Cameron Sparks 101181932

Program Pseudo Code

pokemon.h

```
// Include the necessary header files (stdio, stdlib, string, unistd, semaphore, semaphore)
struct Pokemon{
        // Variables needed to store a pokemon's information.
}
struct Shared{
        // Variables needed to be shared between threads.
}
// Functions declarations
```

---

assignment3.c

```
// Include pokemon.h
int main(int argc, char **argv){
        //Create and Initialize variables to store (users choice, number of threads, number of saved files,
Shared data, save file names)
        //Initialize mutex.
        //Welcome the user to the program and enter the function enterCsv(Shared data).
        //Dynamically allocate memory for a new thread, new file name and a new pokemon.

        START INFINITE LOOP:
        //Prompt for the user to select an option (1,2 or 3).
        if(userOption==1){
                // Prompt the user to enter a pokemon type.
                // Open a thread to query the type (function typeSearch(Shared data)).
                // Allocate more memory for a new thread.
        }
        else if(userOption==2){
                START INFINITE LOOP:
                //prompt the user to enter a filename to store query information or quit.
                if(user chooses to quit){
                        //Break out of loop
                }
                if(file can be written to & the file is NEW){
                        //Open a thread to store query information (function saveAll(Shared data)).
                        //Update save file names and allocate more memory for more file names.
                        //Allocate more memory for a new thread and a new save file name.
                        //Break out of loop
                }
                else{
                        //Prompt the user to enter the filename again
                }
                END OF INFINITE LOOP:
        }
        else{
                //Break out of loop
        }
        END OF INFINITE LOOP
        //Stop threads, Print total amount of successful queries and print saved file names.
        //Free dynamically allocated data.
        //EXIT PROGRAM
}
```

---

assignment3functions.c

```
// Include pokemon.h
void enterCsv(Shared *sharedData){
        START INFINITE LOOP:
        //Prompts the user to enter a pokemon csv filename or to quit the program.
        //Saves filename into a variable within sharedData or quit depending on user input.
        // Break out of the loop if the filename was entered correctly or the user quit.
        END INFINITE LOOP:
}

void *saveAll(void *tempData){
        // Typecast tempData to Shared and copy the filename.
        //Lock mutex so other threads can't access the Shared data.
        //Write all queried pokemon data to the chosen file.
        // Unlock mutex so other threads can now access the Shared data
}

void *typeSearch(void *tempData){
        // Typecast tempData to Shared and copy the pokemon type 1.
        // Lock mutex so other threads can't access the Shared data.
        // Create a temporary Pokemon variable to store the pokemons information on each line of the file
        for(length of the file){
                //Parse file information into the temp pokemon variable (Using stringAssign() for strings)
                if(temp pokemon's type 1 == user given type 1){
                        pokemonAssign(sharedData, temp pokemon).
                }
                emptyTempPokemon(memory location of the temp pokemon).
        }
        // Unlock mutex so other threads can now access the Shared data.
}

void stringAssign(char *str, FILE *file){
        //Parse an individual string from the file and store it in the string
}
void emptyTempPokemon(Pokemon *pokemon){
        //Clear the temp pokemons memory to be used again for another pokemon.
}
void pokemonAssign(Shared *sharedData, Pokemon pokemon){
        //Copy all data from the temp pokemon and add it to the list of queried pokemon in the shared
Data.
        //Dynamically allocate more memory for the queried pokemon.
}
```

END OF PSEUDO CODE

1. The program will contain two C files and one header file. The assignment3.c file contains the central functionality of the program including the menu the users interact with. This C file will contain the code that dynamically allocates starting memory, starts/stops threads and then finally frees all the dynamic data. The assignment3functions.c file contains the functions used by threads to either query or save data, as well as a handful of helper functions to parse strings or copy data. Finally the pokemon.h header file contains all the necessary includes to other header files, structures used by both C files and function declarations. The simple answer to why there are two C files is to make the program more modular and separate the main functionally from the thread functionality. The header file is the result of this modularity as both C files require the same includes and structures as well as function declarations.
2. FUNCTIONAL REQUIREMENTS:
    a. The program implements the first use case by going into an infinite loop and prompting the user to enter a CSV file. If the user enters the CSV file then the program breaks out of the loop and starts the main menu. If the user types “q” then the program exits. Then finally if the user enters an incorrect filename then the program asks the user to type their filename again and loops to the start.
    b. The program implements the second use case by opening the given CSV file and parsing each specific string and number into variables within a Pokemon data type. The program then adds all of the separate pokemon’s data into one shared array. Also the program uses multithreading for the parsing, so as the program is parsing it returns back to the menu so the user can immediately interact with the program again.
    c. Essentially the same as the above use case except instead of the thread parsing a CSV file it is writing all of the queried pokemons data to a given file. Due to the use of mutexes for both the query and save threads, the save thread will only write completed queries to the file.
    d. The program implements the fourth use case by running specific thread functions that terminate threads (queries and saves both). Then the program will display the amount of completed queries which in this case is queries that found at least one matching result. Then the program will display the new file names which were stored using a double char pointer or an array of strings.
3. NON-FUNCTIONAL REQUIREMENTS:
    a. The program uses threads to query and save data, so while the threads are running specific tasks in the background the program returns back to the menu immediately for the user to enter another choice. The threads are also stored dynamically so a large amount of threads can be run at once.
4. The string library and functions are used to compare and copy strings from a tempPokemon to the final pokemon array. The unistd library and functions are used to check if a file can be accessed. The semaphore library and functions are used to create mutexes to ensure threads don’t write and read over each other. Then finally the pthread library and functions are used to create the threads.
5. The queried pokemons data, amount of total pokemon, amount of total queries, the pokemon.csv filename and specific thread info will be stored in shared memory that is used by all threads. The data will be protected by mutexes which act as blockades to portions of code where these variables are being written to and read.