

Assignment 1

Sayali Kudale

Documentation:

OnlineTicTacToe Multiplayer:

To start the OnlineTicTacToe Multiplayer game both the player will provide Ip address and port number. The Socket connection and synchronization is happening between OnlineTicTacToe(InetAddress addr, int port), Counterpart thread and actionPerformed methods. The detailed functionality of all three methods is explained below:

OnlineTicTacToe(InetAddress addr, int port)

This constructor is responsible for creating multithreaded client and server design.

1. First server Socket object is created using the input port number.
2. After Creating the server socket the program wait for the client to send the connection request. This is achieved by creating the infinite while loop.
3. Whenever another machine sends a connection request using the same port number, server will accept that request. This machine will be considered as the server and plays the first by using the 'O' mark. Here, Handshake is happening between the server and client.
4. At client side once server accepted the connection request new client socket object is created using the InetAddress and port number. This player is considered as the latter player and will play the game using icon 'X'.
5. To establish the connection between the client and server machine ObjectOutputStream and ObjectInputStream objects are created by using each other's inputstream and outputstream data. This will create the read and write pipe between client and server through which further communication will happen.

The above steps will ensure communication between two remote machines. To play multiplayer game on the single player need to handle extra conditions:

1. We first get the IP address of the local machine and compare it with the input address to identify the game is played on the local machine.
2. If the game is played on the local machine, then the server socket is created and SOTimeout is not set to the interval because in same machine doesn't need to wait for accepting connection from another machine.
3. When we run another instance using localhost and same port number then the port already in use exception is thrown. This exception is caught and here we identified that both the requests are from the same machine and we can consider the new process as client and client socket is created.
4. Here, as well to create communication channel between client and server socket we use inputStream and OutputStream.

actionPerformed(ActionEvent event):

Both the players will take write action in this method. Myturn object is synchronized between action performed and counterpart thread. So that, while action is taken in the actionPerformed the counterPart thread will wait (to read from opponent input) and vice versa.

Counterpart:

This thread will keep continuously running and will wait to read the input and change the myturn value and notify to the main thread.

OnlineTicTacToe Single Player with computer:

In this mode the default constructor OnlineTicTacToe will work as a robot player against the player. OnlineTicTacToe (InetAddress addr, int port, auto) will create the JSCH connection and start a new process

which will act as a robot player and communication between the processes will be done using the inputStream and OutputStream.

Same as multiplayer CounterpartAuto will continuously keep reading the input from player and OnlineTicTacToe constructor will perform the write action. The synchronization between the constructor and counterpartAuto is managed by synchronizing the myTurn object.

filledCells ArrayList Object is used to keep track of the blocked cells in a grid, so that random number generated by the robot will be from unfilled cells.

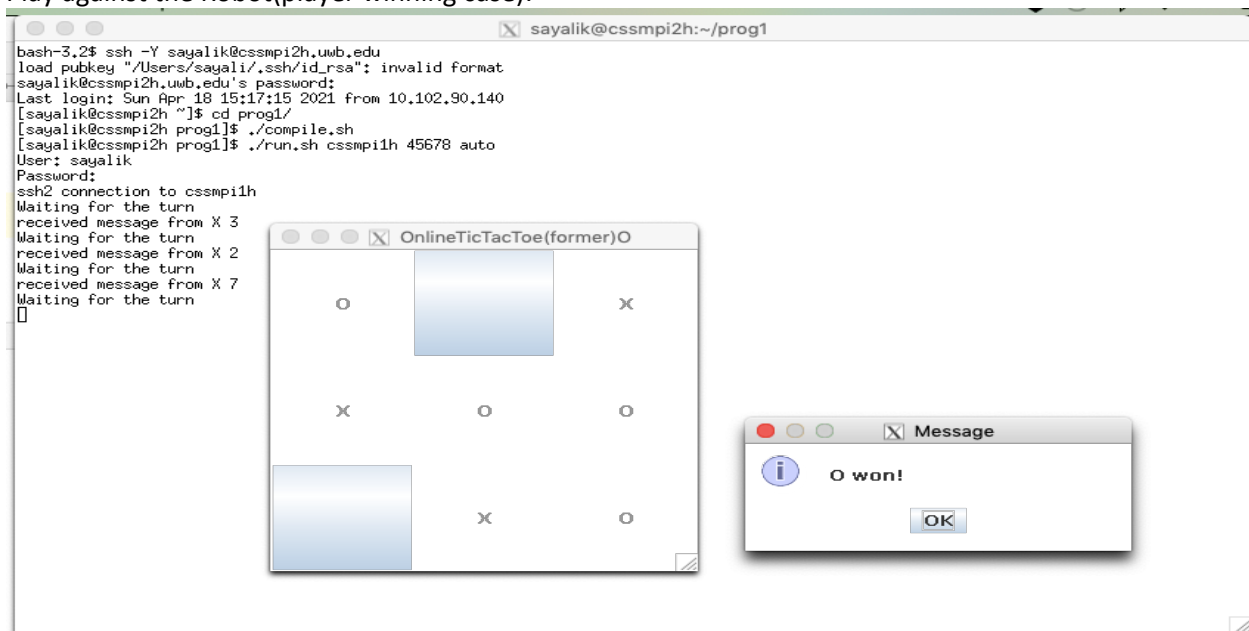
counterpartAuto thread is used instead of counterpart because there is difference in some functionalities of robot and multiplayer such as the multiplayer has to mark the window from counterpart whereas for robot does have UI window hence that action is not need. Also, for robot additional filledCells array list needs to be updated after each action to keep track of the filled cells.

Source Code:

Kindly refer the OnlineTicTacToe.java file.

Execution Output:

Play against the Robot(player winning case):

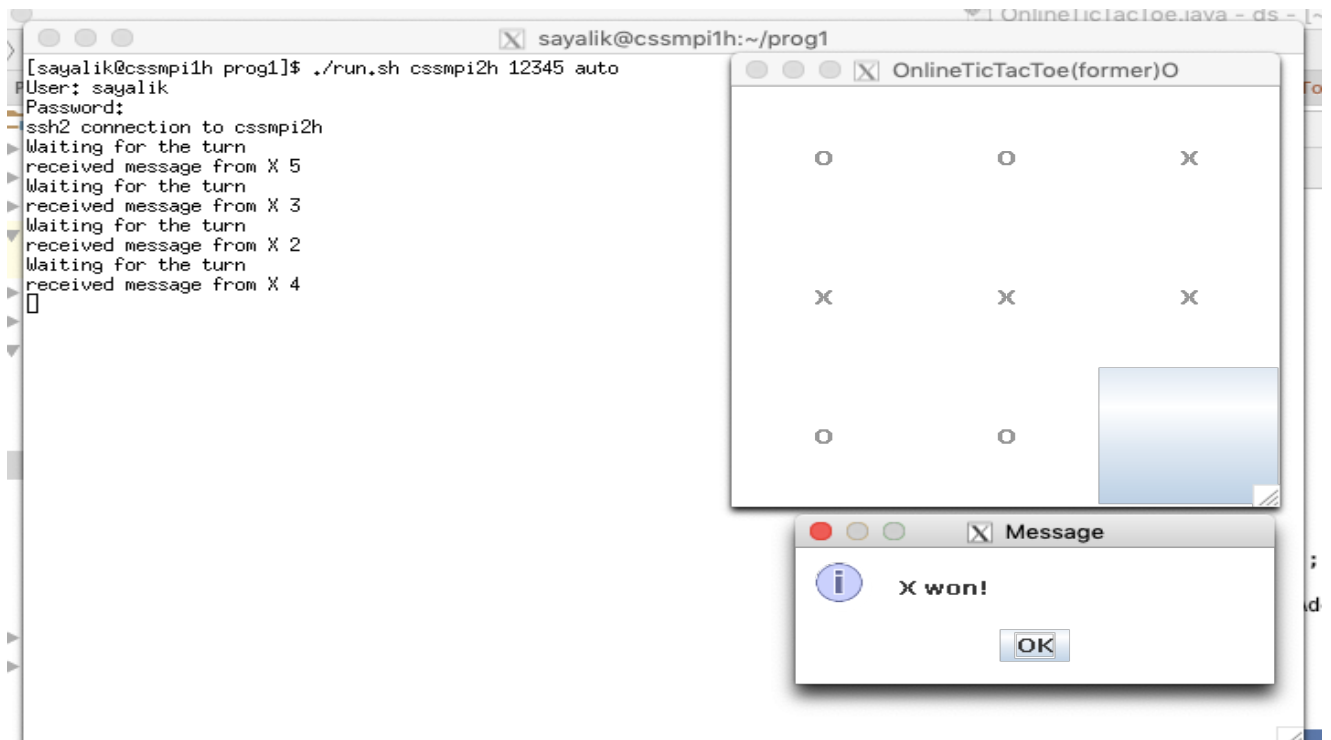


The screenshot shows a terminal window with the following output:

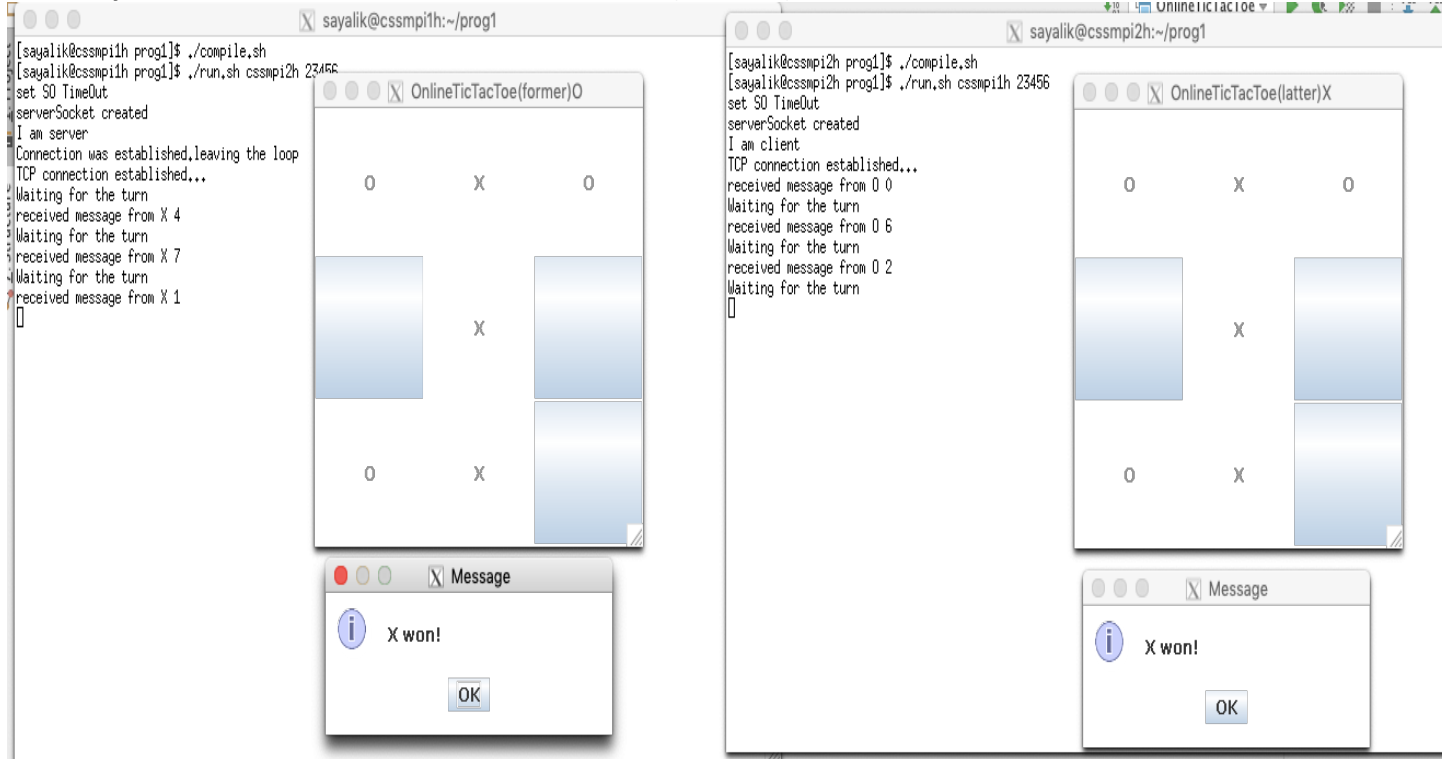
```
bash-3.2$ ssh -Y sayalik@cssmpi2h.uwb.edu
load pubkey "/Users/sayali/.ssh/id_rsa": invalid format
sayalik@cssmpi2h.uwb.edu's password:
Last login: Sun Apr 18 15:17:15 2021 from 10.102.90.140
[sayalik@cssmpi2h ~]$ cd prog1/
[sayalik@cssmpi2h prog1]$ ./compile.sh
[sayalik@cssmpi2h prog1]$ ./run.sh cssmpi1h 45678 auto
User: sayalik
Password:
ssh2 connection to cssmpi1h
Waiting for the turn
received message from X 3
Waiting for the turn
received message from X 2
Waiting for the turn
received message from X 7
Waiting for the turn
█
```

Overlaid on the terminal is a graphical window titled "OnlineTicTacToe(former)O" showing a 3x3 grid. The grid contains 'O' and 'X' characters, with some cells highlighted in blue. To the right of the terminal is a small "Message" dialog box with an information icon and the text "O won!" and an "OK" button.

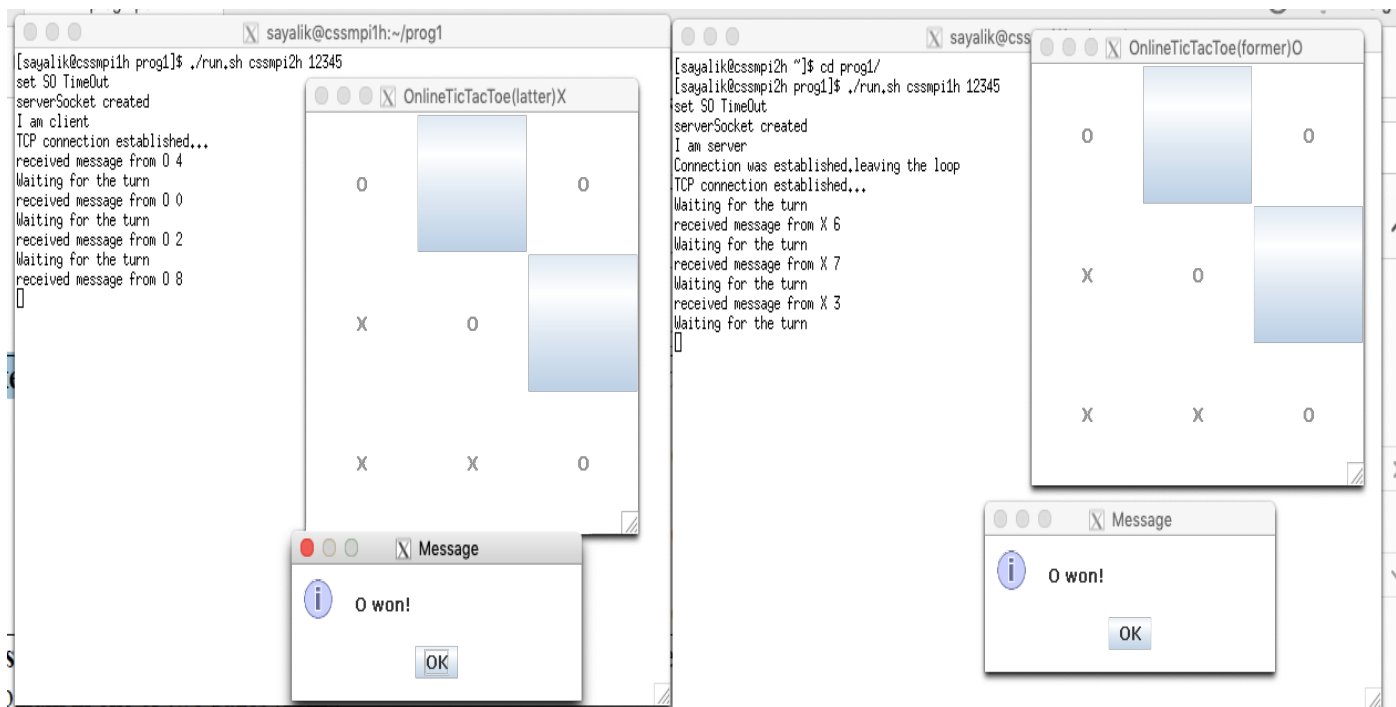
Play against the Robot(Robot winning case):



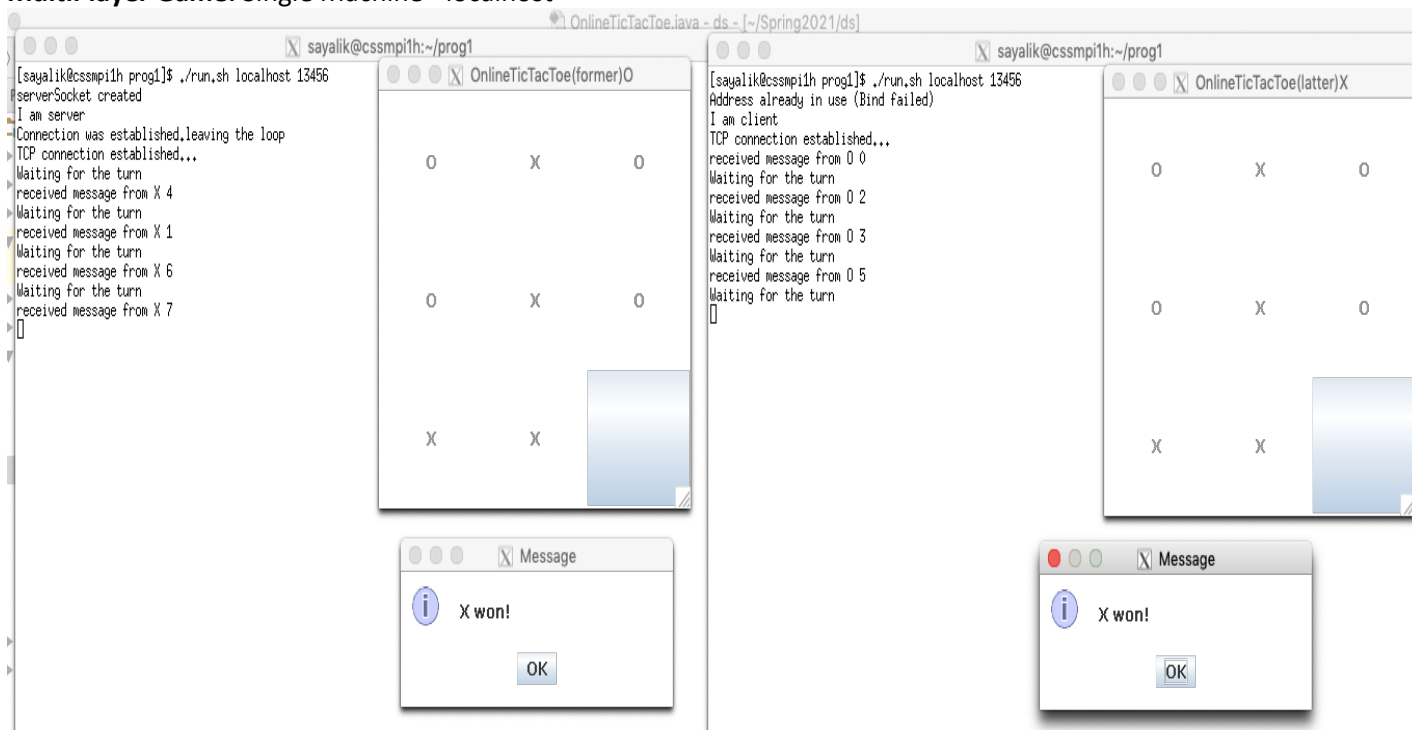
MultiPlayer Game: On two different Remote Machines (X winner)



MultiPlayer Game: On two different Remote Machines (O winner)



MultiPlayer Game: Single Machine- localhost



Discussions:

The program 1 was great hands-on socket Programming and Multithreading, barrier synchronization.

Barrier synchronization:

As we learned in LAB1, barrier thread synchronization. Where it enables multiple threads to wait until all threads have reached a particular point of execution or barrier before any thread continues. In our case suppose we have 3 threads for 2 iteration, then until all the threads (3 in this case) comes to iteration 1 no thread process further.

```
Thread1_itr1
Thread2_itr1
Thread3_itr1 ..
then
Thread1_itr2
Thread2_itr2
Thread3_itr2...
```

Once we reach at the end we notifyAll that count has been reach and start next iteration for same thread set (3 thread in our case)

This is very useful to manage multiple threads execution in parallel programming

Server Non-Blocking

- In this program, we have implemented non-blocking accept(), and understood how important it is to have non-blocking socket. The SOTimeout is used to set servers as non-blocking.
- In Blocking for example if we call connect(), program doesn't regain control until either the connection is made, or an error occurs, this process of waiting is referred as blocking. Whereas in non-blocking mode, you never wait for an operation to complete.
- In Non-blocking sockets when we call accept() and there isn't already a client connecting to you, it will return a value "Operation Would Block" to tell you that it can't complete the accept() without waiting.
- We have created the client-server architecture using this concept.
- Communication between client and server processes is happening with the socket inputStream and outputStream features.
- Both the processes send the messages to the counterpart and then wait for reading the counterpart's messages.

JSCH Connection:

JSCH library is provided by java which can be used to launch the processes at the remote machine.

In this program we are launching the remote process of onlineTicTacToe which is behaving as the robot player in case of single player game.

Improvements in the program:

1. Free the resources after use, for example socket, connection etc.
2. Exception handling can be improved to avoid various below errors/Exceptions:
 - a. the pipe broken error : this error may cause because of the either sockets ends may not be in sync.
 - b. too many connections error
 - c. Connection reset error
 - d. Connection refused error

- e. Port already in use error in case of remote multiplayer
 - f. SocketException: this exception is thrown while reading or writing to the close socket which can be because of the slow network or firewall issues
 - g. Maximum Connections reached exception
3. Close the program as well as kill all the running threads upon completion.
 4. Give the timeout message and close the program if action is not taken for specified time.

Lab Sessions:

Please refer the lab code files in lab1, lab2 and lab2 folders.

Lab 1 output:

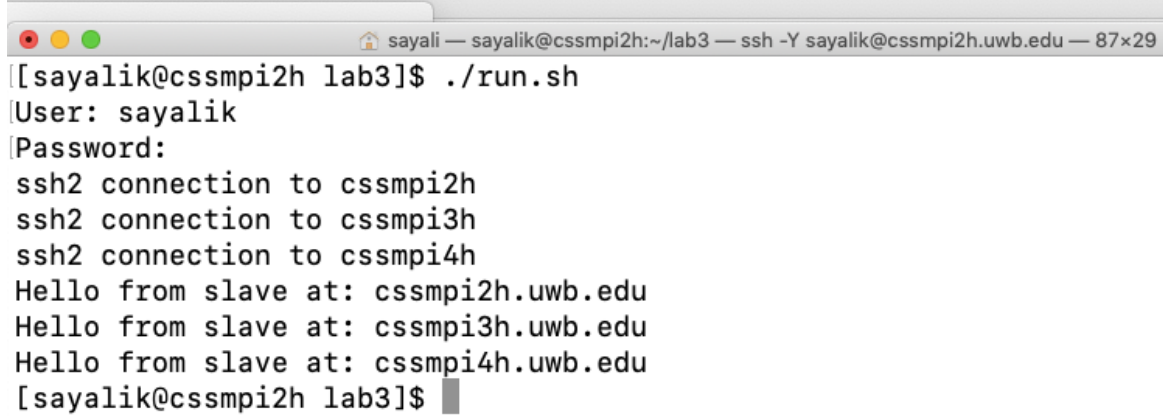
```
sayali — sayalik@cssmpi1h:~/lab1 — ssh sayalik@cssmpi1h.uwb.edu — 86x28
~ — -bash
7 barriers completed by Thread[main,5,main]
7 barriers completed by Thread[Thread-1,5,main]
8 barriers completed by Thread[Thread-1,5,main]
8 barriers completed by Thread[main,5,main]
8 barriers completed by Thread[Thread-0,5,main]
9 barriers completed by Thread[Thread-1,5,main]
9 barriers completed by Thread[main,5,main]
9 barriers completed by Thread[Thread-0,5,main]
[sayalik@cssmpi1h lab1]$ java BarrierThread 3 6
0 barriers completed by Thread[main,5,main]
0 barriers completed by Thread[Thread-1,5,main]
0 barriers completed by Thread[Thread-0,5,main]
1 barriers completed by Thread[Thread-0,5,main]
1 barriers completed by Thread[main,5,main]
1 barriers completed by Thread[Thread-1,5,main]
2 barriers completed by Thread[Thread-1,5,main]
2 barriers completed by Thread[Thread-0,5,main]
2 barriers completed by Thread[main,5,main]
3 barriers completed by Thread[Thread-1,5,main]
3 barriers completed by Thread[main,5,main]
3 barriers completed by Thread[Thread-0,5,main]
4 barriers completed by Thread[Thread-0,5,main]
4 barriers completed by Thread[Thread-1,5,main]
4 barriers completed by Thread[main,5,main]
5 barriers completed by Thread[main,5,main]
5 barriers completed by Thread[Thread-0,5,main]
5 barriers completed by Thread[Thread-1,5,main]
[sayalik@cssmpi1h lab1]$
```

Lab2 output:

```
sayali — sayalik@cssmpi1h:~/lab2 — ssh sayalik@cssmpi1h.uwb.edu — 78x26
~ — -bash
[sayalik@cssmpi1h lab2]$ java P2P
Usage: java P2P ipAddr port message
[sayalik@cssmpi1h lab2]$ java P2P cssmpi2h 12345 "Hii from cssmpi1h"
TCP connection established...
Hello from cssmpi2h from cssmpi2h.uwb.edu
[sayalik@cssmpi1h lab2]$

sayali — sayalik@cssmpi2h:~/lab2 — ssh -Y sayalik@cssmpi2h.uwb.edu — 87x29
Last login: Sun Apr 18 22:49:36 on ttys001
Abhijeets-MacBook-Air:~ sayali$ ssh -Y sayalik@cssmpi2h.uwb.edu
load pubkey "/Users/sayali/.ssh/id_rsa": invalid format
[sayalik@cssmpi2h.uwb.edu's password:
Last login: Sun Apr 18 22:10:55 2021 from 10.102.89.27
[sayalik@cssmpi2h ~]$ cd lab2
[sayalik@cssmpi2h lab2]$ ls
P2P.class P2P.java P2P.java~
[sayalik@cssmpi2h lab2]$ P2P cssmpi1h Hello from cssmpi2h
-bash: P2P: command not found
[sayalik@cssmpi2h lab2]$ java P2P cssmpi1h "Hello from cssmpi2h"
Usage: java P2P ipAddr port message
[sayalik@cssmpi2h lab2]$ java P2P cssmpi1h 12345 "Hello from cssmpi2h"
TCP connection established...
Hii from cssmpi1h from cssmpi1h.uwb.edu
[sayalik@cssmpi2h lab2]$
```

Lab3 output:

A screenshot of a terminal window. The title bar shows a home icon, the name 'sayali', and the command 'sayalik@cssmpi2h:~/lab3 — ssh -Y sayalik@cssmpi2h.uwb.edu — 87x29'. The terminal content shows a user running a script and receiving output from three slave nodes.

```
[sayalik@cssmpi2h lab3]$ ./run.sh
User: sayalik
Password:
ssh2 connection to cssmpi2h
ssh2 connection to cssmpi3h
ssh2 connection to cssmpi4h
Hello from slave at: cssmpi2h.uwb.edu
Hello from slave at: cssmpi3h.uwb.edu
Hello from slave at: cssmpi4h.uwb.edu
[sayalik@cssmpi2h lab3]$
```