Perform operation based on choice

switch(choice) {

case 1:

printf("Result: %d\n", a + b);

break;

case 2:

printf("Result: %d\n", a - b);

break;

case 3:

printf("Result: %d\n", a \* b);

break;

case 4:

if(b != 0)

printf("Result: %.2f\n", (float)a / b);

else

printf("Error! Division by zero not possible.\n");

break;

default:

printf("Error! Invalid choice.\n");

break;

}

printf("Do you want to continue? (Y/N): ");

getchar();

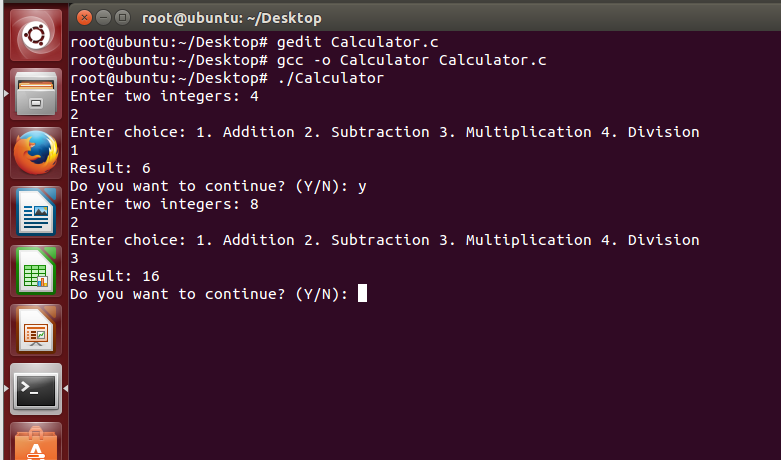
scanf("%c", &exit);

} while(exit == 'Y' || exit == 'y');

return 0;

}

**OUTPUT Screenshot:**

****

**LAB TASK 3:**

**Fork Function in C language Using Linux Operating System**

**LINUX Commands:**

root@ubuntu:~/Desktop# gedit Fork.c

root@ubuntu:~/Desktop# gcc -o Fork Fork.c

root@ubuntu:~/Desktop# ./Fork

Hello from Parent

root@ubuntu:~/Desktop# Hello from Child

Hello from Parent

Hello from Child

**C Language Code:**

#include<stdio.h>

#include<unistd.h>

main()

{

fork();

if(fork()==0)

{

printf("Hello from Child \n");

}

else{

printf("Hello from Parent \n");

}

}

**OUTPUT Screenshot:**

****

**Fork Function and Pid in C language Using Linux Operating System**

**LINUX Commands:**

root@ubuntu:~/Desktop# gedit ForkPid.c

root@ubuntu:~/Desktop# gcc -o ForkPid ForkPid.c

root@ubuntu:~/Desktop# ./

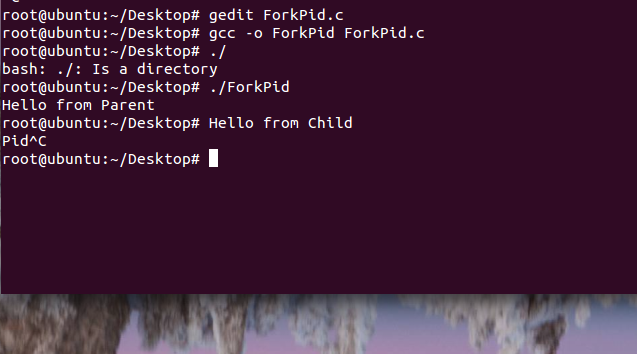
bash: ./: Is a directory

root@ubuntu:~/Desktop# ./ForkPid

Hello from Parent

root@ubuntu:~/Desktop# Hello from Child

Pid

**Screenshot:**

**Fork Function and Pid Second Child Process in C language Using Linux Operating System**

**LINUX Commands:**

root@ubuntu:~/Desktop# gedit ForkSecondChild.c

root@ubuntu:~/Desktop# gcc -o ForkSecondChild ForkSecondChild.c

root@ubuntu:~/Desktop# ./

bash: ./: Is a directory

root@ubuntu:~/Desktop# ./ForkSecondChild

Hello from Parent

root@ubuntu:~/Desktop# Hello from Second Child

Hello from First Child

**C Language Code:**

#include <stdio.h>

#include <unistd.h>

int main() {

pid\_t Pid1 = fork();

if (Pid1 == 0) {

printf("Hello from First Child\n");

} else {

pid\_t Pid2 = fork();

if (Pid2 == 0) {

printf("Hello from Second Child\n");

} else {

printf("Hello from Parent\n");

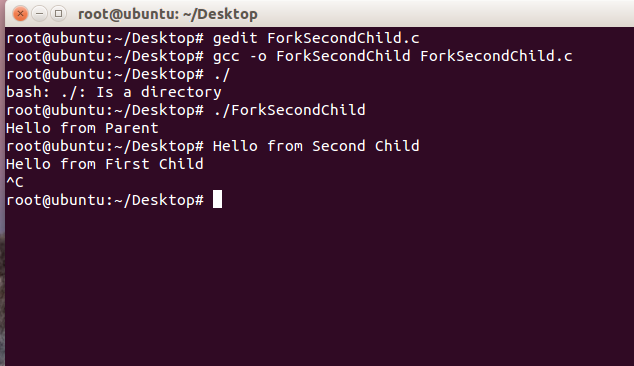
}

}

return 0;

}

**OUTPUT Screenshot:**



**LAB TASK 4:**

**Write a program for matrix addition, subtraction and multiplication using multithreading.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define SIZE 2 // 2x2 matrix

// Declare matrices globally

int A[SIZE][SIZE], B[SIZE][SIZE], result\_add[SIZE][SIZE];

int result\_sub[SIZE][SIZE], result\_mul[SIZE][SIZE];

// Structure to pass arguments to threads

typedef struct {

int row;

int col;

} ThreadArgs;

// Matrix addition function

void \*add\_matrices(void \*args) {

ThreadArgs \*t\_args = (ThreadArgs \*)args;

int i = t\_args->row;

int j = t\_args->col;

result\_add[i][j] = A[i][j] + B[i][j];

pthread\_exit(NULL);

}

// Matrix subtraction function

void \*subtract\_matrices(void \*args) {

ThreadArgs \*t\_args = (ThreadArgs \*)args;

int i = t\_args->row;

int j = t\_args->col;

result\_sub[i][j] = A[i][j] - B[i][j];

pthread\_exit(NULL);

}

// Matrix multiplication function

void \*multiply\_matrices(void \*args) {

ThreadArgs \*t\_args = (ThreadArgs \*)args;

int i = t\_args->row;

int j = t\_args->col;

int k;

result\_mul[i][j] = 0;

for (k = 0; k < SIZE; k++) {

result\_mul[i][j] += A[i][k] \* B[k][j];

}

pthread\_exit(NULL);

}

// Function to print matrices

void print\_matrix(int mat[SIZE][SIZE]) {

int i, j;

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

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

printf("%d ", mat[i][j]);

}

printf("\n");

}

}

int main() {

pthread\_t add\_threads[SIZE][SIZE], subtract\_threads[SIZE][SIZE], multiply\_threads[SIZE][SIZE];

ThreadArgs t\_args[SIZE][SIZE];

int i, j;

// Input matrix A

printf("Enter elements for matrix A (2x2):\n");

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

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

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

}

}

// Input matrix B

printf("Enter elements for matrix B (2x2):\n");

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

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

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

}

}

// Matrix addition (create threads for each element)

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

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

t\_args[i][j].row = i;

t\_args[i][j].col = j;

pthread\_create(&add\_threads[i][j], NULL, add\_matrices, (void \*)&t\_args[i][j]);

}

}

// Wait for all addition threads to finish

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

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

pthread\_join(add\_threads[i][j], NULL);

}

}

printf("Result of Matrix Addition:\n");

print\_matrix(result\_add);

// Matrix subtraction (create threads for each element)

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

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

t\_args[i][j].row = i;

t\_args[i][j].col = j;

pthread\_create(&subtract\_threads[i][j], NULL, subtract\_matrices, (void \*)&t\_args[i][j]);

}

}

// Wait for all subtraction threads to finish

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

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

pthread\_join(subtract\_threads[i][j], NULL);

}

}

printf("Result of Matrix Subtraction:\n");

print\_matrix(result\_sub);

// Matrix multiplication (create threads for each element)

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

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

t\_args[i][j].row = i;

t\_args[i][j].col = j;

pthread\_create(&multiply\_threads[i][j], NULL, multiply\_matrices, (void \*)&t\_args[i][j]);

}

}

// Wait for all multiplication threads to finish

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

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

pthread\_join(multiply\_threads[i][j], NULL);

}

}

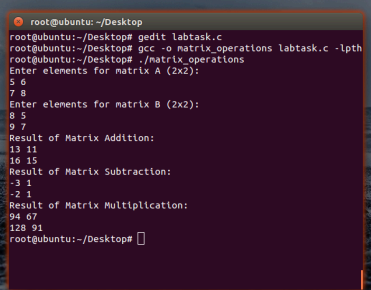
printf("Result of Matrix Multiplication:\n");

print\_matrix(result\_mul);

return 0;

}

**Output Screenshot:**



**MIDTERM TASKS:**

**Q1: Write a program that asks the user to input a random positive integer. The program should create a child process that checks the number and behaves according to these conditions:  
  
•    If the number is divisible by 5, the child process should display "Number is divisible by 5" and then exit.  
•    If the number is not divisible by 5, the parent process should display "Try again with a different number" and exit.** **#include <stdio.h>**

**The program should use the wait() system call to synchronize the parent and child processes.**

Question1:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

int number;

printf("Enter a positive integer: ");

scanf("%d", &number);

pid\_t pid = fork();

if (pid == 0) { // Child process

if (number % 5 == 0) {

printf("Number is divisible by 5\n");

exit(0);

}

exit(1);

} else if (pid > 0) { // Parent process

int status;

wait(&status); // Wait for child process

if (WEXITSTATUS(status) != 0) {

printf("Try again with a different number\n");

}

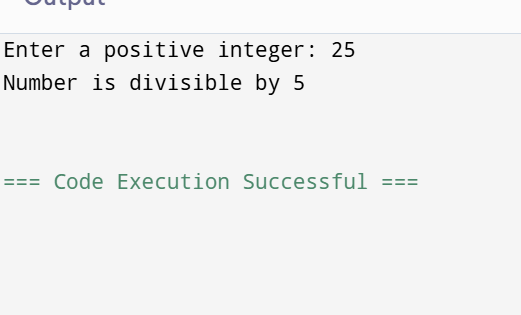
} else {

perror("fork failed");

}

return 0;

}



**Q2: The perimeter of a triangle is calculated based on all the sides provided in a triangle. As a programmer calculates the area and perimeter of a triangle with the help of the formulas given below, also know this Program has two processes: The parent process (Program itself) and the Child process fork by the parent.  
●    \*Area of triangle = 1/2 \*base \*vertical height  
●    Perimeter of a triangle = Side1 + Side2 + Side3  
●    For both equations input is taken by the parent**

Question2

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

double base, height, side1, side2, side3;

printf("Enter base and height of the triangle: ");

scanf("%lf %lf", &base, &height);

printf("Enter three sides of the triangle: ");

scanf("%lf %lf %lf", &side1, &side2, &side3);

pid\_t pid = fork();

if (pid == 0) { // Child process

double area = 0.5 \* base \* height;

printf("Area of the triangle: %.2lf\n", area);

} else if (pid > 0) { // Parent process

wait(NULL); // Wait for child to finish

double perimeter = side1 + side2 + side3;

printf("Perimeter of the triangle: %.2lf\n", perimeter);

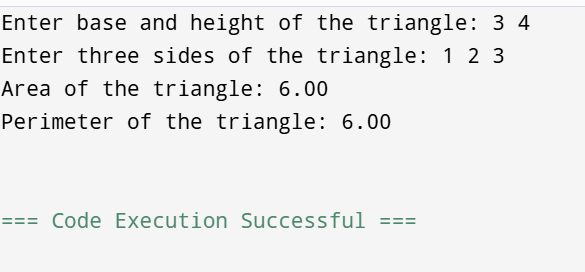
} else {

perror("fork failed");

}

return 0;

}



**Q3:  In a private telecom networking company, one customer creates a signal for call making to main branch. IT assistant dispatcher in call center is asked to create two process or threads call from that signal to display process ID of customer to find ID same or not so user can lodge a complaint in complaint cell. about the internet issue. Write a c language code for this hypothetical scenario.**

**Note: gcc -pthread program.c -o program**

Question#3  
#include <stdio.h>

#include <stdlib.h> // For exit()

#include <unistd.h> // For fork() and getpid()

#include <sys/wait.h> // For wait()

int main() {

pid\_t pid1, pid2;

pid1 = fork(); // Create the first child process

if (pid1 == 0) {

// In the first child process

printf("Process ID of Customer: %d\n", getpid());

exit(0); // Terminate the child process

} else {

// In the parent process

wait(NULL); // Wait for the first child to finish

pid2 = fork(); // Create the second child process

if (pid2 == 0) {

// In the second child process

printf("Process ID of Dispatcher: %d\n", getpid());

exit(0); // Terminate the child process

}

wait(NULL); // Wait for the second child to finish

// Compare the process IDs in the parent process

if (pid1 == pid2) {

printf("Process IDs are the same.\n");

} else {

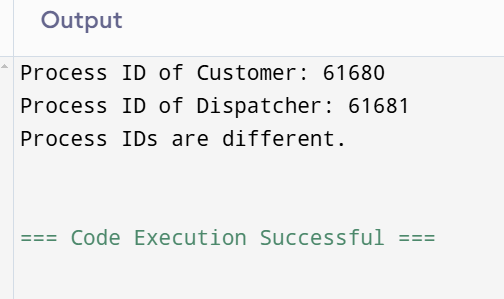
printf("Process IDs are different.\n");

}

}

return 0;

}



**Q4: Write a Multithreaded program which performs the following**

**Steps:**

**1.    Input a message from user.  
2.    Create three independent threads each of which will execute different functions (Add, Sub, Multiply).  
3.    Main function takes two numbers as input and pass to threads.  
4.    Each thread applies separate operation.  
5.    All threads send the result of operation to the main function.  
6.    Main function displays the output.**

Question #4

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

// Structure to pass arguments to threads

typedef struct {

int num1;

int num2;

int result;

} ThreadData;

// Function to perform addition

void \*add(void \*args) {

ThreadData \*data = (ThreadData \*)args;

data->result = data->num1 + data->num2;

return NULL;

}

// Function to perform subtraction

void \*subtract(void \*args) {

ThreadData \*data = (ThreadData \*)args;

data->result = data->num1 - data->num2;

return NULL;

}

// Function to perform multiplication

void \*multiply(void \*args) {

ThreadData \*data = (ThreadData \*)args;

data->result = data->num1 \* data->num2;

return NULL;

}

int main() {

char message[100];

int num1, num2;

// Input a message from user

printf("Enter a message: ");

fgets(message, sizeof(message), stdin);

// Input two numbers

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

// Create thread data

ThreadData addData = {num1, num2, 0};

ThreadData subData = {num1, num2, 0};

ThreadData mulData = {num1, num2, 0};

// Create threads

pthread\_t thread1, thread2, thread3;

pthread\_create(&thread1, NULL, add, &addData);

pthread\_create(&thread2, NULL, subtract, &subData);

pthread\_create(&thread3, NULL, multiply, &mulData);

// Wait for threads to finish

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

pthread\_join(thread3, NULL);

// Display the results

printf("\nMessage: %s", message);

printf("Addition result: %d\n", addData.result);

printf("Subtraction result: %d\n", subData.result);

printf("Multiplication result: %d\n", mulData.result);

return 0;

}

**Q5: In an online booking system, a user requests a ticket for an event. The support team is asked to create two processes to verify the user's booking request. The program should print the Process IDs of each process and validate whether the booking was successfully processed. The system should also check whether both process IDs match for proper validation. Write a program to simulate this scenario.**

Question#5

#include <stdio.h>

#include <stdlib.h> // Include stdlib.h for exit()

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid1 = fork();

if (pid1 == 0) { // First child process

printf("Process 1 ID: %d\n", getpid());

exit(0);

} else {

wait(NULL); // Wait for the first child process to finish

pid\_t pid2 = fork();

if (pid2 == 0) { // Second child process

printf("Process 2 ID: %d\n", getpid());

exit(0);

}

wait(NULL); // Wait for the second child process to finish

// Parent process compares its own process ID to ensure correct logic

printf("Process 1 ID was: %d\n", pid1); // pid1 contains child's PID

printf("Process 2 ID was: %d\n", pid2); // pid2 contains child's PID

if (pid1 == pid2) {

printf("Booking validation successful.\n");

} else {

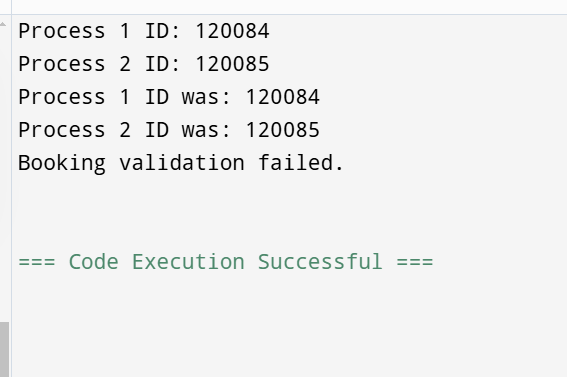
printf("Booking validation failed.\n");

}

}

return 0;

}



**Q6: Write a multithreaded program that performs the following steps:•    Input an integer array from the user.  
•    Create three independent threads, each of which will perform the following tasks:  
•    Find the maximum value in the array.  
•    Find the minimum value in the array.  
•    Calculate the sum of the array elements.  
•    The main function takes the array as input and passes it to the threads.  
•    Each thread performs its operation and sends the result back to the main function.  
•    The main function displays all three results.**

Question#6

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

// Structure to hold array and its size

typedef struct {

int \*array;

int size;

} ArrayData;

// Global variables to store results

int max\_value, min\_value, sum;

// Thread function to find maximum value

void \*find\_max(void \*arg) {

ArrayData \*data = (ArrayData \*)arg;

int i;

max\_value = data->array[0];

for (i = 1; i < data->size; i++) {

if (data->array[i] > max\_value) {

max\_value = data->array[i];

}

}

pthread\_exit(0);

}

// Thread function to find minimum value

void \*find\_min(void \*arg) {

ArrayData \*data = (ArrayData \*)arg;

int i;

min\_value = data->array[0];

for (i = 1; i < data->size; i++) {

if (data->array[i] < min\_value) {

min\_value = data->array[i];

}

}

pthread\_exit(0);

}

// Thread function to calculate the sum of the array

void \*calculate\_sum(void \*arg) {

ArrayData \*data = (ArrayData \*)arg;

int i;

sum = 0;

for (i = 0; i < data->size; i++) {

sum += data->array[i];

}

pthread\_exit(0);

}

int main() {

int n, i;

// Input array size

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

// Allocate memory for the array

int \*array = (int \*)malloc(n \* sizeof(int));

if (array == NULL) {

fprintf(stderr, "Memory allocation failed\n");

return 1;

}

// Input array elements

printf("Enter the elements of the array:\n");

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

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

}

// Prepare data to pass to threads

ArrayData data = {array, n};

// Create thread identifiers

pthread\_t threads[3];

// Create threads

pthread\_create(&threads[0], NULL, find\_max, &data);

pthread\_create(&threads[1], NULL, find\_min, &data);

pthread\_create(&threads[2], NULL, calculate\_sum, &data);

// Wait for threads to complete

for (i = 0; i < 3; i++) {

pthread\_join(threads[i], NULL);

}

// Display results

printf("Maximum value: %d\n", max\_value);

printf("Minimum value: %d\n", min\_value);

printf("Sum of array elements: %d\n", sum);

// Free allocated memory

free(array);

return 0;

}

