CAB403 - Process Management and Distributed Computing Assignment

Timothy Mollenhauer n9405194
Luke Kane n7004249

Each group member has contributed equally to the assignment.

Statement of Completeness

Task 1

We have completed the majority of Task 1. We were unable to get the server to exit gracefully on CTRL-C. There are some bugs that we have been unable to fix. On some occasions numerous words get send to a single client at the same time. This only occurs sometimes and only on Ubuntu.

On occasion the client will have an issue with one of our variables using malloc and will core dump.

Task 2

The multithreaded implementation is partially complete. The server concurrently runs on multiple threads, however the leaderboard does not share memory and therefore can only display the current user's score.

Task 3

The thread pool has been implemented. We are able to create and use thread pools up to 10 clients concurrently. Thread cleaning has been an issue and remains partially complete. If 10 users connect concurrently, it will not allow any additional users to connect as expected, however if a user disconnects they are unable to reconnect.

Compilation Instructions

We did not use a make file. The instructions for compilation are:
gcc -o Server Server.c -pthread
gcc -o Client Client.c

Section 1 Leaderboard Structure

We did not find the time to implement the critical section of the leaderboard so the leaderboard data structure is quite simple.

```
int games_won;
int games_played;
```

The Leader board uses the games_won and games_played variables to record the number of games won and games played. Since we do not have to track multiple users as our leader board is only built show a single user there wins and losses there is no need to track the username of the user on the server. This is handled on the client.

Section 3 Thread Pools

```
for (int i = 0; i < MAX_USERS; i++) {
    if (pthread_create(&threads[i], NULL, client_handler, (void *) &clients_infos[i]) != 0) {
        exit(EXIT_FAILURE);
    }
}
```

The thread pool is created in the main function it is designed to create a number of threads up to the max users allowed by the server. It will then call the client_handler function to manage threads.

```
void* client_handler(void *client_Info)
{
    int client_fd = 0;

    while (server_running) {
        sem_wait(&sclient);

        if (!server_running) {
            break;
        }

        client_fd = get_client_from_queue();
        run_game(client_fd);
    }
        close(client_fd);
        sem_post(&clienthandler);
        return NULL;
}
```

The client_handler function is used to manage threads. Up to 10 instances of this function are created by the thread creator.
The function waits for a client to connect then it will run through the game until the client disconnects it close the thread. Currently it does not dispose of threads correctly and has trouble if the limit of concurrent users is reached they will be able to disconnect but not reconnect.