Fishnet Assignment 1: Hello World

Due: Monday Oct. 8, 2001 at the beginning of class. Out: Monday Oct 1, 2001. CSE/EE461 Autumn 2001; Wetherall.

In this assignment, you will develop a single C program that is a Fishnet node. You must do this on your own; only the later assignments are done in pairs. The node will connect to other Fishnet nodes and allow you to send simple messages that you type on the keyboard, such as “Hello World”, to those other nodes. The goal of this first assignment is simply to introduce you to the Fishnet development environment and the Fishnet components that you must interact with throughout the course.

1 Introduction to the Fishnet

Fishnet is a class-sized network that we will build over the course of the quarter! You will build a Fishnet node progressively, working in pairs except for this assignment, over a series of five assignments. Once we are done, nodes from each team will be able to communicate with one another in a shared network.

1.1 Development Environment

We will be developing under Linux and writing C code. Typically, you will go the the CSE Labs, log into a Windows machine, and from there remotely access a Linux server using teraterm (ssh) or X windows. All programming, compiling and running of the Fishnet will take place on the Linux machines. Each node of the Fishnet network will run as a process on a Linux machine. The compiler we will use is gcc, and the debugger we will use is gdb. You can use any editor you prefer, such as emacs. Sorry, but you must use EXACTLY the tools and style we ask for your assignments so that we can manipulate your programs automatically for grading checks.

1.2 What is a Fishnet?

The term “Fishnet” is somewhat overloaded, and before going further we want to clarify what it means. First, it is the code that makes up all of the assignments. This is the *Fishnet development environment*. Second, it is a running network of many nodes and a single fishhead (a key component to be described next). This is what we most commonly mean by a *Fishnet*. Note, however, that there are many potential Fishnets. You will all start your own, private, Fishnet to develop and test the code required for this assignment. Each of you will run your own fishhead, and nodes of your network will not interact with nodes of other students’ networks. Finally, there is The Fishnet, with a capital “T”. This is the one public, shared Fishnet (a running network) we will create over the quarter that everyone’s node is able to join and participate in a collective class network. This is accomplished simply by pointing your code at a fishhead that we run. You do not need to use this shared Fishnet for this assignment.

1.3 Fishnet Components

The Fishnet project code, like everything else you need, is available from the course web site. The package we provide you with contains the following key components:

1.3.1 fishhead

fishhead is a program that manages Fishnet nodes. A network can contain many nodes, but only one fishhead. The main function of the fishhead is to tell individual nodes who their neighbors are. It is important to understand that the fishhead manages the topology and decides who is connected to whom, not you in your programs. (For the curious, the nodes of networks you operate are run as separate processes that communicate with each other using a UDP overlay.) When you develop a Fishnet node (this programming assignment) and run it, one of the first things it will do is to join a Fishnet network by contacting the fishhead. This means that before you start one of your Fishnet nodes there must be a fishhead process running. You only need to start a fishhead for your network once, even though nodes of the network can come and go.

1.3.2 libfish.a

The fish library implements all of the Fishnet functionality that you will need for your assignments. When you send a message using the fish_sendpacket() function, for example, libfish.a is called to do the work of sending the packet. libfish.a also prints a large (but controllable) amount of debugging information to the console (stderr) to help you understand what is going on with your program. The library source code is available in the fishsrc directory so that you can see how the Fishnet really works and to help with your debugging, but you MUST NOT change this code at all so that you remain compatible with other people's nodes.

1.3.3 fish.h

This is the header file that you should include in your C program to gain access to the functionality implemented in the fish library. It contains the dozen or so Fishnet API functions that you can call, as well as the structures that define packet formats, and other Fishnet constants. You should read the comments in this file, as it is the definitive Fishnet API documentation rather than a separate manual.

2 What You Need To Do

Write a C program in a single file called hw1.c that, when run, does the following:

- Takes three whitespace separated command line arguments: first, a hostname on which you have started a fishhead running; second, the port number on which you started the fishhead; and last the address you want for your node in the Fishnet, which is simply a small number.

- Joins the Fishnet described by the command line arguments with the specified address. You will need to use the fish_joinnetwork() call, and you will need to have started your own fishhead before running your program. Remember, you choose the address of your node, but the fishhead decides what other nodes you are connected to, and your neighbors change as other nodes come and go.

- Waits to get a line of input from the keyboard and sends it to all of the neighboring nodes. You will need to use fish_main(), fish_keybhook(), fish_getneighbors() and fish_sendframe().When you get a line of input you must print out "`Got input: <input>`" where `<input>` stands for the line you received and the line ends with a newline. Please follow print instructions exactly, otherwise automatic grading checks may fail. Once you have the line of input and have printed it out, send it to all of the node neighbors. You should ignore the packet structures in fish.h to do this, and simply send the string directly. A key concept here is that of upcalls. Rather than directly call a function like, for example, getlineoftext(), that waits for a line of text, you register a function (using fish_keybhook()) ahead of time that is called by libfish.a when a line of keyboard input is ready. Then you call a main loop function (fish_main) and wait for the callback.

- Waits to receive a packet for a neighboring node and then prints it out. You will need to use fish_recvhook(), which registers an upcall. When you print out the packet, use the format "`Got packet: <packet>`" where `<packet>` stands for the contents of the packet you received and the line ends with a newline. Your program should repeat getting input, sending, receiving and printing indefinitely. (When you enter ^D (end of file) libfish.a will automatically end the program.)

- You must write only one program in the single file hw1.c, rather than separate programs for sending and receiving, and your program must be capable of getting input and sending, and receiving and printing in any order. The latter will happen without any effort on your part if you are using upcalls in the right way.

In later assignments you will work in pairs (start looking for a partner!) but we want everyone to do this assignment by themselves to get familiar with Fishnet.

3 Step By Step Instructions

Here is the entire set of steps that you should follow for the entire assignment.

0. Check that you can get into the CSE Lab and log into a Windows machine. From there, check that you can remotely access one of the Linux servers (`fiji, tahiti, sumatra,` or `ceylon`). If you are a non-CSE major, you must fill out request forms in order to gain access and accounts. All of your work will take place on the Linux servers. The easiest way to do this is to use teraterm to ssh to these machines; there is a shortcut to do this on most desktops. You can also use any X windows packages that you are familiar with.

1. Download the Fishnet package from the course web pages into your home directory and unpack it. The easiest way to do this is to run the lynx browser (`lynx http://www.washington.edu/461`), follow the Fishnet links, and download the tarball. Alternatively you could use `wget` to download the tarball. To unpack the tarball, use `tar` with the arguments given on the web page. You should now have a fishnet directory with files in it. As a general note, you can use

the `man` command to get a help page for any unix command or function we ask you to use that you don't understand.

2. Read `fish.h`. The comments in this file tell you what the Fishnet API calls are, what they do, what arguments they take and return, and so forth. You won't find this information anywhere else.

3. Start a fishhead process to manage a Fishnet by running the fishhead program that is inside the fishnet directory. You will need to give it a magic command line argument to indicate what port it listens on for messages. Pick a number between 1024 and 32K. Your number should be different from everyone else's, otherwise fishhead will fail to start. The fishhead program runs indefinitely to keep the Fishnet it is running up. You should leave it running and create another window in which to do your development. When you're done, stop it by typing ^C.

4. Develop your program, which should be contained in a single file called hw1.c in the top level directory created when you unpacked Fishnet. The fishsrc directory contains the source code that we have written so that you may see it and use it for debugging, but you must not change it. We have supplied a Makefile so that you can type "`make hw1`" to compile your program. This will invoke the compiler, gcc, with the right command line arguments. As you write your program, note that we can see all of our solution on the screen at the same time, so if yours is more than 100 lines then you are doing something the hard way and should ask for help.

5. Test your program on a two node network by running it twice to make a network with two nodes that are connected to one another. You will need to use different command line arguments to select different node addresses. You will find it easiest to bring up two separate windows for the two nodes, as well as the window for the fishhead. If your program works, you should be able to enter text in one window and see it echoed in the other window, and vice versa, as well as see fish debugging messages.

6. Iterate on the above two steps until you think you have your solution! Remember that only feasible way to develop software of any complexity is to find a way to break the task down into simpler, checkable, subtasks. In this case, you might first write a program that had a main() function that accepted the command line arguments, printed them out and finished. When this compiles and runs properly (experience shows most of you will have found and fixed at least one error already!) move to the next task. The next task might be joining the network and sitting in fish_main(). You can use debugging messages to check if this program has worked, as you will get messages when you have joined. You can quit the program by entering ^D. Next, you might try for keyboard input, and print it out but not send a packet yet, and so forth. While this strategy might seem overkill at this stage, you should get used to it before attempting the larger assignments. The beauty of testing subtasks is that the problems are rarely where you think they are (otherwise you wouldn't have made the mistake) and testing is the only way to

pin them down. Despite this strategy, more complex bugs will require you to use the debugger.

7. Comment your program if you haven't already. Good comments don't belabor the obvious (e.g., "calling the main loop" near fish_main()). Rather, good comments tell us how you have arranged your code and assumptions you have make, as well as anything non-obvious (e.g., "this is the packet receive upcall from fish_main" near the function you installed using the receive hook). In truth, this first assignment should need very few comments, but in later assignments we will need comments from you to understand the code.

8. To prepare for turn-in, start a three node network, and when it is up type a line of input into each of the three windows, and then stop all of the nodes. Now redo this while capturing the output in each of the three node windows, beginning with and including the command you used to start each node. To do this, you should use the `script` command in each of the three windows when you first start the nodes.

9. Finally, come up with an answer to the following discussion question: What is an important advantage and an important disadvantage of the "upcall" style of programming? You should supply at most two sentences per advantage and two per disadvantage.

4 What To Turn In and Our Grading Philosophy

For this assignment, as well as for future assignments, you need to turn in both electronic and paper material as follows:

1. One and only one C file containing the source code of your solution. In this case, hw1.c. Submit this electronically using the `turnin` program on the Linux servers. You must do this before class on the day that it is due. The code you send should be suitable for us to manipulate automatically for grading checks.

2. One stapled paper writeup with your name that you bring to class on the due date that contains:

    a. A printout of any output we have asked you to capture. In this case there are three output files captured with `script`, one for each of the three nodes running the test case in step 8.
    b. A printout of the source code you submitted electronically. In this case it is hw1.c.
    c. Short answers to any discussion questions. In this case, it is your answer to the question posed in step 9.

In later assignments, where you work in pairs, we only want one copy of the C file and writeup per team.

The goal of the first assignment is to introduce you to the Fishnet. There is little networking content, so we will tend to grade with a small number of levels (read: pass, 100% if it works). For later assignments, you should know that our intent is to have you explore networking issues, rather than write a program that manages to pass a given set of acceptance tests. We will grade you on this material in two ways. First, we will look at your writeup and the C source code. The printout, your description of what is going on, your answers to the discussion questions, and the design clarity of your program, give us a very good idea of how well you got the program working and understand the issues that it explores. Second, we will sometimes compile and run your programs. We will do this cases that we are unsure of after looking at the source and your printout. We also reserve the right to randomly test programs to see that they produce the output that you claim they produce. When we test, it isn't our intent to penalize you significantly for small errors in execution, as opposed to larger errors in understanding of the concepts and design of the solution, but we will deduct some points for small errors or not following our instructions. We will also sometimes include optional material in the assignments, and hope that you choose to explore some of the issues this material raises, but we won't grade optional material.

€€€