CSE 344
HW5
REPORT

Ömer Faruk SAYAR
171044038

Global Variables:

```c
typedef struct {
    int src_fd;
    int dest_fd;
    char src_path[MAX_PATH_LENGTH];
    char dest_path[MAX_PATH_LENGTH];
} Copy;

pthread_mutex_t buffer_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t output_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t total_bytes_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t total_files_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t buffer_not_full_cond = PTHREAD_COND_INITIALIZER;
pthread_cond_t buffer_not_empty_cond = PTHREAD_COND_INITIALIZER;

Copy* buffer;
int buffer_size = 0;
int buffer_capacity = 0;
int buffer_front = 0;
int buffer_rear = 0;
int num_consumers = 0;
long long total_file = 0;
long long total_byte = 0;
long long total_directory = 0;
__sig_atomic_t done = 0;
__sig_atomic_t total_fifo = 0;

pthread_t producer_thread;
pthread_t* consumer_threads;
```

To provide synchronization between producer thread and consumer threads I used mutex and condition variables.

Since our buffer is the critical section, producer locks the buffer and wait until buffer_not_full signal comes using pthread_cond_wait before enqueue the Copy struct. After the enqueue operation it signals to buffer_not_emtpy condition variable.

Consumers also lock the buffer using mutex and wait until buffer_not_empty signal comes using pthread_cond_wait. After the dequeue operation consumer send signal to buffer_not_full variable.

Producer:

```c
Copy request;
request.src_fd = src_fd;
request.dest_fd = dest_fd;
strncpy(request.src_path, src_file, sizeof(request.src_path));
strncpy(request.dest_path, dest_file, sizeof(request.dest_path));

pthread_mutex_lock(&buffer_mutex);

while (is_buffer_full()) {
    pthread_cond_wait(&buffer_not_full_cond, &buffer_mutex);
}

enqueue(request);

pthread_cond_signal(&buffer_not_empty_cond);
pthread_mutex_unlock(&buffer_mutex);
```

Consumer:

```c
void* consumer(void* arg) {
    while (1) {

        pthread_mutex_lock(&buffer_mutex);

        while (is_buffer_empty() && !done) {
            pthread_cond_wait(&buffer_not_empty_cond, &buffer_mutex);
        }

        if (is_buffer_empty() && done) {
            pthread_mutex_unlock(&buffer_mutex);
            break;
        }

        Copy request = dequeue();
        pthread_cond_signal(&buffer_not_full_cond);
        pthread_mutex_unlock(&buffer_mutex);

        copy_file(request);
    }
    pthread_exit(NULL);
}
```

I also used a mutex for writing to stdout and another mutex to sum total byte copied and total regular file number copied. I used sig_atomic_t type for counting number of FIFO files copied.

```c
void signal_handler(int signum){

    pthread_mutex_lock(&buffer_mutex);
    done = 1;
    pthread_cond_broadcast(&buffer_not_empty_cond);
    pthread_mutex_unlock(&buffer_mutex);

    pthread_cancel(producer_thread);
    for (int i = 0; i < num_consumers; i++) {
        pthread_cancel(consumer_threads[i]);
    }

    // Wait for producer and consumer threads to terminate
    pthread_join(producer_thread, NULL);
    for (int i = 0; i < num_consumers; i++) {
        pthread_join(consumer_threads[i], NULL);
    }

    free(buffer);
    free(consumer_threads);

    printf("Exit signal received! \n");
    printf("Total number of regular files: %lld\n", total_file);
    printf("Total number of directories: %lld\n", total_directory);
    printf("Total number of bytes: %lld\n", total_byte);
    printf("Total number of FIFOs: %d\n", total_fifo);

    exit(signum);
}
```

There is a signal handler for SIGINT, SIGTERM and SIGQUIT signals. This handler makes done flag 1 and broadcasts buffer_not_empty signal then cancels producer thread and consumer threads. After wait until all threads terminates. Then frees all resources that allocated and prints the results.

I implemented buffer as a circular array queue with fixed size for better performance.

```c
void init_buffer(int size) {
    buffer = (Copy*)malloc(sizeof(Copy) * size);
    buffer_size = 0;
    buffer_capacity = size;
    buffer_front = 0;
    buffer_rear = 0;
}

int is_buffer_empty() {
    return buffer_size == 0;
}

int is_buffer_full() {
    return buffer_size == buffer_capacity;
}

void enqueue(Copy request) {
    buffer[buffer_rear] = request;
    buffer_rear = (buffer_rear + 1) % buffer_capacity;
    buffer_size++;
}

Copy dequeue() {
    Copy request = buffer[buffer_front];
    buffer_front = (buffer_front + 1) % buffer_capacity;
    buffer_size--;
    return request;
}
```

TEST RESULTS

Buffer Size vs. Performance:

Buffer size: 10
Consumer: 2

```
Total time taken: 0.67 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Buffer size: 50
Consumer: 2

```
Total time taken: 0.66 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Buffer size: 100
Consumer: 2

```
Total time taken: 0.63 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Buffer size: 500
Consumer: 2

```
Total time taken: 0.62 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Observations: From the results, increasing the buffer size from 10 to 500 generally leads to a decrease in the total time taken, indicating improved performance. However, after a certain threshold (around 500), the performance gain becomes marginal. Therefore, a buffer size of around 100-500 seems to provide a good balance between performance and memory usage.

Number of Consumer Threads vs. Performance:

Buffer Size: 500
Number of Consumers: 2

```
Total time taken: 0.64 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Buffer Size: 500
Number of Consumers: 4

```
Total time taken: 0.56 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Buffer Size: 500
Number of Consumers: 8

```
Total time taken: 0.55 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Buffer Size: 500
Number of Consumers: 16

```
Total time taken: 0.58 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Buffer Size: 500
Number of Consumers :32

```
Total time taken: 0.57 seconds
Total file copied: 7108
Total directory copied: 1581
Total FIFO copied: 1
Total byte copied: 761571694
```

Observations: Increasing the number of consumer threads from 2 to 4 leads to a significant improvement in performance, reducing the total time taken. However, further increasing the number of consumer threads beyond 4 does not result in a significant improvement. In fact, in some cases, it may slightly degrade performance due to increased thread coordination overhead. Therefore, a number of consumer threads between 4 and 8 seems to provide the best performance.

Conclusion: Based on the conducted experiments, the following combinations of buffer size and number of consumer threads produce the best results for my code:

- Buffer Size: 500
- Number of Consumer Threads: 8