Jake Chou
Nicholas Fong
Jonathan Hong
May 13, 2014
Network Centric Programming

Final Project Report

Our game consists of two players, one of which is the host. The objective is simple, to not be the player that takes the last fish stick. The initial number of fish sticks is between 20 and 40. We selected this range because we felt it would produce a round that is not too short and not too long. On each player's turn, the player can take 1, 2, or 3 fish sticks. The two players take turns removing fish sticks until a player takes the last fish stick; whichever player this is loses the game.

In order to play a game, both players must run the source code. To host a game, a user types in the following three arguments in the terminal: ./filename host <port>. For a player to join this game, another user types in the following arguments: ./filename join <ip:port>. If you are the host, you will designate the port number. If you are the player that joins, you must supply the ip address to connect to. Our implementation uses TCP in order to relay messages stored in a buffer back and forth between the host and joiner. In the function *hostGame()*, we implement a basic TCP server. The basic steps of this function are as follows:

- Socket() - Create a socket
- Bind() - Bind the socket
- Listen() - Listen for connections
- Accept() - Accept a connection from the joiner

Once a connection is established with the joiner, we can begin the game. In order to determine the initial number of fish sticks, we utilize the *rand()* function with a specific range of 20 to 40. We initialize a

random seed using the *time()* function (*srand(time(NULL))*). The number of fish sticks is kept in the integer variable *gameObj*. The host is always the first player to remove fish sticks. Each move is carried out through the function *userMove()*. When it is a player's turn, the player is prompted to enter 1, 2, or 3 on the terminal to indicate how many fish sticks to remove. The user's input is read using *scanf()* and is kept in the integer variable *input*. If the player enters something other than 1, 2, or 3, the player is prompted to re-enter a valid input. We subtract this input from *gameObj* and put the new *gameObj* into the buffer using *snprintf()*. This buffer is now sent to the opponent (represented by *clientfd*); this completes the first move. The host now waits for the opponent's move and *recv()* is called when the host gets the modified buffer. In the *recv()* function, we include the flag MSG_WAITALL, which requests the operation to block until the full request is satisfied. Before using this flag, we ran into an issue in which the host would send the buffer and receive the same buffer without allowing the opponent to remove fish sticks. With this flag, we allow the host and joiner to take turns modifying and sending the buffer. This entire process resides within a while loop and continues while the fish sticks count is greater than 0.

To join a game, we have a function called *joinGame()*. The first step is to put the port and ip address into the character pointers *ip* and *portstr* using *strtok()*. We need these values in order to set the address values. The basic steps of the *joinGame()* method are as follows:

- Socket() - Create a socket
- Bind() - Bind the socket
- Connect() - Connect to host

Once a connection is established with the host, we can receive the host's first move through *recv()*. When the joiner receives the host's first move, the joiner inputs 1, 2, or 3 fish sticks to remove from

gameObj. The joiner now works in a similar fashion as the host. In a while loop, *userMove()* is called, the buffer is sent, the joiner waits for the host's turn, and the joiner receives back the modified buffer.

In order to determine if the host or joiner wins, we must do a check every time the buffer is sent and received. This check is implemented by the function *printObj()*. The game terminates when the number of fish sticks (*gameObj*) reaches 0. If *gameObj* is greater than 0, we print out the remaining fish sticks left in play. If *gameObj* is less than or equal to 0, the player that caused the fish sticks count to reach 0 with his or her last turn will be deemed the loser and the other player the winner. On the winner's terminal, the message "YOU WIN!" is printed and on the loser's terminal, the message "YOU LOSE!" is printed.

Throughout the course, we learned many concepts to help us on this project. One part that we learned and used were sockets. We created, binded, listened, and accepted sockets. socket(AF_INET, SOCK_STREAM, 0) was used to create the socket. bind(sockfd, (struct sockaddr*) was used to bind it and listen(sockfd, 5) was used to listen and accept the socket. We were also able to use and assign address values in order to connect (AF_INET). This led to connecting to ip addresses and port numbers for our computers to interact and communicate (addr.sin_addr.s_addr = htonl(INADDR_ANY) and (addr.sin_port = htons(atoi(port)). Lastly, we were able to modify and control our buffers as well as sending and receiving them. Some functions we implemented were send(clientfd, buf, sizeof(buf), 0) and recv(clientfd, buf, sizeof(buf), MSG_WAITALL)).

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MAXLENGTH 128

// Function prototypes
int hostGame(char*);
int joinGame(char*);
int userMove(int, int);
int printObj(int, int);

char buf[MAXLENGTH];

int main(int argc, char** argv){

  if(argc != 3){
       printf("The proper usage is:\n  ./filename host <port>\n  ./filename join
<ip:port>\n");}

  if(strcmp(argv[1], "host") == 0){
       hostGame(argv[2]);}

  else if(strcmp(argv[1], "join") == 0){
       joinGame(argv[2]);}

  return 0;
}

// Host waits till a player joins and the game can start
int hostGame(char* port){
  int sockfd;
  int clientfd;
  struct sockaddr_in addr;
  int recvd = 0;

  printf("Hosting game... \n");

  // Socket
  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
       perror("Error: hostGame() socket error.");
       exit(1);
  }
```

```
// Set address values
bzero(&addr, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(atoi(port));

// Bind the socket
if (bind(sockfd, (struct sockaddr*) &addr, sizeof(struct sockaddr_in)) < 0){
     perror("Error: hostGame() bind error.");
     exit(1);
}

// Listen for connections
if(listen(sockfd, 5) < 0){
     perror("Error: hostGame() listen error.");
     exit(1);
}

printf("Waiting for other player... \n");

// Accept a connection
if((clientfd = accept(sockfd, 0,0)) < 0){
     perror("Error: hostGame() accept error.");
     exit(1);
}

printf("Player connected. Game start.\n");

// Initialize random seed
srand(time(NULL));

// Generate a random number
int gameObj = (rand() % 20) + 20;
printf("Number of fishsticks: %d\n", gameObj);

// Game starts
while(gameObj > 0){
     gameObj = userMove(gameObj, clientfd);
     if(send(clientfd, buf, sizeof(buf),0) < 0){
     perror("Error: hostGame send error.");
     exit(1);
     }
     printObj(gameObj, 0);

     printf("Waiting for opponent's move...\n");
     if((recvd = recv(clientfd, buf, sizeof(buf), MSG_WAITALL)) < 0){
     perror("Error: joinGame recv error.");
     exit(1);
     }
     gameObj = atoi(buf);
```

```
        printObj(gameObj, 1);
  }
}

// Join a game initialized by a host
int joinGame(char* ipaddr){
  char* ip;
  char* portstr;
  int sockfd;
  int clientfd;
  int recvd = 0;
  int gameObj = 1;

  // Put ip address into ip
  portstr = strtok(ipaddr, ":");
  ip = portstr;

  // Put port number into portstr as a string
  portstr = strtok(NULL, ":");

  printf("Connecting to %s:%s...\n", ip, portstr);

  // Set address values
  struct sockaddr_in addr;
  bzero(&addr, sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = inet_addr(ip);
  addr.sin_port = htons(atoi(portstr));

  // Socket
  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        perror("Error: joinGame() socket error.");
        exit(1);
  }

  // Set address values
  struct sockaddr_in newaddr;
  bzero(&newaddr, sizeof(newaddr));
  newaddr.sin_family = AF_INET;
  newaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  newaddr.sin_port = htons(atoi(portstr));

  // Bind the socket
  if (bind(sockfd, (struct sockaddr*) &newaddr, sizeof(struct sockaddr_in)) < 0){
        perror("Error: joinGame() bind error.");
        exit(1);
  }

  // Connect to host
  if((clientfd = connect(sockfd, (struct sockaddr *) &addr, sizeof(addr))) < 0){
```

```
        printf("Error: joinGame() unable to connect to host.\n");
        exit(1);
  }
  // Receive host’s first move
  printf("Waiting for opponent's move...\n");
  if((recvd = recv(sockfd, buf, sizeof(buf), MSG_WAITALL)) < 0){
        perror("Error: joinGame recv error.");
        exit(1);
  }

  gameObj = atoi(buf);
  printf("%d fishsticks left.\n", gameObj);

  // Game starts
  while(gameObj > 0){
        gameObj = userMove(gameObj, sockfd);
        write(sockfd, buf, sizeof(buf));
        printObj(gameObj, 0);

        printf("Waiting for opponent's move...\n");
        if((recvd = recv(sockfd, buf, sizeof(buf), MSG_WAITALL)) < 0){
        perror("Error: joinGame recv error.");
        exit(1);
        }
        gameObj = atoi(buf);
        printObj(gameObj, 1);
  }
  return 0;
}

// Function that prints number of remaining fish sticks and also determines if player has
// won or lost
int printObj(int gameObj, int player){
  if(gameObj > 0){
        printf("%d fishsticks left.\n", gameObj);
  }
  // Game ends when no more fish sticks are left
  if(gameObj <= 0){
        printf("No fishsticks left.\n");
        if(player == 0){
         printf("YOU LOSE!\n");}
        else if(player == 1){
        printf("YOU WIN!\n");}
        exit(1);
  }
  return 0;
}
```

```
// Function that represents a player’s turn
int userMove(int gameObj, int sockfd){
  int input;
  // Player can only remove 1, 2, or 3 fish sticks per turn
  printf("Enter a number of fishsticks to remove (1, 2, or 3): ");
  scanf("%d", &input);
  // Check if input is valid, if not, ask the user to re-enter input
  if (input == 1 || input == 2 || input == 3) {
        printf("You took %d fishsticks. \n", input);
        gameObj = gameObj - input;
        snprintf(buf, sizeof(buf), "%d", gameObj);
  }
  else {
        printf("Please input a valid number of fishsticks. \n");
        userMove(gameObj, sockfd);
  }
  return gameObj;
}
```