```
// Generic includes
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <time.h>
#include <stdint.h>

// Socket/network includes
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <fcntl.h>

#include <pthread.h>

// Mqueue include
#include <mqueue.h>

// Error handling
#include <errno.h>

#define MAX 50
#define PORT 8000
```

```
#define SA struct sockaddr

struct Player added_player;
struct Computer added_computer;
char prev[100];
char new[100];
char newf[101] = "";
char newadd[101] = "\n";
char usedWords[100][100];
uint32_t noUsedWords = 1;
char letters [6];
char fname[14] = "";
size_t nnew;

// Player struct
struct Player
{
        int score;
        char firstname[50];
        char lastname[50];
        char country[50];
        int num_words;
        int num_words_added;
        int resets;
} Player;

struct Computer
{
        int score;
```

```
        int num_words;
        int num_words_added;
        int resets;
} Computer;

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 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;
        }
    }
}

void minusPlayerScore(struct Player new_player, int num)
{
    new_player.score += num;
}

void addPlayerScore(struct Player new_player)
{
    size_t length = nnew;
    if (length == 3 || length == 4)
    {
        new_player.score += 1;
    }
```

```
    if (length == 5)
    {
        new_player.score += 2;
    }
    if (length == 6)
    {
        new_player.score += 3;
    }
    if (length == 7)
    {
        new_player.score += 4;
    }
    if (length >= 8)
    {
        new_player.score += 11;
    }
}

void addComputerScore(struct Computer new_computer)
{
    size_t length = nnew;
    if (length == 3 || length == 4)
    {
        new_computer.score += 1;
    }
    if (length == 5)
    {
        new_computer.score += 2;
    }
```

```
    if (length == 6)
    {
        new_computer.score += 3;
    }
    if (length == 7)
    {
        new_computer.score += 4;
    }
    if (length >= 8)
    {
        new_computer.score += 11;
    }
}


int computerTurn()
{
    //Computer will check input file line by line for usable words (if they're wordbuilder words that haven't been used yet, it plays them)
    int prevlen = 0;
    for (int i = 0; prev[i]!='\0'; i++)
    {
        prevlen++;
    }
    FILE* filePointer;
    int wordExist=0;
    int bufferLength = 99;
    char line[bufferLength];
    int linelen = 0;
    int disallowed = 0;
```

```
int run = 1;
int skip = 0;
strcpy(newf,""); //"new\n"
strcpy(newadd,"\n");    // "\nnew\n"
filePointer = fopen(fname, "r");
while(fgets(line, bufferLength, filePointer) && run!=0)
{
   if (skip!=3)
   {
      skip++;
      continue;
   }
   disallowed = 0;
   strcpy(new,"");
   int c = getc(filePointer);
   if (c == EOF)
   {
      printf("\nComputer could not find appropriate word");
      return 0;
   }
   else
   {
      ungetc(c,filePointer);
   }
   linelen=0;
   for (int i = 0; line[i]!='\0'; i++)
   {
      linelen++;
   }
```

```
    for (int i = 0; i < linelen-1; i++)
    {
       new[i] = line[i];
    }
    new[linelen-1] = '\0';
    strcpy(newf,"");
    strcpy(newadd,"\n");
    strcat(newf, new);
    //strcat(newf,"\n");
    strcat(newadd, new);
    strcat(newadd,"\n");
    size_t n = sizeof(prev)/sizeof(char);
    nnew = sizeof(new)/sizeof(char);
    size_t nnewf = sizeof(newf)/sizeof(char);
    for (int i=0; i<n;i++)
    {
       for (int x = 0; x < nnew && disallowed==0 && new[x]!='\0'; x++)
       {
          //make sure no disallowed characters are in it
          for (int y = 0; y < 6 && new[x]!='\0'; y++)
          {
             //printf("\n Iteration %d y iteration %d we're looking at %c in new and %c in letters\n", x, y,
new[x], letters[y]);
             if (new[x]!=letters[y])
             {
                if (letters[y+1]=='\0')
                   disallowed=1;
                else
                   continue;
```

```
            }
            else
               break;
         }
      }
      if (disallowed==0)
      {
         if (new[0]==prev[i])
         {
            //printf("\nUsed correct characters!");
            int j = i;
            int k = 0;
            while ((j<n) && (new[k]==prev[j]) && !((new[k]=='\0') && (prev[j]=='\0')))
            {
               //printf("\n Iteration %d we're looking at %c in new and %c in prev\n", i, prev[j], new[k]);
               j++;
               k++;
               //printf("The value of j is %d k is %d n is %d", j,k,n);
            }
            if ((j==n) || (prev[j]=='\0') || ((new[k]=='\0') && (prev[j]=='\0')))
            {
               //printf("\nComputer's Word is valid!");
               //check if word has already been used
https://stackoverflow.com/questions/63132911/check-if-a-string-is-included-in-an-array-and-append-if-not-c
               int dup = 0;
               for (int j = 0; j < 100; j++)
               {
                  if(strcmp(new, usedWords[j]) == 0)
```

```
                {
                    printf("\nWord %10s has been found in %d of usedWords as
%10s",new,j,usedWords[j]);
                    dup = 1;   // got a duplicate
                }
            }
            if (dup == 0) {    // not a duplicate: add it to usedWords
                strcpy(usedWords[noUsedWords+1], new);
                noUsedWords += 1;
            }
            if(dup)
            {
                printf("\nWORD HAS ALREADY BEEN USED THIS GAME.");
                //penalise
                for (int i = 0; i<=noUsedWords;i++)
                {
                    printf("\nUsed word %d of %d is %s",i,noUsedWords,usedWords[i]);
                }
                break;
            }
            else
            {
                //printf("\nWord has NOT been used this game. Added to used words.");
                printf("\nComputer played the word: %s",new);
                for (int i = 0; i<=noUsedWords;i++)
                {
                    printf("\nUsed word %d of %d is %s",i,noUsedWords,usedWords[i]);
                }
                strcpy(prev,new);
```

```
                strcpy(new,"");
                run = 0;
                return 1;

            }
        }
        else
        {
            if (j<n)
                continue;
            printf("\n Invalid but part of it was at some point");
            //in theory we should never be here?
            //penalise
        }
      }
      if (i==(n-1))
      {
        //printf("\nComputer's Word is not valid.");
        //penalise
      }
    }
    else
    {
      printf("\nWord contains disallowed characters.");
      //penalise
    }
  }
}
fclose(filePointer);
```

```
}

void dictionaryCheck(mqd_t dictionary, size_t nnewf, char *lowernew, int newSocket)
{
   for(int w = 0; w<nnewf; w++)
   {
      lowernew[w] = tolower(newf[w]);
   }
   FILE* filePointerd;
   int wordExistd=0;
   int bufferLengthd = 255;
   char lined[bufferLengthd];
   int linedlen = 0;
   int lowernewlen = 0;
   printf("\nChecking if %s is a valid dictionary word\n", lowernew);
   filePointerd = fopen("dictionary.txt", "r");
   for (int i = 0; lowernew[i]!='\0'; i++)
   {
      lowernewlen++;
   }
   while(fgets(lined, bufferLengthd, filePointerd))
   {
      linedlen=0;
      for (int i = 0; lined[i]!='\0'; i++)
      {
         linedlen++;
      }
      char *ptrd = strstr(lined, lowernew);
      if (ptrd != NULL && (linedlen==lowernewlen))
```

```
        {
            //printf("\nline is %d characters long and newf is %d long",linedlen,lowernewlen);
            wordExistd=1;
            break;
        }
    }
    bzero(lowernew,sizeof(lowernew));
    fclose(filePointerd);
    if (wordExistd==1)
    {
        sendDictionaryMsg(dictionary, "CORRECT", 7);
    }
    else
    {
        sendDictionaryMsg(dictionary, "INCORRECT", 9);
    }
}

int inputCheck()
{
    //check if word is already in the input file https://www.efaculty.in/c-programs/check-whether-a-given-word-exists-in-a-file-or-not-program-in-c/
    FILE* filePointer;
    int wordExist=0;
    int bufferLength = 255;
    char line[bufferLength];
    int linelen = 0;
    int newflen = 0;
    for (int i = 0; newf[i]!='\0'; i++)
```

```
    {
        newflen++;
    }
    filePointer = fopen(fname, "r");
    while(fgets(line, bufferLength, filePointer))
    {
        linelen=0;
        for (int i = 0; line[i]!='\0'; i++)
        {
            linelen++;
        }
        char *ptr = strstr(line, newf); //check newf in debugger
        if (ptr != NULL && (linelen==newflen))
        {
            //printf("\nINPUT.txt line is %d characters long and newf is %d long",linelen,newflen);
            wordExist=1;
            return 0;
        }
    }


    if (wordExist!=1)
    {
        //add word to input file https://stackoverflow.com/questions/19429138/append-to-the-end-of-a-file-in-c
        FILE * fptr;
        fptr = fopen(fname, "a");
        fputs(newadd, fptr);
        fclose (fptr);
    }
```

```
    strcpy(prev,new);
    strcpy(new,"");

    return 1;
    fclose(filePointer);
}

int gameLogic(int newSocket, char *buffer)
{
    strcpy(new, buffer);
    strcpy(newf,"");
    strcpy(newadd,"\n");
    strcat(newf, new);
    strcat(newf,"\n");
    strcat(newadd, newf);
    size_t n = sizeof(prev)/sizeof(char);
    size_t nnew = sizeof(new)/sizeof(char);
    size_t nnewf = sizeof(newf)/sizeof(char);
    char lowernew[101];

    mqd_t dictionary = openMsgQueue("/Dictionary_Check");

    printf("GOT TO THE LOOP\n");
    int disallowed = 0;
    for (int i=0; i<n;i++)
    {
        for (int x = 0; x < nnew && disallowed==0 && new[x]!='\0'; x++)
        {
            for (int y = 0; y < 6 && new[x]!='\0'; y++)
```

```
        {
            //printf("\n Iteration %d y iteration %d we're looking at %c in new and %c in letters\n", x, y, new[x], letters[y]);
            if (new[x]!=letters[y])
            {
                if (letters[y+1]=='\0')
                    disallowed=1;
                else
                    continue;
            }
            else
                break;
        }
    }
    if (disallowed==0)
    {
        if (new[0]==prev[i])
        {
            printf("\nUsed correct characters!\n");
            int j = i;
            int k = 0;
            while ((j<n) && (new[k]==prev[j]) && !((new[k]=='\0') && (prev[j]=='\0')))
            {
                //printf("\n Iteration %d we're looking at %c in new and %c in prev\n", i, prev[j], new[k]);
                j++;
                k++;
                //printf("The value of j is %d k is %d n is %d", j,k,n);
            }
            if ((j==n) || (prev[j]=='\0') || ((new[k]=='\0') && (prev[j]=='\0')))
```

```
{
   printf("\nWord is valid!\n");
   //check if word is a dictionary word
   printf("\nConverting %s to lower\n",new);
   for(int w = 0; w<nnewf; w++)
   {
      lowernew[w] = tolower(newf[w]);
   }

   // Dictionary
   // int dictionaryCheck(size_t nnewf, char *lowernew, int newSocket)
   // FORKING ------------------------------------------------------------
   dictionaryCheck(dictionary, nnewf, lowernew, newSocket);

   // Recieve dictionary check posix message, return 0 if incorrect
   if (strcmp(recieveDictionaryMessage(dictionary), "INCORRECT") == 0)
   {
      bzero(buffer, sizeof(buffer));
      strcpy(buffer, "INCORRECT");
      printf("INCORRECT DICT\n");
      minusPlayerScore(added_player, -1);
      send(newSocket, buffer, 1024, 0);
      return 0;
   }

   wait(NULL);
   // Used words check
```

```
        //check if word has already been used
https://stackoverflow.com/questions/63132911/check-if-a-string-is-included-in-an-array-and-append-if-not-c
        int dup = 0;
        for (int j = 0; j < 100; j++)
        {
           if(strcmp(new, usedWords[j]) == 0)
           {
              dup = 1;   // got a duplicate
              break;
           }
        }
        if (dup == 0)
        {   // not a duplicate: add it to usedWords
           strcpy(usedWords[noUsedWords+1], new);
           noUsedWords += 1;

           // Send correct message
           printf("CORRECT: %s", buffer);
           bzero(buffer, sizeof(buffer));
           strcpy(buffer, "CORRECT");
           send(newSocket, buffer, 1024, 0);
           strcpy(prev, new);
           return 1;
        }
        if(dup)
        {
           //penalise
           bzero(buffer, sizeof(buffer));
```

```
            strcpy(buffer, "INCORRECT");
            printf("INCORRECT DUPLICATE\n");
            minusPlayerScore(added_player, -2);
            send(newSocket, buffer, 1024, 0);
            return 0;
          }
        }
        else
        {
          bzero(buffer, sizeof(buffer));
          strcpy(buffer, "INCORRECT");
          printf("INCORRECT NOT VALID\n");
          minusPlayerScore(added_player, -1);
          send(newSocket, buffer, 1024, 0);
          return 0;
        }
    }
    if(i==(n-1))
    {
        bzero(buffer, sizeof(buffer));
        strcpy(buffer, "INCORRECT");
        printf("INCORRECT CHARS");
        minusPlayerScore(added_player, -1);
        send(newSocket, buffer, 1024, 0);
        return 0;
    }
}
else
{
```

```
            bzero(buffer, sizeof(buffer));
            strcpy(buffer, "INCORRECT");
            printf("INCORRECT DISALLOWED\n");
            minusPlayerScore(added_player, -1);
            send(newSocket, buffer, 1024, 0);
            return 0;
        }
    }
}

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 = 1; //(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 main()
{
   createServer();
   return 0;
}
```