Patrick Ashby
08370140
ECE 4220
Final Project
12/14/10

Abstract:

Video games have been around for about 40 years now, but online gaming has only recently risen in popularity. This project aims to implement a low level form of network gaming using only two TS-7250 boards and two computers. The project consists of creating a program that allows two users to play a simple pong-like video game on two different computers across a network.

Problem Statement and Technical Background:

The problem of this project is to allow two people to play a game of pong against each other on two different computers. The users control the paddles of the game by pressing buttons on the I/O board we used in the labs this semester. In order for the two computers to communicate with one another, they must use socket communication. There are two popular forms of socket communication: TCP and UDP. TCP stands for transmission control protocol. In this form of communication, all outgoing packets are sent along the same path ensuring that they arrive at their destination in the same order that they were sent. UDP stands for user datagram protocol. In this form of communication, outgoing packets are not necessarily received in the same order that they are sent. However, since lost or late packets are not waited for, UDP is usually a faster form of communication than TCP. Although not as fast as UDP, TCP is considered to be more reliable. Since, video games rely on fast graphics refresh rates, and low

levels of lost packets are rarely noticed, I chose to use UDP in my socket communication for this project. In order to easily transmit all of the relevant information between the two computers, I created a structure named message to send and receive through the socket. A structure can contain variables of different data types and helps organize information for easy transmission. Another programming feature used in this project is the real time task. Real time tasks allow programmers to ensure that certain parts of the code are executed at exact time intervals. In the case of this program, I used real time tasks to ensure that the screens of both computers were refreshed at a constant and nearly synchronous rate of 10 Hz. Another feature I used is a pthread. When a pthread is created, the computer will split computing time between the main and the thread. There is no practical reason that I chose to use a thread in this program other than to make sure I could use one correctly and a small layer of complexity to the project. I also used memory mapping to map the buttons of the I/O board the memory on the board.

Proposed Approach:

In order to implement this project, I created a .c file for each computer. While not exactly the same, these files are very similar to ensure that the graphics behave the same way on both computers. In both files, I define a structure named message. Message contains the positions the paddles, x and y positions of the ball, and the score of both players. The main function of both programs is the same. A pthread named thread1 is defined and created using the function pthread_create. This function creates a thread that enters a function named checkButton. The next line of code calls a function named pthread_join. This function forces the main to pause its execution until the thread has finished its execution. After the thread ends its execution, the main is finished, and the program ends. Next, we enter the checkButton

function that is associated with the thread that I discussed earlier. In both files, mmap is needed to map the I/O boards to memory. This is needed so that the buttons on the board can be used as a controller for the game. After the memory is mapped, a pointer named portbir is set to the value of 0x00 in order to configure all of the buttons as inputs.

Next, the socket has to be configured and created on both files. The file named 'Project.c' serves as the server and 'Project2.c' serves as the client. On the server side, there are to two sockaddr_in structures; one named server and one named send. Server.sin_port and send.sin_port are set to 3333 in order for the socket to use port 3333. Send.sin_addr.s_addr is set to "10.3.52.255." The 255 in the IP address allows the server to broadcast to all members. The client side of the socket contains a sockaddr_in structure named server_addr and a hostent structure named host. Host is hardcoded to the IP of the server. Server_addr.sin_port is set to 3333 to use the same port as the server. Server_addr.sin_addr is set to host.h_addr. On both the server and the client sides, socket(AF_INET,SOCK_DGRAM,0) is called to create a UDP socket. On the server side, the bind function is also called to bind the socket to port predefined port.

After the socket is created, both programs initialize the variables used in the creation of the graphics. In this program, the graphics are created by constantly updated a 2D character array and outputting the entire array to a maximized console screen. Player1 and player2 are variables representing the top y coordinate of the player paddles. The initial value for both of these variables is set to 35 as it is roughly in the center of the screen. The x coordinates of the paddles are locked at 10 and 11 for player1 and 199 and 200 for player2. BallX and ballY are the x and y coordinates for the pong ball, represented on the screen by a '0.' Score1 and score2 are variables representing the score for each player. They are initialized as 48 as this is the

ASCII code for the number zero. SlopeX and slopeY are variables representing how many spaces the ball will move with each screen refresh. The value of these numbers changes depending on where on the paddle the pong ball is hit. SlopeX is initialized to 2 and slopeY is initialized to 1. BallIsRight is a variable to control the direction that the ball is moving. When this variable is one, the ball moves to the right. When the variable is 0, the ball moves to the left. The 2D character array is given dimensions of 70x211 which is roughly the size of a maximized console screen. Each member of the array is initialized to ' '. An integer array named doPrint is also created. This array contains 70 members, each corresponding to a row of the 2D character array. A value of 1 in this array means that the corresponding character row will be printed. If the member of doPrint is 0, then the corresponding character row will be skipped and '\n' will be printed in its place. This practice cuts down on the execution time of the program by skipping the rows on the screen that do not need to be printed.

After all of the variables are initialized, the server calls a receive command and the client calls a send command. This ensures that neither program will continue until a socket connection is established a message is sent across the socket. A real time task is then created in both of the files. The task is then made periodic with a period of a tenth of a second. After the task is created, both programs enter a while loop that executes until a player scores five times and wins the game. The first thing done in the while loop is the positions of the 2D array that contained the paddles and the ball are cleared. Next, portb is checked to see if the up or down buttons are pushed on the I/O board. If the up button is pressed, the value of the program's corresponding player is decremented, moving the paddle up on the screen. If the down button is pressed, the value of the player is incremented, moving the paddle down on the screen. The server side

calculates the position of player1's paddle, and the client handles the position of player2's paddle.

Next, the client side calculates the next position of the ball based on the current position, slopeX, slopeY, and ballIsRight. Also, the values of these variables are updated if necessary. If the ball is at the top or bottom of the screen, slopeY is multiplied by negative one, effectively reversing the vertical direction of the ball. If the ball hits a paddle, the value of ballIsRight is switched. If the ball is at the far right or far left of the screen and misses the paddle, the score of the scoring player is incremented, the ball is moved to the center of the screen and the direction of the ball is changed. The server side does not calculate the position of the ball as this would be redundant and a waste of computing time. After the ball and paddle positions have been calculated, the client sends the values to the server, and the server sends the value of player one's paddle to the client. The 2D character arrays are updated, and the old output on the console screens are cleared. The lines containing non whitespace characters are then printed to the console screens and the real time tasks wait until the next period.

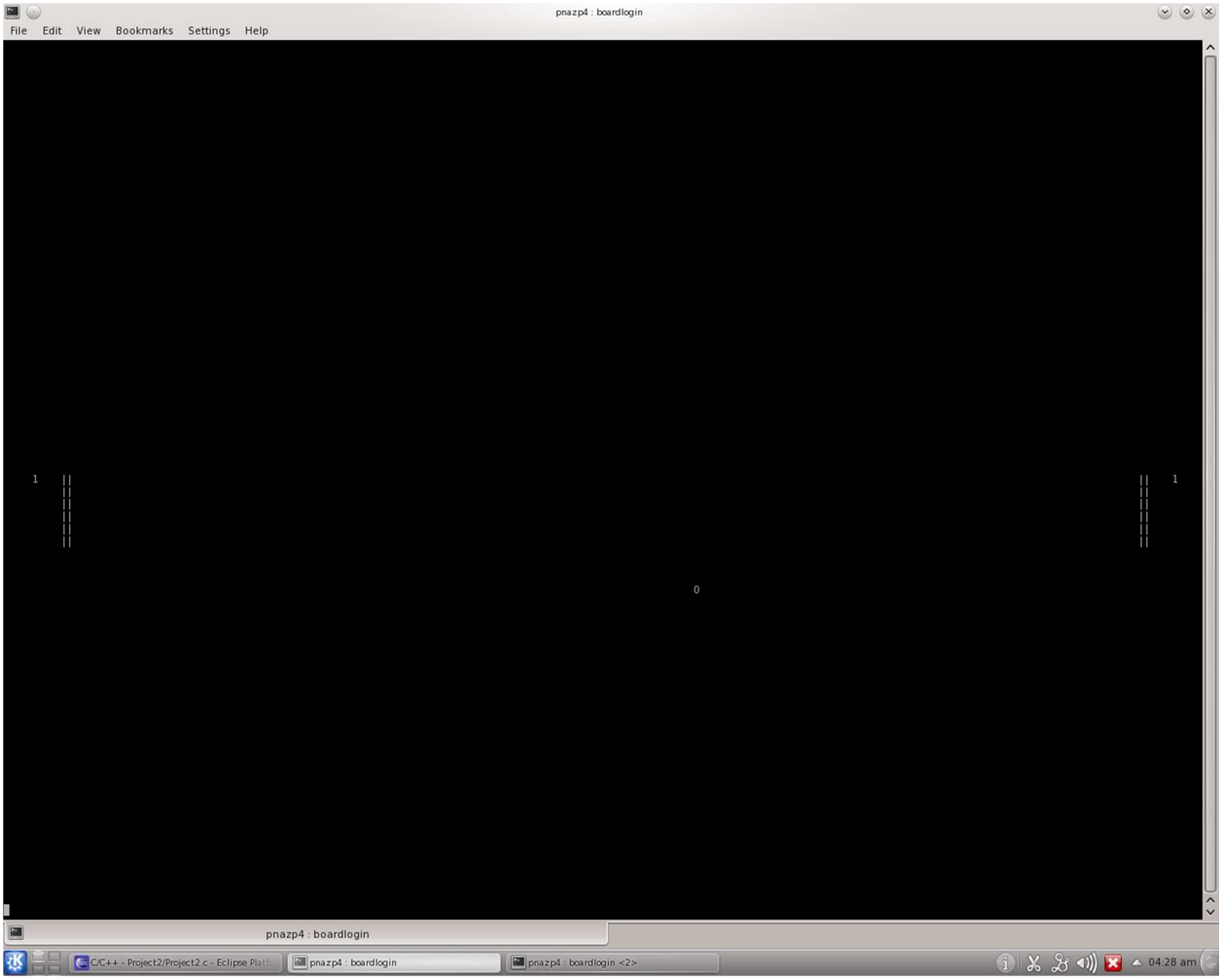
After a player has scored 5 times, the while loop ends, and the name of the player is printed on the screen. The thread ends its execution and joins back with the main.

Results:

The biggest problem I had with creating this project was getting the 2D graphic array to output correctly. At first, the screen would be filled with unintelligible garbage characters. This was because I was printing the array line by line and not ending each line with a null character. After I fixed this problem, the graphics printed correctly, but the screen was sometimes slow to update and flashed often. This was because I was printing every character of the 2D array during

each iteration of the loop. Once I started printing only the lines that contained non whitespace characters, the output became much more fluid.

I tested the final version of my programs by playing 30 games of pong across the network. The games behaved exactly like many traditional games of pong do, and at no point of the testing did I notice any discernable bugs. Below is a screen shot of the output on the console, and attached with this report is a video showing the game in action. In the video, only one paddle is moving because I could only find one I/O board in the laboratory when I made the video.



Conclusion:

From this project, I learned that socket programming is a viable way to facilitate online gaming. However, I am not sure how well this board would perform under the demands of a more complex game that did not use text based graphics. Since I had some latency issues using print statements to output my graphics in the early version of the program, I can predict that the TS-7250 boards might have problems with more demanding programs.