Project Report

This project is a revised version of the old one and improves on the parts that the previous one failed to achieve

The first key feature is left intact since it achieves it.

The second feature which is Interprocess Communications Mechanisms, was improved upon. Originally, it had these functions where it would try to demonstrate the shared memory and message passing but was done poorly and did not demonstrate the correct functionalities. So in the revised version, i’ve added additional definitions like so:

* #define MAX\_SEGMENTS 10
* #define MAX\_THREADS 10
* #define MAX\_PROCESSES 10
* #define SHARED\_MEMORY\_KEY 1234
* #define MESSAGE\_SIZE 256
* #define SEGMENT\_SIZE 1000

And created structure type

typedef struct {

pthread\_mutex\_t mutex;

char message[MESSAGE\_SIZE];

int flag;

int terminate;

} SharedData;

I got rid of cpu usage and memory usage functions in the original because it was unnecessary.

Added this:

//Interprocess communication mechanisms

// Shared memory and message passing for processes

void\* process\_ipc(void\* arg) {

SharedData\* shared\_data = (SharedData\*)arg;

while (!shared\_data->terminate) {

pthread\_mutex\_lock(&shared\_data->mutex);

if (shared\_data->flag) {

printf("Process received message: %s\n", shared\_data->message);

shared\_data->flag = 0;

}

pthread\_mutex\_unlock(&shared\_data->mutex);

usleep(100000); // Simulating processing time (100ms)

}

return NULL;

}

// Shared memory and message passing for threads

void\* thread\_ipc(void\* arg) {

SharedData\* shared\_data = (SharedData\*)arg;

while (!shared\_data->terminate) {

pthread\_mutex\_lock(&shared\_data->mutex);

if (shared\_data->flag) {

printf("Thread received message: ");

if(strlen(shared\_data->message) > 50) {

printf("%.50s...\n", shared\_data->message); // Display only first 50 characters

} else {

printf("%s\n", shared\_data->message);

}

shared\_data->flag = 0; // Reset flag after processing the message

}

pthread\_mutex\_unlock(&shared\_data->mutex);

usleep(100000); // Simulating processing time (100ms)

}

return NULL;

}

// Function to handle sending messages between processes or threads

void send\_message\_ipc(SharedData\* shared\_data) {

printf("Enter message to send: ");

char message[MESSAGE\_SIZE];

scanf("%s", message);

pthread\_mutex\_lock(&shared\_data->mutex);

strcpy(shared\_data->message, message);

shared\_data->flag = 1;

pthread\_mutex\_unlock(&shared\_data->mutex);

printf("Message sent successfully.\n");

}

// Evaluate and compare performance of IPC mechanisms

void evaluate\_ipc\_performance(SharedData\* shared\_data) {

clock\_t start, end;

double cpu\_time\_used;

// Sending short messages

start = clock();

strcpy(shared\_data->message, "Short message");

shared\_data->flag = 1;

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken to send a short message: %f seconds\n", cpu\_time\_used);

// Sending long messages

start = clock();

char long\_message[MESSAGE\_SIZE];

memset(long\_message, 'A', MESSAGE\_SIZE - 1);

long\_message[MESSAGE\_SIZE - 1] = '\0'; // Null-terminate the string

strcpy(shared\_data->message, long\_message);

shared\_data->flag = 1;

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken to send a long message: %f seconds\n", cpu\_time\_used);

}

SharedData shared\_data;

pthread\_mutex\_init(&shared\_data.mutex, NULL);

shared\_data.flag = 0;

// create threads for IPC

pthread\_create(&process\_ipc\_thread, NULL, process\_ipc, (void\*)&shared\_data);

pthread\_create(&thread\_ipc\_thread, NULL, thread\_ipc, (void\*)&shared\_data);

// Wait for IPC threads to terminate

pthread\_join(process\_ipc\_thread, NULL);

pthread\_join(thread\_ipc\_thread, NULL);

For the third feature which is parallel text processing:

It was unclear in the original how it simulated the feature. In the revised version I’ve added:

void load\_file(TextProcessor\* tp) {

FILE\* file = fopen("something.txt", "r");

if (file == NULL) {

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

exit(1);

}

char buffer[SEGMENT\_SIZE];

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

strcpy(tp->segments[tp->segment\_count].segment, buffer);

tp->segment\_count++;

}

fclose(file);

}

void\* process\_segment(void\* arg) {

Segment\* segment = (Segment\*)arg;

for (int i = 0; i < strlen(segment->segment); i++) {

if (isalpha(segment->segment[i])) {

segment->segment[i] = toupper(segment->segment[i]);

}

}

return NULL;

}

void\* count\_characters(void\* arg) {

TextProcessor\* tp = (TextProcessor\*)arg;

for (int i = 0; i < tp->segment\_count; i++) {

for (int j = 0; j < strlen(tp->segments[i].segment); j++) {

if (isalpha(tp->segments[i].segment[j])) {

tp->counts[(int)tp->segments[i].segment[j]].character = tp->segments[i].segment[j];

tp->counts[(int)tp->segments[i].segment[j]].count++;

}

}

}

return NULL;

}

void parallel\_text\_processing() {

TextProcessor tp;

tp.segment\_count = 0;

// Load file into segments

load\_file(&tp);

// Create threads for processing segments

pthread\_t process\_threads[MAX\_THREADS];

for (int i = 0; i < tp.segment\_count; i++) {

pthread\_create(&process\_threads[i], NULL, process\_segment, (void\*)&tp.segments[i]);

}

// Wait for processing threads to complete

for (int i = 0; i < tp.segment\_count; i++) {

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

}

// Create thread for counting characters

pthread\_t count\_thread;

pthread\_create(&count\_thread, NULL, count\_characters, (void\*)&tp);

pthread\_join(count\_thread, NULL);

// Display character counts

printf("Character Counts:\n");

for (int i = 0; i < 256; i++) {

if (isalpha(tp.counts[i].character)) {

printf("%c: %d\n", tp.counts[i].character, tp.counts[i].count);

}

}

}

pthread\_mutex\_destroy(&shared\_data.mutex);

And in the 4th feature, the simple user interface is a command line interface where users are prompted to pick an option among the available ones.