```
//
/*
Group Number: E
▪ Group member name: Adam Loeckle
▪ Email: adam.loeckle@okstate.edu
▪ Date: 03/04/2022
Responsibilities:
POSIX Queue/Server side messaging

Completed:
        POSIX Queue/Server-side messaging
        Entirety of final server method completion as illustrated below
        Entirety of final client structure
        Collaboration with Krish to adapt gamelogic into server/client architecture
        Player/computer structures

References:
https://linux.die.net/man/3/
https://github.com/nikhilroxtomar/Multiple-Client-Server-Program-in-C-using-fork

*/

#include "Main.h"

struct Player newPlayer(char *firstname, char *lastname, char *country)
{
        struct Player new_player;

        new_player.score = 0;
```

```
        strcpy(new_player.firstname, firstname);
        strcpy(new_player.lastname, lastname);
        strcpy(new_player.country, country);
        new_player.num_words = 0;
        new_player.num_words_added = 0;
   new_player.resets = 0;

        return new_player;
}

struct Computer newComputer()
{
        struct Computer new_computer;
        new_computer.score = 0;
        new_computer.num_words = 0;
        new_computer.num_words_added = 0;
        new_computer.resets = 0;

        return new_computer;
}

// Opens message queue, should only be ran once.
mqd_t openMsgQueue(char *queue_name)
{
        // Ensures message queue does not already exist and creates a new one
        mq_unlink(queue_name);
        mqd_t mqd = mq_open(queue_name, O_CREAT | O_RDWR, 0600, NULL);

        if (mqd == -1)
```

```
	{
		perror("mq_open");
	}
	else
	{
		printf("MQ was opened \n");
	}
	return mqd;
}

void closeMsgQueue(mqd_t mqd)
{
	mq_close(mqd);
}

void sendPlayerConnectMsg(mqd_t mqd)
{
	mq_send(mqd, "WAITING", 1, 10);
}

int recievePlayerConnectMsg(mqd_t mqd)
{
	int prio = 10;
   struct mq_attr attr;
	mq_getattr(mqd, &attr);
	char *p_buffer = calloc(attr.mq_msgsize, 1);
	int num_msgs = attr.mq_curmsgs;

	unsigned int priority = 0;
```

```
	if (num_msgs != 0)
	{
		if ((mq_receive(mqd, p_buffer, attr.mq_msgsize, &priority)) != -1)
		{
			return 1;
		}
		else
		{
			return 0;
		}
	}
	else
	{
		return 0;
	}
}

void sendDictionaryMsg(mqd_t mqd, char *message, int size)
{
	mq_send(mqd, message, size, 10);
}

char * recieveDictionaryMessage(mqd_t mqd)
{
   struct mq_attr attr;
   char *message = malloc(1024);
	mq_getattr(mqd, &attr);
	char *p_buffer = calloc(attr.mq_msgsize, 1);
```

```
    unsigned int priority = 0;
    if ((mq_receive(mqd, p_buffer, attr.mq_msgsize, &priority)) != -1)
    {
        // Collects message from queue
        if (priority == 10)
        {
            strcpy(message, p_buffer);
            return message;
        }
    }
}

int playerTurn(int newSocket)
{
    srand(time(NULL));
    int rng = (rand()%5)+1; //seeding random number from 1 to 10 for first turn word
    int rng2 = (rand()%10)+1; //seeding random number from 1 to 10 for input.txt
    char rng2char[7];
    sprintf(rng2char, "%d.txt", rng2);
    FILE *fileStream;
    printf("\nrng generated was %d",rng);
    if (rng2==10)
        strcat(fname, "input_");
    else
        strcat(fname, "input_0");
    strcat(fname, rng2char);
    printf("\nWe have chosen %s\n",fname);
    fileStream = fopen (fname, "r");
    fgets (letters, 7, fileStream);
```

```
fclose(fileStream);

// FIRST TURN

// Socket variables
        char buffer[1024];

int first = 1;
int pass = 0;

while(pass < 4)
{
   // Sends letters
   bzero(buffer, sizeof(buffer));
   strcpy(buffer, letters);
   send(newSocket, buffer, 1024, 0);

   int resets = 0;
   while (resets < 3)
   {
      if (first == 1)
      {
         // Sends starting character
         bzero(buffer, sizeof(buffer));
         strcpy(buffer, &letters[rng]);
         send(newSocket, buffer, 1024, 0);

         // Client word
         bzero(buffer, sizeof(buffer));
```

```
recv(newSocket, buffer, 1024, 0);

if (strcmp(buffer, "pass") == 0)
{
   pass++;
   break;
}

if (buffer[0] != letters[rng])
{
   first = 1;
   bzero(buffer, sizeof(buffer));
   strcpy(buffer, "INCORRECT");
   send(newSocket, buffer, 1024, 0);
   resets++;
   continue;
}
else
{
   // Game logic
   strcpy(prev, buffer);
   if (gameLogic(newSocket, buffer) == 0)
   {
      resets++;
      continue;
   }
   else
   {
      first = 0;
```

```
            pass = 0;
            addPlayerScore(added_player);
            if (inputCheck() == 0)
            {
               // Send a different message, check for message on client
            }
            break;
         }
      }
   }
   else
   {
      // Game logic

      // Send number of used words
      uint32_t converted = htonl(noUsedWords);
      send(newSocket, &converted, sizeof(converted), 0);

      // Send used words in for loop
      for (int i = 0; i <= noUsedWords; i++)
      {
         bzero(buffer, sizeof(buffer));
         strcpy(buffer, usedWords[i]);
         send(newSocket, buffer, sizeof(buffer), 0);
      }

      // Client word
      bzero(buffer, sizeof(buffer));
      recv(newSocket, buffer, 1024, 0);
```

```
        if (strcmp(buffer, "pass") == 0)
        {
            pass++;
            break;
        }

        strcpy(prev, buffer);
        if (gameLogic(newSocket, buffer) == 0)
        {
            resets++;
            continue;
        }
        else
        {
            pass = 0;
            addPlayerScore(added_player);
            if (inputCheck() == 0)
            {
                // Send a different message, check for message on client
            }
            break;
        }
    }
}

if (computerTurn() == 0)
{
    // Computer passed
```

```
            bzero(buffer, sizeof(buffer));
            strcpy(buffer, "COMP PASSED");
            send(newSocket, buffer, sizeof(buffer), 0);
            pass++;
        }
        else
        {
            // Computer was successful add points
            addComputerScore(added_computer);
            bzero(buffer, sizeof(buffer));
            strcpy(buffer, "COMP CORRECT");
            send(newSocket, buffer, sizeof(buffer), 0);
            pass = 0;
        }

        for (int i = 0; i <= noUsedWords; i++)
        {
            printf("\nWORD USED: %s\n", usedWords[i]);
        }
    }
    return 0;
}

int createServer()
{
    int sockfd, ret, newSocket;
            struct sockaddr_in serverAddr, newAddr;
            socklen_t addr_size;
            char buffer[1024];
```

```
	pid_t childpid;

	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	memset(&serverAddr, '\0', sizeof(serverAddr));

	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(PORT);
	serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

	ret = bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
	if(ret < 0)
	{
		printf("ERROR: Could not bind to port.\n");
		exit(1);
	}
	printf("CONSOLE: Binded to port %d\n", 4444);
if(listen(sockfd, 10) == 0){
		printf("[+]Listening..\n..\n\n");
	} else {
		printf("[-]Error in binding.\n");
	}

	// Player information
	char firstname[50];
char lastname[50];
char country[50];

	while(1)
	{
```

```
		newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
		if(newSocket < 0)
		{
			exit(1);
		}
		printf("Connection accepted from %s:%d\n", inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));

		if((childpid = fork()) == 0)
	{
			close(sockfd);
			while(1)
			{
				recv(newSocket, buffer, 1024, 0);
				printf("%s\n", buffer);
		if(strcmp(buffer, "1") == 0)
				{
					// Single player game

	  // Receiving player information
	  recv(newSocket, buffer, 1024, 0);
	  strcpy(firstname, buffer);
	  bzero(buffer, sizeof(buffer));

	  recv(newSocket, buffer, 1024, 0);
	  strcpy(lastname, buffer);
	  bzero(buffer, sizeof(buffer));

	  recv(newSocket, buffer, 1024, 0);
```

```
                strcpy(country, buffer);
                bzero(buffer, sizeof(buffer));

                // Create new player and computer struct
                added_player = newPlayer(firstname, lastname, country);
                                        added_computer = newComputer();

                                        printf("First: %s Last: %s Country: %s", added_player.firstname,
                                        added_player.lastname, added_player.country);

                                        bzero(firstname, sizeof(firstname));
                                        bzero(lastname, sizeof(lastname));
                                        bzero(country, sizeof(country));

                                        // Game starts
                                        if(playerTurn(newSocket) == 0)
                                        {
                    // SCOREBOARD METHOD HERE, MAKE SCOREBOARD METHOD AND PUT IT ABOVE
                    // singlePlayerScoreboard();
                    // NEEDS TO SEND CLIENT SCORE OF PLAYER AND COMPUTER
                    // NEEDS TO LET CLIENT KNOW IF THEY WERE ADDED TO SINGLE PLAYER SCOREBOARD
FILE
                    // IF PLAYER HAS HIGHER SCORE THAN COMPUTER ADD THEM TO SINGLE PLAYER
SCOREBOARD FILE
                                        }
                                }
            if(strcmp(buffer, "2") == 0)
                                {
                                        // POSIX queues
```

```
                                        //mqd_t waiting_players =
openMsgQueue("/Waiting_players");

                                        // Receiving player information
            recv(newSocket, buffer, 1024, 0);
            strcpy(firstname, buffer);
            bzero(buffer, sizeof(buffer));

            recv(newSocket, buffer, 1024, 0);
            strcpy(lastname, buffer);
            bzero(buffer, sizeof(buffer));

            recv(newSocket, buffer, 1024, 0);
            strcpy(country, buffer);
            bzero(buffer, sizeof(buffer));

            // Create new player struct
            struct Player added_player = newPlayer(firstname, lastname, country);

            printf("First name: %s\n", added_player.firstname);
            printf("Last name: %s\n", added_player.lastname);
            printf("Country: %s\n", added_player.country);

                                        //if (recievePlayerConnectMsg(waiting_players) == 1)
                                        //{
                                                // Starts multiplayer game with other connected player
                                        //}
                                        //else
                                        //{
```

```
                                        // Sends message to POSIX queue that this client is
waiting.
                                        // Asks client if they want to wait after two minutes of
waiting.
                                        //sendPlayerConnectMsg(waiting_players);
                                //}
                        }
        // Clean client exit
                        if(strcmp(buffer, "3") == 0)
                        {
                                printf("Disconnected from %s:%d\n",
inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));
                                break;
                        }
        else
        {
           printf("Disconnected from %s:%d\n", inet_ntoa(newAddr.sin_addr),
ntohs(newAddr.sin_port));
                                break;
        }
                        }
                }
        }
        close(newSocket);
        return 0;
}


int clientGame()
{
        int clientSocket, ret;
```

```
        struct sockaddr_in serverAddr;
        char buffer[1024];

        clientSocket = socket(AF_INET, SOCK_STREAM, 0);
        if(clientSocket < 0)
        {
                printf("ERROR: Cannot create client socket.\n");
                exit(1);
        }
        printf("CONSOLE: Created client socket.\n");

        memset(&serverAddr, '\0', sizeof(serverAddr));
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(PORT);
        serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

        ret = connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
        if(ret < 0)
        {
                printf("ERROR: Cannot connect to server.\n");
                exit(1);
        }
        printf("CONSOLE: Connected to Server.\n");

while(1)
{
   printf("Welcome to the word game.\n");
   printf("Please choose an option below (1 for Singleplayer, 2 for Multiplayer, 3 for exit)\n");
   printf("1) Singleplayer\n");
```

```
printf("2) Multiplayer\n");
printf("3) Exit\n");
printf("> ");

scanf("%s", &buffer[0]);
send(clientSocket, buffer, strlen(buffer), 0);
printf("Input: %s\n", buffer);

if (strcmp(buffer, "1") == 0)
{
   printf("\nSingle Player Mode\n");
   printf("Enter your first name: ");
   bzero(buffer, sizeof(buffer));
   scanf("%s", &buffer[0]);
   send(clientSocket, buffer, strlen(buffer), 0);

   printf("\nEnter your last name: ");
   bzero(buffer, sizeof(buffer));
   scanf("%s", &buffer[0]);
   send(clientSocket, buffer, strlen(buffer), 0);

   printf("\nEnter your country: ");
   bzero(buffer, sizeof(buffer));
   scanf("%s", &buffer[0]);
   send(clientSocket, buffer, strlen(buffer), 0);

   int first = 1;
   int pass = 0;
   while(pass < 4)
```

```
{
    bzero(buffer, sizeof(buffer));
    recv(clientSocket, buffer, 1024, 0);
    printf("Letters: %s\n", buffer);

    int resets = 0;
    while (resets < 3)
    {
        if (first == 1)
        {
            // Recieves starting character
            char starting_char = '0';
            bzero(buffer, sizeof(buffer));
            recv(clientSocket, buffer, 1024, 0);
            strcpy(&starting_char, buffer);
            printf("The starting character is: %c\n", starting_char);

            // First words submission
            printf("\nEnter your word: ");
            bzero(buffer, sizeof(buffer));
            scanf("%s", &buffer[0]);
            send(clientSocket, buffer, 1024, 0);

            if (strcmp(buffer, "pass") == 0)
            {
                pass++;
                break;
            }
```

```
    // Receives answer
    bzero(buffer, sizeof(buffer));
    recv(clientSocket, buffer, 1024, 0);

    if (strcmp(buffer, "INCORRECT") == 0)
    {
        printf("INCORRECT\n");
        resets++;
        continue;
    }
    if (strcmp(buffer, "CORRECT") == 0)
    {
        first = 0;
        pass = 0;
        printf("USER SCORED\n");
        // Check for bonus points
        break;
    }
}
else
{
    // Recieves number of used words
    // Recieves used words
    char usedWords[100][100];
    uint32_t converted = 0;
    recv(clientSocket, &converted, sizeof(converted), 0);
    uint32_t noUsedWords = htonl(converted);
    printf("NUMBER OF WORDS: %d\n", noUsedWords);
```

```
printf("WORDS USED: ");
for (int i = 0; i <= noUsedWords; i++)
{
   bzero(buffer, sizeof(buffer));
   recv(clientSocket, buffer, sizeof(buffer), 0);
   printf("%s ", buffer);
}
printf("\n");

// First words submission
printf("\nEnter your word: ");
bzero(buffer, sizeof(buffer));
scanf("%s", &buffer[0]);
send(clientSocket, buffer, 1024, 0);

if (strcmp(buffer, "pass") == 0)
{
   pass++;
   break;
}

// Receives answer
bzero(buffer, sizeof(buffer));
recv(clientSocket, buffer, 1024, 0);

if (strcmp(buffer, "INCORRECT") == 0)
{
   printf("INCORRECT\n");
   resets++;
```

```
                continue;
            }
            if (strcmp(buffer, "CORRECT") == 0)
            {
                pass = 0;
                printf("USER SCORED\n");
                break;
            }
          }
       }
       // Computer plays
       // Recieves if computer scored or not
       bzero(buffer, sizeof(buffer));
       recv(clientSocket, buffer, sizeof(buffer), 0);
       printf("COMPUTER BUFF: %s\n", buffer);
       if (strcmp(buffer, "COMP CORRECT") == 0)
       {
          pass = 0;
       }
       if (strcmp(buffer, "COMP PASSED") == 0)
       {
          pass++;
       }
    }

    // SCOREBOARD OUTPUT HERE
}
if (strcmp(buffer, "2") == 0)
{
```

```
    }
  }
}
```