Paper, Scissors, Rock Project

Abraham J. Reines

April 26, 2024

1 Project Overview

This project has concurrent programming to simulate game "Paper, Scissors, Rock" using multiple processes in Unix. system has three components: ‘play‘ program, ‘referee‘ program, and multiple ‘player‘ programs.

2 Implementation Details

2.1 Play Program

‘play‘ program initiates game by creating player processes and a referee process. All processes are correctly synchronized and communicates game results.

2.2 Referee Program

referee program controls game flow, receives choices from players, determines winner, and sends results back to play program.

2.3 Player Programs

Each player program represents a participant in game. These programs send their game choices to referee and wait for results.

3 Execution

3.1 Compiling Programs

To compile programs use GNU Compiler Collection (gcc).

```
make -f makefile.mak
```

3.2 Running Simulation

To run simulation open terminal windows. execute programs different terminals:

1. **Terminal 1: Start Play Program** In first terminal,

   ```
   ./play
   ```

2. **Terminal 2: Start Referee Program** Open a second terminal and start ‘referee‘ program:

   ```
   ./referee
   ```

3. **Terminal 3 : Start Player Programs** Each player will run in a separate terminal.

   ```
   ./player
   ```

   Repeat this step in each new terminal for additional players.

Note: ‘play‘ program is executed before ‘referee‘ and ‘player‘ programs.

4 Project Execution and Compliance Report

4.1 Execution on 'stu'

This project was tested to compile and run without errors on 'stu' platform.

4.2 Project Specifications

Project adheres to specifications. socket programming for client/server model and creation of sub-processes for Paper Scissors Rock (PSR) game, conform with project requirements.

4.3 Development and Debugging Process

Throughout development process, no significant issues were encountered.

5 Code Listings

5.1 Play Program

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

/**
 * Main function for "Paper, Scissors, Rock".
 *
 * @param arg1 number of command-line arguments.
 * @param arg2 array of strings with command-line arguments.
 * @return exit status of program.
 */
int main(int arg1, char *arg2[]) {
    if (arg1 != 2) {
        fprintf(stderr, "Usage: %s <number_of_rounds>\n", arg2[0]);
        exit(EXIT_FAILURE);
    }

    printf("Written by: Abraham J. Reines\n");
    printf("Paper, Scissors, Rock: %s iterations\n", arg2[1]);

    pid_t pid = fork();
    if (pid == 0) {
        // Launch referee
        execl("./referee", "referee", arg2[1], (char *)NULL);
        perror("Failed to execute referee");
        exit(EXIT_FAILURE);
    }

    int status;
    waitpid(pid, &status, 0);  // Wait for referee to finish
    // if (WIFEXITED(status)) {
    //     printf("Game completed successfully.\n");
    // }

    return EXIT_SUCCESS;
}
```

Listing 1: play.c

5.2 Referee Program

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 4444

/**
 * Sends a message to specified socket.
 *
 * @param socket socket to send message to.
 * @param message message to send.
 */
void send_player_messages(int socket, char *message) {
    char formatted_message[1024];
    snprintf(formatted_message, sizeof(formatted_message), "%s\n", message);
    if (send(socket, formatted_message, strlen(formatted_message), 0) < 0) {
        perror("send failed");
        exit(EXIT_FAILURE);
    }
}

/**
 * main function of referee program.
 *
 * @param arg1  number of command-line arguments.
 * @param arg2  array of command-line arguments.
 * @return  exit status of program.
 */
int main(int arg1, char *arg2[]) {
    if (arg1 != 2) {
        fprintf(stderr, "Usage: %s <number_of_rounds>\n", arg2[0]);
        exit(EXIT_FAILURE);
    }

    int rounds = atoi(arg2[1]);
    int server_fd, player_needs_sock[2];
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char choices[2][10];  // store choices from both players
    int scores[2] = {0, 0};  // Score player 1 and player 2

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 2) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < 2; i++) {
        player_needs_sock[i] = accept(server_fd, (struct sockaddr *)&address, (
    ↪ socklen_t*)&addrlen);
```

```
        if (player_needs_sock[i] < 0) {
            perror("accept");
            exit(EXIT_FAILURE);
        }
        printf("          Player %d: Ready\n", i + 1);
        send_player_messages(player_needs_sock[i], "READY");
    }

    for (int round = 0; round < rounds; round++) {
        printf("Go Players [%d]:\n", round + 1);
        for (int j = 0; j < 2; j++) {
            send_player_messages(player_needs_sock[j], "GO");
            memset(choices[j], 0, sizeof(choices[j]));
            if (recv(player_needs_sock[j], choices[j], sizeof(choices[j]), 0) < 0) {
                perror("recv failed");
                exit(EXIT_FAILURE);
            }
            printf("          Player %d: %s\n", j + 1, choices[j]);
        }

        int result = 0; // 0 = Draw, 1 = player 1 Wins, 2 = player 2 Wins
        if (strcmp(choices[0], choices[1]) == 0) {
            printf("          Players Draw\n");
        } else if ((strcmp(choices[0], "Rock") == 0 && strcmp(choices[1], "Scissors")
↪  == 0) ||
                   (strcmp(choices[0], "Scissors") == 0 && strcmp(choices[1], "Paper"
↪ ) == 0) ||
                   (strcmp(choices[0], "Paper") == 0 && strcmp(choices[1], "Rock") ==
↪  0)) {
            printf("          Player 1 Wins\n");
            scores[0]++;
            result = 1;
        } else {
            printf("          Player 2 Wins\n");
            scores[1]++;
            result = 2;
        }

        if (result == 0) {
            send_player_messages(player_needs_sock[0], "Draw");
            send_player_messages(player_needs_sock[1], "Draw");
        } else if (result == 1) {
            send_player_messages(player_needs_sock[0], "Win");
            send_player_messages(player_needs_sock[1], "Lose");
        } else {
            send_player_messages(player_needs_sock[0], "Lose");
            send_player_messages(player_needs_sock[1], "Win");
        }
    }

    printf("Final Score: \n          Player 1: %d \n          Player 2: %d\n", scores
↪ [0], scores[1]);
    if (scores[0] > scores[1]) {
        printf("Winner is Player 1!\n");
    } else if (scores[1] > scores[0]) {
        printf("Winner is Player 2!\n");
    } else {
        printf("Players Draw\n");
    }

    for (int i = 0; i < 2; i++) {
        send_player_messages(player_needs_sock[i], "STOP");
        close(player_needs_sock[i]);
    }

    // printf("Game completed, server shutting down.\n");
```

```
    close(server_fd);
    return 0;
}
```

Listing 2: referee.c

5.3 Player Program

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

#define PORT 4444
#define SERVER_IP "127.0.0.1"

/**
 * Displays error message and exits program.
 *
 * @param messages error message to display.
 */
void error(const char *messages) {
    perror(messages);
    exit(1);
}

/**
 * Generates random move from options "Rock", "Paper", and "Scissors".
 *
 * @return pointer to move.
 */
const char* getRandomMove() {
    const char *moves[3] = {"Rock", "Paper", "Scissors"};
    return moves[rand() % 3];
}

int read_some_line(int sockfd, char *buffer, int maxLen) {
    char *ptr = buffer;
    char read_char;
    int n;

    while ((n = read(sockfd, &read_char, 1)) > 0) {
        if (read_char == '\n') break; // like breakdancer
        if ((ptr - buffer) < maxLen - 1) *ptr++ = read_char;
    }
    *ptr = 0; // strings need to be socially distanced
    return n <= 0 ? -1 : strlen(buffer); //  life is full of ups and downs
}

int main(int arg1, char *arg2[]) {
    int sockfd;
    struct sockaddr_in serv_addr;
    char buffer[256];

    srand(time(NULL));  // randomness is spice of life

    // make socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");

    // we need to know where party is
    serv_addr.sin_family = AF_INET;
```

```
    serv_addr.sin_port = htons(PORT);
    serv_addr.sin_addr.s_addr = inet_addr(SERVER_IP);

    // connect to party
    if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR connecting");

    // printf("Connected successfully to referee.\n");

    // main loop, would you like to play game?
    while (1) {
        bzero(buffer, 256);
        if (read_some_line(sockfd, buffer, 255) < 0)
            error("ERROR reading from socket");

        printf("Message from referee: %s\n", buffer);

        if (strcmp(buffer, "GO") == 0) {
            const char *move = getRandomMove();  // life is unpredictable
            printf("Chose: %s\n", move);
            if (write(sockfd, move, strlen(move)) < 0)
                error("ERROR writing to socket");
        } else if (strcmp(buffer, "STOP") == 0) {
            // Terminator
            printf("Received STOP. Closing connection.\n");
            break;
        }
    }

    close(sockfd);  // close connection, because we're not animals
    return 0;
}
```

Listing 3: player.c

5.4 Readstr Function

```
/**
 * @file readstr.c
 * @brief Utility function to read a string from a file descriptor until a newline
   ↪ character.
 * @author Abraham Reines
 * @date Mon Apr  8 15:47:11 PDT 2024
 */

#include <unistd.h>

/**
 * Reads characters from a file descriptor into a buffer until a newline is
   ↪ encountered.
 *
 * @param fd file descriptor to read.
 * @param str buffer where string will be stored.
 *
 * Notes:
 * - function assumes ‘str‘ has space to store read data.
 * - string stored in ‘str‘ will be terminated.
 * - If a newline is read, it is not included in stored string.
 * - function stops reading if a newline is encountered or an error happens.
 */
void readstr(int fd, char *str) {
    char ch;
    ssize_t readResult;

    while (1) {
        readResult = read(fd, &ch, 1);

```

```
        if (readResult > 0) {
            if (ch == '\n') break; // Stop at newline
            *str++ = ch; // Store character and move pointer
        } else {
            // End loop if readResult is 0 or less
            break;
        }
    }

    *str = '\0'; // terminate string
}
```

Listing 4: readstr.c

5.5 MakeFile

```
CC = gcc
CFLAGS = -g -Wall
OBJS = play.o referee.o player.o readstr.o

all: play referee player

play: play.o
  $(CC) $(CFLAGS) play.o -o play

referee: referee.o readstr.o
  $(CC) $(CFLAGS) referee.o readstr.o -o referee

player: player.o readstr.o
  $(CC) $(CFLAGS) player.o readstr.o -o player

play.o: play.c
  $(CC) $(CFLAGS) -c play.c -o play.o

referee.o: referee.c
  $(CC) $(CFLAGS) -c referee.c -o referee.o

player.o: player.c
  $(CC) $(CFLAGS) -c player.c -o player.o

readstr.o: readstr.c
  $(CC) $(CFLAGS) -c readstr.c -o readstr.o

clean:
  rm -f $(OBJS) play referee player
```

Listing 5: Makefile

References

1. Robbins, K. A., & Robbins, S. (n.d.). *Unix™ Systems Programming: Communication, Concurrency, and Threads*. O'Reilly Media. Retrieved from https://www.oreilly.com
2. Toptal®. (n.d.). *Beginner's Guide to Concurrent Programming*. Retrieved from https://www.toptal.com
3. (n.d.). *Start Concurrent: A Gentle Introduction to Concurrent Programming*. Retrieved from https://start-concurrent.github.io

Academic Integrity Pledge

"This work complies with JMU honor code. I did not give or receive unauthorized help on this assignment."