COMP 530
Introduction to Operating Systems

HW 5
Building a Distributed Linux Shell

Kevin Jeffay
Department of Computer Science
University of North Carolina at Chapel Hill
jeffay@cs.unc.edu
October 18, 2017

http://www.cs.unc.edu/~jeffay/courses/comp530

 1

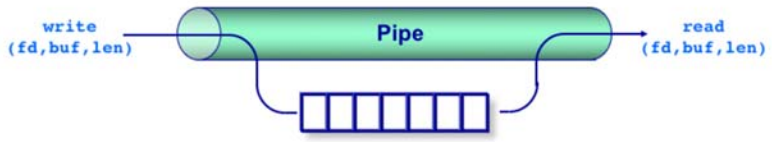
A Distributed Linux Shell
A message passing via sockets example

- Goal: Learn the socket interface and messaging over sockets
  - » Control your shell program remotely from a second machine
- Same basic structure as HW2:
  - » Write a program that reads a command line from *stdin*
    - ❖ Print a prompt ("% " — a percent sign followed by a space)
    - ❖ Read a command
  - » Create a child process to parse and execute the command
  - » Parent process just waits for the child to terminate
- But now the "child" is a process created by a server process on a remote machine

 2

Review of HW 4

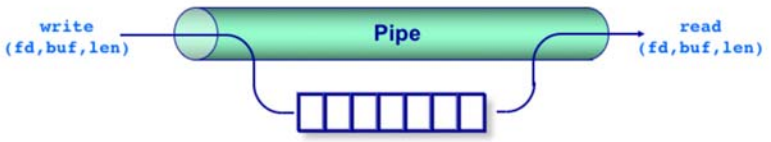
Linux Interprocess Communication (IPC)



- IPC in Linux is like message passing except more general
- The programming abstraction is a "pipe" — a shared, in-memory file
  - » A queue of 4K bytes
  - » Linux *read*/*write* systems calls used to "pass messages" between processes

Review of HW 4

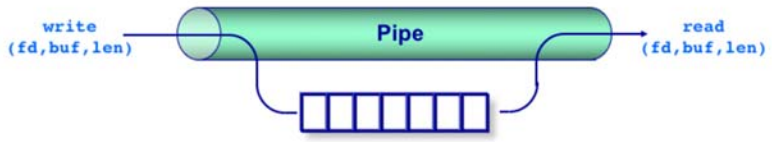
Linux Interprocess Communication (IPC)



- Pipes provide a form of buffered, asynchronous message passing
  - » Data written to one end of the pipe by one process can be read at the other end of the pipe by another process
    - ❖ A reader of the pipe blocks when the queue is empty
    - ❖ A writer to the pipe blocks when the queue is full
  - » Data is handled in a FIFO (first-in-first-out) order
  - » Pipes are unidirectional

Review of HW 4

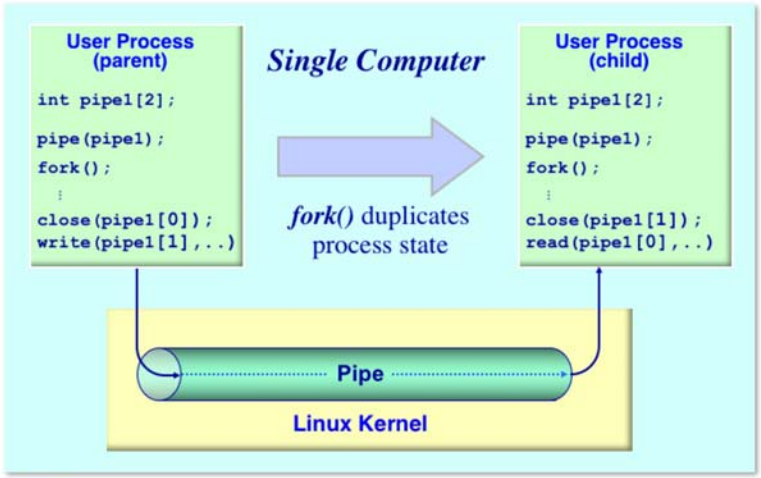
Linux Interprocess Communication (IPC)



- Typically, a process creates a pipe just before it forks one or more child processes
  - » Creating a pipe results in a process receiving a read file descriptor and a write file descriptor
- The pipe is then used for communication between the parent and child process, or between two sibling processes
  - » One process uses the read file descriptor (and ignores the write descriptor) and a second process uses the write file descriptor (and ignores the read descriptor)

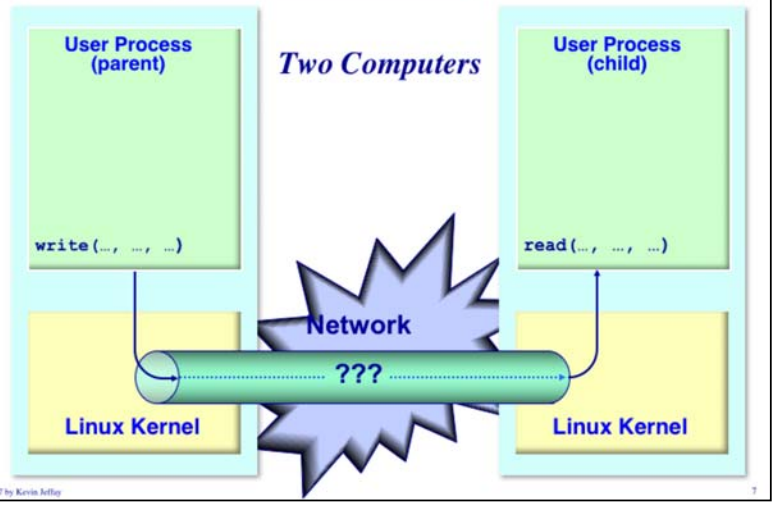
Linux Pipes

Programming details



Linux Pipes

Programming details



Client/Server Paradigm

Socket programming

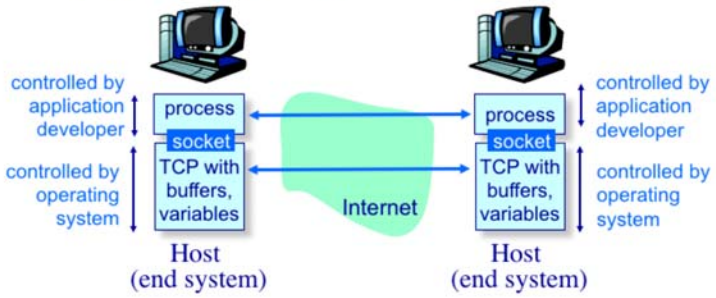
- Sockets are the fundamental building block for client/server systems
- Sockets are created and managed by applications
  - » Strong analogies with files
- Two types of transport services are available via the socket API:
  - » UDP sockets: unreliable, datagram-oriented communications
  - » TCP sockets: reliable, stream-oriented communications

socket

a *host-local*, *application created/released*, *OS-controlled* interface into which an application process can *both send and receive* messages to/from another (remote or local) application process

Client/Server Paradigm

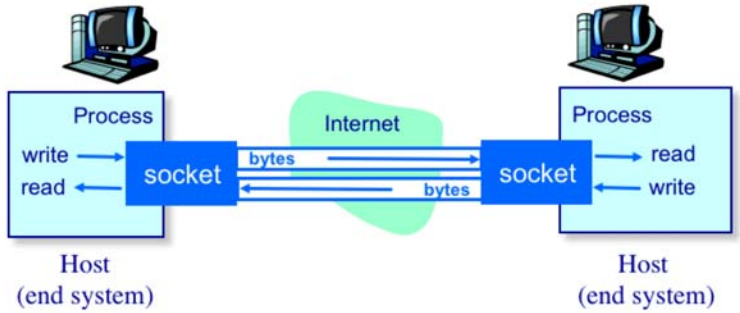
Socket-programming using TCP



- A socket is an application created, OS-controlled interface into which an application can both send and receive messages to and from another application
  - » A "door" between application processes and end-to-end transport protocols

Socket-programming using TCP

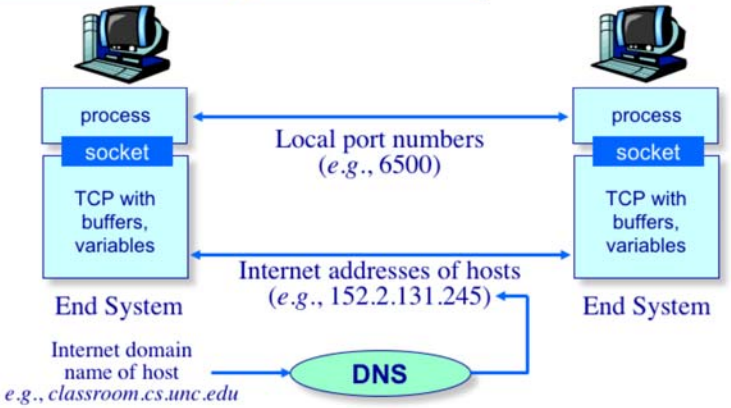
TCP socket programming model



- A TCP socket provides a reliable bi-directional communications channel from one process to another
  - » A "pair of pipes" abstraction

Socket-programming using TCP

Network addressing for sockets

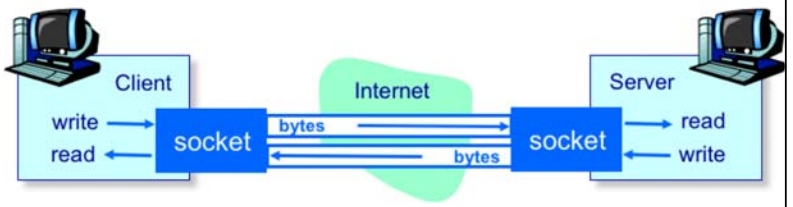


- Sockets are addressed using an IP address and port number

 11

Socket-programming using TCP

Socket programming

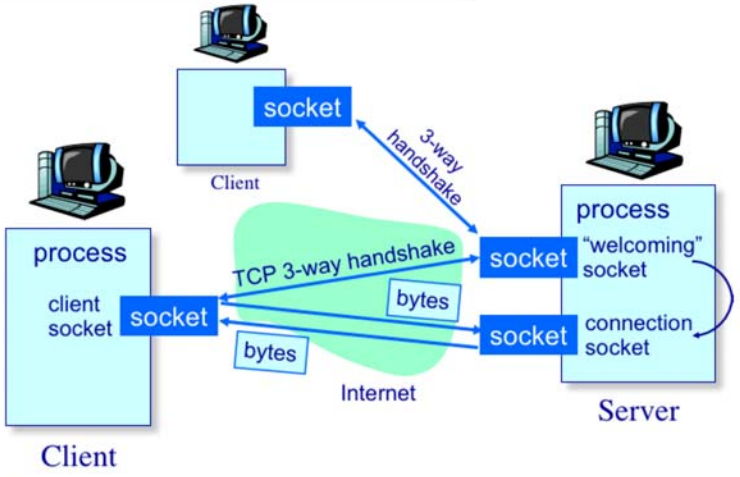


- Client creates a local TCP socket specifying the host and port number of server process
  - » Host names resolved to IP addresses using DNS
- Client contacts server
  - » Server process must be running
  - » Server must have created socket that "welcomes" client's contact
- When the client creates a socket, the client's TCP establishes connection to server's TCP
- When contacted by a client, server creates a new socket for server process to communicate with client
  - » This allows the server to talk with multiple clients

 12

Socket-programming using TCP

Socket creation in the client-server model



 13

Socket-programming using TCP

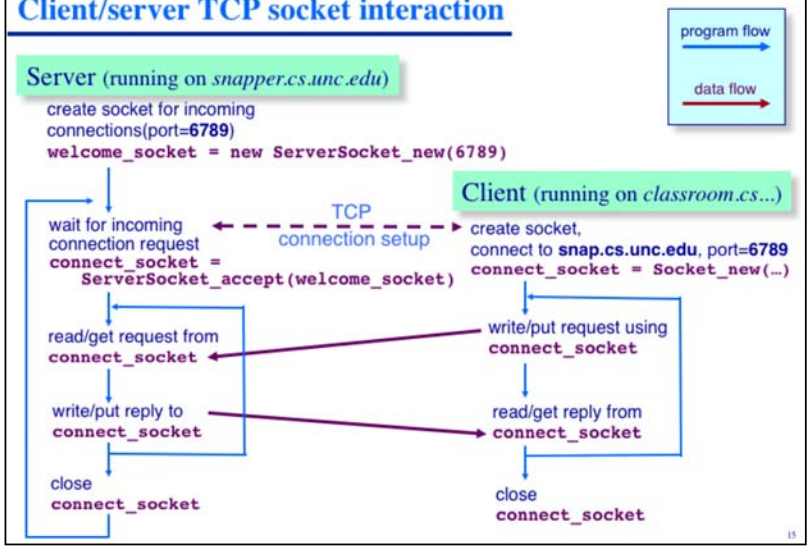
Simple client-server example

- The client reads a line of text from standard input and sends the text to the server via a socket
- The server receives the line of text from the client and converts the line of characters to all uppercase
- The server sends the converted line back to the client
- The client receives the converted text and writes it to standard output

 14

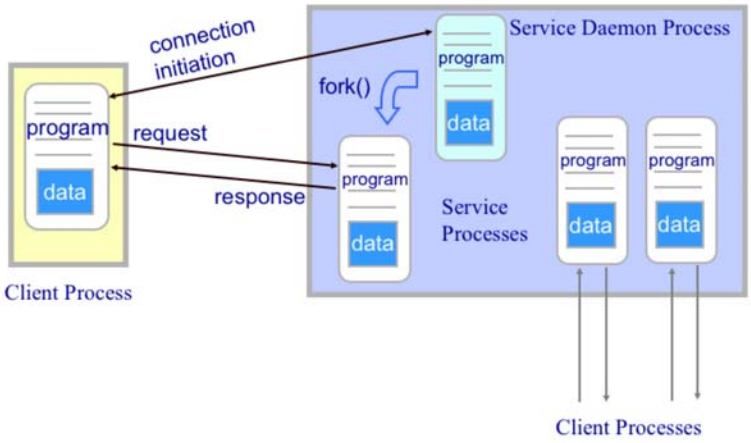
Socket programming with TCP Example

Client/server TCP socket interaction



Socket-programming using TCP

"Daemon" service model



Socket-programming using TCP

Using the Daemon service model