CS420: Operating Systems

Client/Server Communication

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania

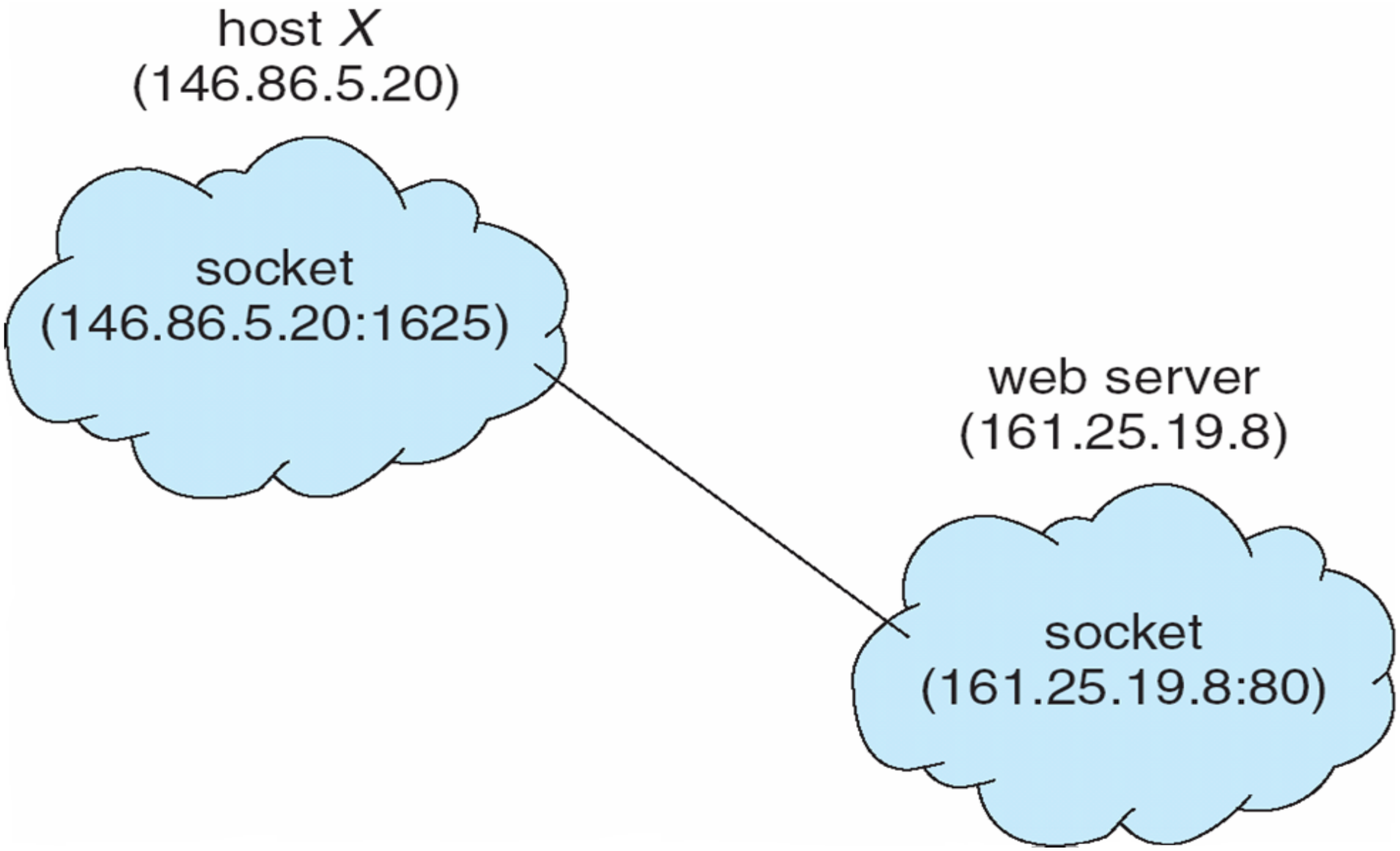
Communications in Client-Server Systems

- **Three additional strategies for communicating between processes (local and/or remote processes)**
  - Sockets
  - Remote Procedure Calls
  - Ordinary Pipes and Name Pipes

Sockets

- **A socket is defined as an endpoint for communication**
- **A socket is a concatenation of IP address and a port**
- **The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8**
- **Communication links exist between a pair of sockets**
  - All connections between sockets must be unique

Socket Communication



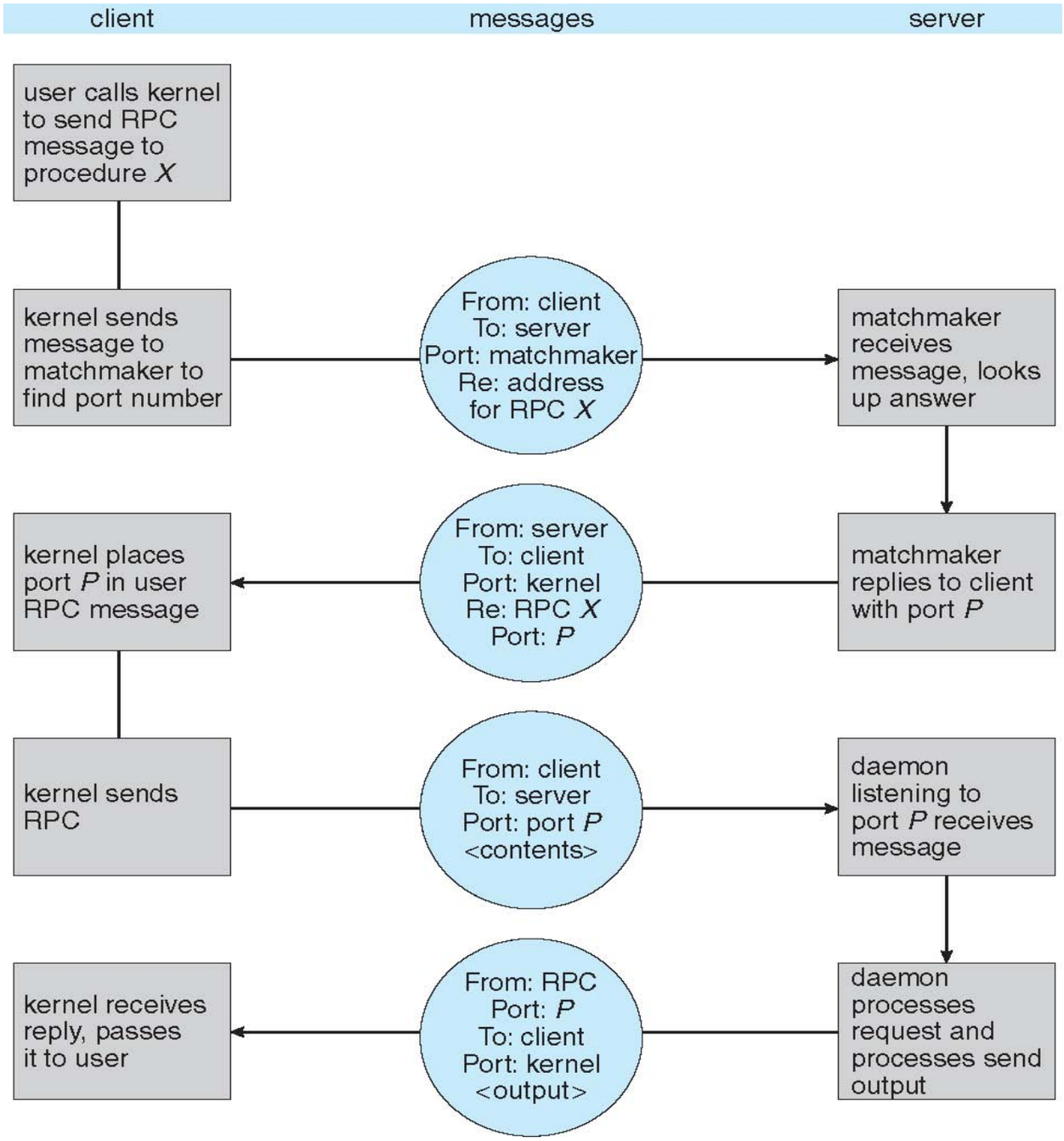
A Simple Java-based Server

```
import java.net.*;
import java.io.*;

public class DateServer {
    public static void main(String[] args) {
        try {
            ServerSocket sock = new ServerSocket(6013);

            while (true) {  /* now listen for connections */
                Socket client = sock.accept();
                PrintWriter pout = new PrintWriter(client.getOutputStream(), true);

                /* write the Date to the socket */
                pout.println(new java.util.Date().toString());

                /* close the socket and resume listening for connections */
                client.close();
            }
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

Remote Procedure Calls

- **Remote procedure call (RPC) abstracts procedure calls between processes on networked systems**
  - Allows a client to invoke a procedure on a remote host the same as it would locally
- **Stubs – client-side proxy for the actual procedure on the server**
  - Typically, a separate stub exists for each unique remote procedure
- **The client-side stub locates the server and marshalls the parameters**
- **The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server**

Execution of RPC



Pipes

- **Pipes serve as a conduit allowing two processes to communicate**

- **Implementation considerations:**
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e. parent-child) between the communicating processes?
  - Can the pipes be used over a network or only on local machine?

- **Two common types of pipes, ordinary pipes and named pipes**

Ordinary Pipes

- **Ordinary pipes allow communication in standard producer-consumer style**
- **Ordinary pipes are unidirectional**
- **Opened and treated similarly to a file**
  - Producer writes to one end (the write-end of the pipe)
  - Consumer reads from the other end (the read-end of the pipe)
- **Ordinary pipe cannot be accessed from outside the process that creates it**
  - Children inherit all of parents open files when created
  - Relationship created with fork allows parent-child communication through pipe

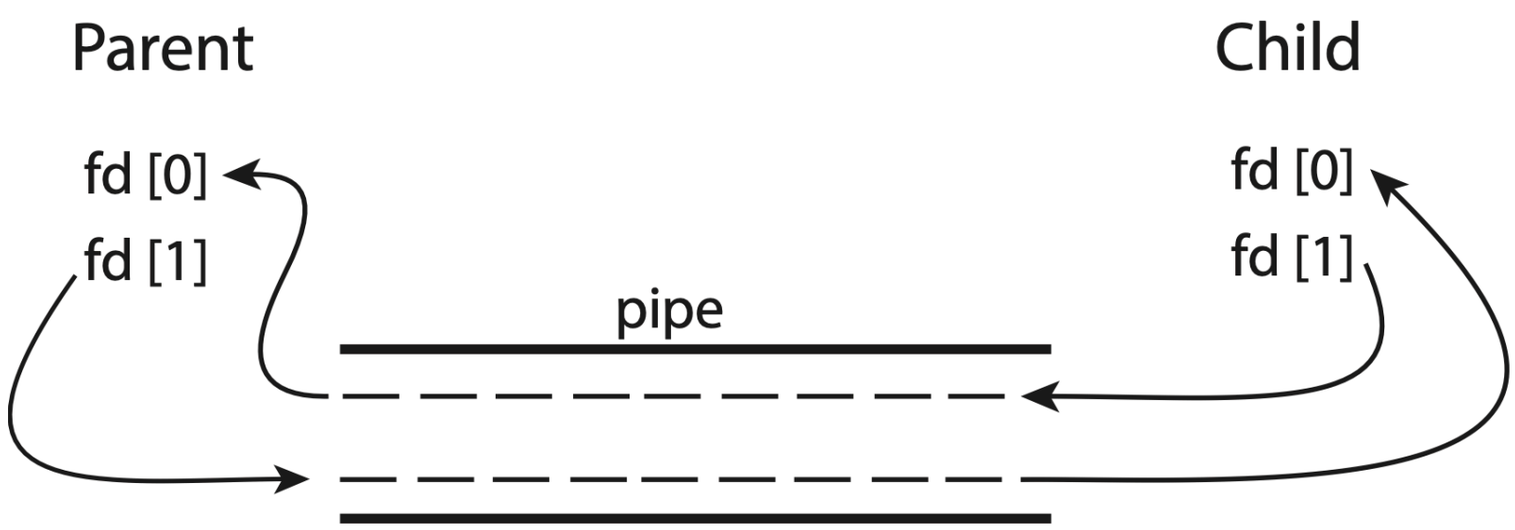
Ordinary Pipes (cont…)

- **Can be constructed as follows:**

```
pipe(int fd[])
```

where:

- `fd[0]` is the read-end of the pipe
- `fd[1]` is the write-end of the pipe



Using an Ordinary UNIX Pipe

```c
#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1

int main(void) {
    char write_msg[BUFFER_SIZE] = "Greetings";
    char read_msg[BUFFER_SIZE];
    int fd[2];
    pid_t pid;

    if (pipe(fd) == -1) {  /* create the pipe */
        fprintf(stderr, "Pipe failed");
        return 1;
    }

    pid = fork();   /* fork a child process */
    if (pid < 0) {  /* check for error */
        fprintf(stderr, "Fork Failed");
        return 1;
    }

    if (pid > 0) {  /* parent process */
        close(fd[READ_END]); /* close the unused end of the pipe */
        write(fd[WRITE_END], write_msg, strlen(write_msg) + 1);  /* write to pipe */
        close(fd[WRITE_END]);  /* close the write end of the pipe */
    } else {  /* child process */
        close(fd[WRITE_END]);  /* close the unused end of the pipe */
        read(fd[READ_END], read_msg, BUFFER_SIZE);  /* read from pipe */
        printf("Message read from pipe: %s", read_msg);
        close(fd[READ_END]);  /* close the read end of the pipe */
    }
    return 0;
}
```

Named Pipes

- **Named pipes are more powerful than ordinary pipes**
  - Communication is bidirectional
  - No parent-child relationship is necessary between the communicating processes
  - Several processes can use the named pipe for communication
  - Named pipe continue to exists even after processes terminate
- **Provided on both UNIX and Windows systems**
  - On POSIX systems: `man 3 mkfifo`

Everyday Pipes

- **Pipes are incredibly powerful and useful to use at the command line**
  - Feed output of one program to input of another
  - Feed output of '**`ps`**' program to input of '**`grep`**' to find the PID of a process

```
#> ps aux | grep -i Terminal
```

  - Rename .jpeg files to .jpg

```
#> find . -type f -name '*.jpeg'
    | while read n ;
      do mv "$n" "`echo "$n" | sed -e 's/.jpeg$/.jpg/'`" ;
      done
```