LIS2062-1 Operating Systems

“Final Project - Battleship”

Problem Statement.

The main objective of this project is to implement the classic Battleship game in the C language, adapting its rules for two players, turn-based play, and guessing. In addition, we will make use of the knowledge acquired about operating systems. In this game, we will be able to apply concepts such as processes, child processes, threads, signals, alarms, synchronization, race condition, critical region, mutual exclusion, semaphores, mutex, or condition variables.

By implementing this game, our objective is to reinforce our understanding of operating system concepts and their practical applications. Furthermore, we will gain experience in applying the C language to solve complex problems and implementing an efficient and effective algorithm to simulate the game's mechanics.

General and individual objectives.

In this work we will seek to implement what we have learned in the course in the battleship game.This means that as a personal goal we will try to include all possible themes for the game which can include alarms, threads, processes, etc. However, the tools that are useful and functional for the game were placed.

All this subjected to certain guidelines that we must comply with, such as:

- It's a game for two players and each player has a board, at the beginning of the game they must establish where they will position their ships.
- Each user must guess (by turns) where the opponent's hidden ships are. When a position is indicated, it is discovered and allows to see if there was a hit or a miss.
- Each turn, it should print to the users how many hits the opponent has.
- The map measures 10x10.
- The user will enter the position they want to attack.
- Your board should show column and row numbers to make the map easier to read.
- If the user destroys all ships, a congratulatory message should be displayed.
- Cell types (suggested nomenclature, you can modify it).
    - . unexplored cell.
    - x cell where a ship has been hit.
    - '-' cell where an attempt to hit a ship was made but failed.

Design of the solution proposal:

- **Operation diagrams.**

  The program starts at main(), which calls the initialize_boards() function to set up the boards for both players and the print_board() function to print the boards. Then main() creates two threads, one for each player, which calls the player_thread() function. Each thread continues to call the attack() function until one of the players loses all their ships. The attack() function takes the row and column coordinates of the attack and checks if the attack is a hit or a miss. If the attack is a hit, it is marked on the board of the attacked player. After each attack, the print_board() function is called to display the new state of the board. The gameOver() function checks if all of a player's ships have been sunk and returns true if that is the case.

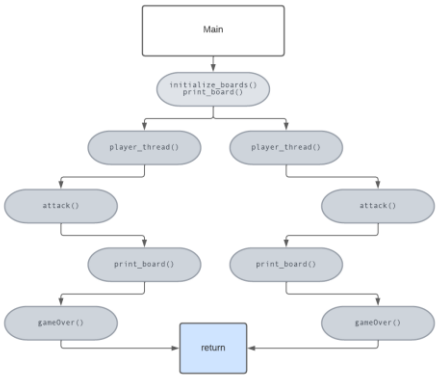


Fig 1. Diagram

- **Screenshots of the code.**

The defined variables are

- Player#_board: A two-dimensional array of characters that represents the game board for player #.
- mutex: a pthread mutex that is used to synchronize access to the game boards.

The defined functions are:

- initialize_boards(): a function that initializes the game boards with a specified number of ships.
- print_board(): a function that prints the game board to the console.
- attack(): a function that performs an attack by one player on the other player's board.
- gameOver(): a function that checks if the game is over by checking if all ships have been sunk.
- player_thread(): a function that is executed by a thread and represents a player's turn in the game.

This function initializes the game boards for both players with the given number of ships. It first sets all the cells of the board to be unexplored, then prompts each player to enter the starting row and column for each ship and places the ships horizontally on the board by marking the cells as SHIP. The function assumes that the ships have a fixed length of 1 cell.

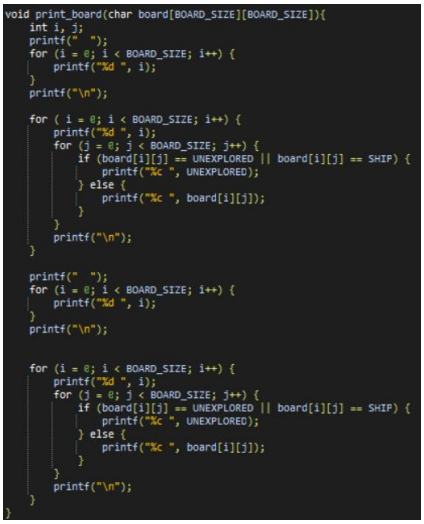
```
void boards(int ship_count){
    int i, j;

    for (i = 0; i < BOARD_SIZE; i++) {
        for (j = 0; j < BOARD_SIZE; j++) {
            player1_board[i][j] = UNEXPLORED;
            player2_board[i][j] = UNEXPLORED;
        }
    }

    for (i = 0; i < ship_count; i++) {
        printf("Player 1 enter the starting row and column for ship %d : ", i + 1);
        int row, col;
        scanf("%d %d", &row, &col);

        int ship_length = 1;
        for (j = 0; j < ship_length; j++) {
            player1_board[row][col + j] = SHIP;
        }
    }

    for (i = 0; i < ship_count; i++) {
        printf("Player 2 enter the starting row and column for ship %d : ", i + 1);
        int row, col;
        scanf("%d %d", &row, &col);

        int ship_length = 2;
        for (j = 0; j < ship_length; j++) {
            player2_board[row][col + j] = SHIP;
        }
    }
}
```

Fig 3. Boards

The `print_board` function prints the game board to the console, including both the player's own board (with ships placed, hits and misses) and the opponent's board (with only hits and misses visible).

It takes a two-dimensional array of characters `board` with size `BOARD_SIZE` x `BOARD_SIZE` as input, which represents the game board. The function first prints the column indices at the top, then prints the rows of the player's own board with UNEXPLORED characters represented by dots, SHIP characters represented by the letter 'w', HIT characters represented by the letter 'x', and MISS characters represented by the letter '-'.

Then it prints the column indices again, followed by the rows of the opponent's board, with UNEXPLORED characters represented by dots, and HIT and MISS characters represented as before, but SHIP characters are hidden.



```
void print_board(char board[BOARD_SIZE][BOARD_SIZE]){
    int i, j;
    printf("  ");
    for (i = 0; i < BOARD_SIZE; i++) {
        printf("%d ", i);
    }
    printf("\n");

    for ( i = 0; i < BOARD_SIZE; i++) {
        printf("%d ", i);
        for (j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == UNEXPLORED || board[i][j] == SHIP) {
                printf("%c ", UNEXPLORED);
            } else {
                printf("%c ", board[i][j]);
            }
        }
        printf("\n");
    }

    printf("  ");
    for (i = 0; i < BOARD_SIZE; i++) {
        printf("%d ", i);
    }
    printf("\n");


    for (i = 0; i < BOARD_SIZE; i++) {
        printf("%d ", i);
        for (j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == UNEXPLORED || board[i][j] == SHIP) {
                printf("%c ", UNEXPLORED);
            } else {
                printf("%c ", board[i][j]);
            }
        }
        printf("\n");
    }
}
```

Fig 4. Print_board

This function allows a player to attack the other player's board by entering the row and column to attack. It takes two arguments: the attacking player's identifier (either '1' or '2') and the defending player's board.

The function first prompts the attacking player to enter the row and column to attack. It then acquires a lock on the mutex to ensure that the print_board function will display an accurate representation of the board at the time of the attack.

If the defending player's board at the row and column specified by the attacking player contains a ship, the function prints a message indicating that the attacking player has hit the ship and updates the board to reflect the hit. If the board at the specified location contains a hit or a miss, the function prints a message indicating that the location has already been attacked. If the board at the specified location is unexplored, the function prints a message indicating that the attacking player has missed and updates the board to reflect the miss. After completing the attack, the function calls the print_board function to display the updated board and releases the lock on the mutex.

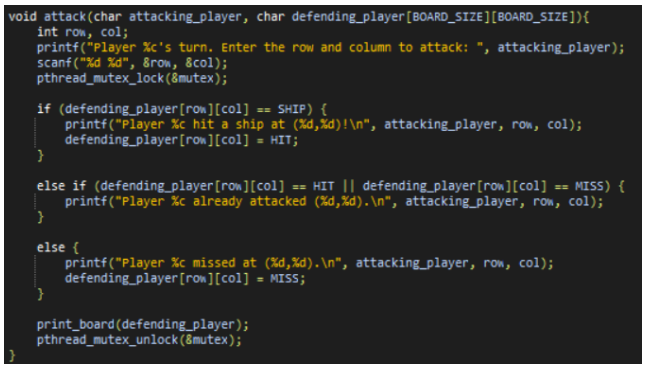


Fig 5. Attack

The `gameOver` function checks if there are any remaining ships on the game board. It iterates over each position on the board and if it finds any position that has a ship on it, it returns 0, indicating that the game is not over yet. If it reaches the end of the board without finding any ships, it returns 1, indicating that the game is over. This function is used to determine when the game should end.

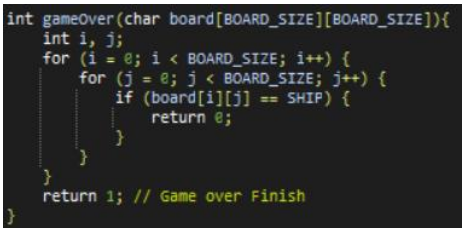


Fig 6. gameOver

This is a thread function that will be called for each player. It takes a pointer to the player's board as a parameter and then loops infinitely until the game is over. In each iteration of the loop, it calls the `attack` function with the player's number and board as arguments. If the game is over after the attack, it prints a message announcing the winner and exits the loop. Finally, it returns `NULL`.

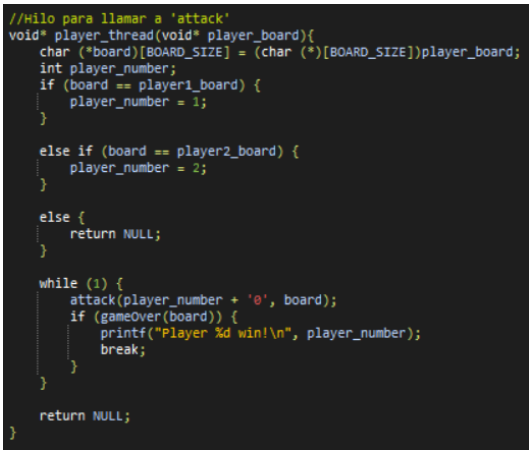


Fig 7. Player_Thread

The main function first prompts the user to enter the number of ships for both players. If the user enters a value less than 2, the program will set the ship count to 2.

Then, the `boards` function is called to initialize the player boards with the given ship count. Next, the program prints out the boards for each player using the `print_board` function. After that, the program initializes a mutex using `pthread_mutex_init`, creates two threads for each player using `pthread_create`, and waits for both threads to finish using `pthread_join`.

Finally, the program destroys the mutex using `pthread_mutex_destroy` and returns 0 to indicate successful completion.

```c
int main(){

    int ship_count;
    printf("Enter the number of ships for both players (minimum 2): ");
    scanf("%d", &ship_count);

    if(ship_count == 1){
        ship_count+=1;
    }

    boards(ship_count);

    printf("Player 1:\n");
    print_board(player1_board);

    printf("\nPlayer 2:\n");
    print_board(player2_board);

    // Initialize mutex
    pthread_mutex_init(&mutex, NULL);
    pthread_t player1_thread, player2_thread;
    pthread_create(&player1_thread, NULL, player_thread, (void*)player1_board);
    pthread_create(&player2_thread, NULL, player_thread, (void*)player2_board);

    // Wait
    pthread_join(player1_thread, NULL);
    pthread_join(player2_thread, NULL);

    // Destroy mutex
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Fig 8. Main

- **Screenshots of the operation.**

```
Enter the number of ships for both players (minimum 2): 2
Player 1 enter the starting row and column for ship 1 : 1 2
Player 1 enter the starting row and column for ship 2 : 3 4
Player 2 enter the starting row and column for ship 1 : 6 7
Player 2 enter the starting row and column for ship 2 : 5 6
```

Fig 9. Here the user assigned the number of boats per person and their position

```
Player 1:
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . . . . .
3 . . . . . . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
```

```
Player 2:
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . . . . .
3 . . . . . . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
```

Fig 10. Boards of each player

Fig 11. Missed attack of player 1

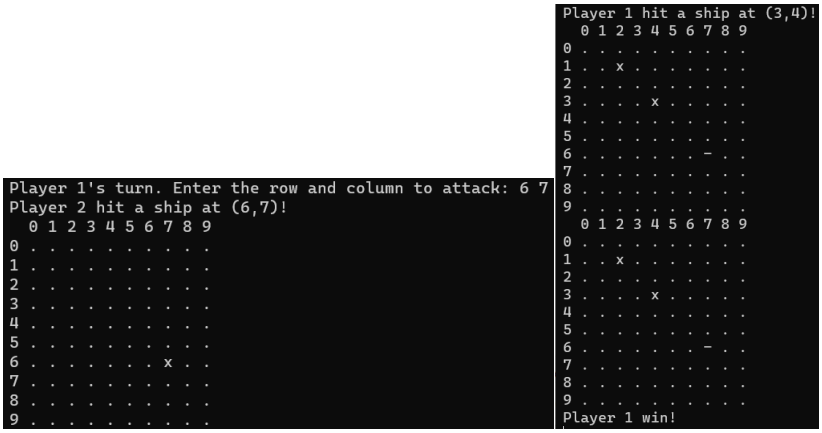


Fig 12. Successful attack of player 2 and Player 1 win

- **Link to code.**

  Battleship

Personal conclusion

In conclusion, it is worth emphasizing the significance of practical activities like the battleship game. It offers an engaging approach to comprehend various operating system concepts like processes, threads, signals, synchronization, and many others. By applying these principles to the game, we gain a deeper understanding of how to utilize these tools correctly, which is crucial in our understanding of computer science. Thus, activities like these can significantly enhance our knowledge and skills in the field of operating systems.

References

Help/guidance/direction for C battleship game. (2010). Retrieved 1 May 2023, from https://cboard.cprogramming.com/c-programming/126395-help-guidance-direction-c-battleship-game.html

C, B., & Ewert, W. (2012). Battleship game in C. Retrieved 3 May 2023, from https://codereview.stackexchange.com/questions/8294/battleship-game-in-c