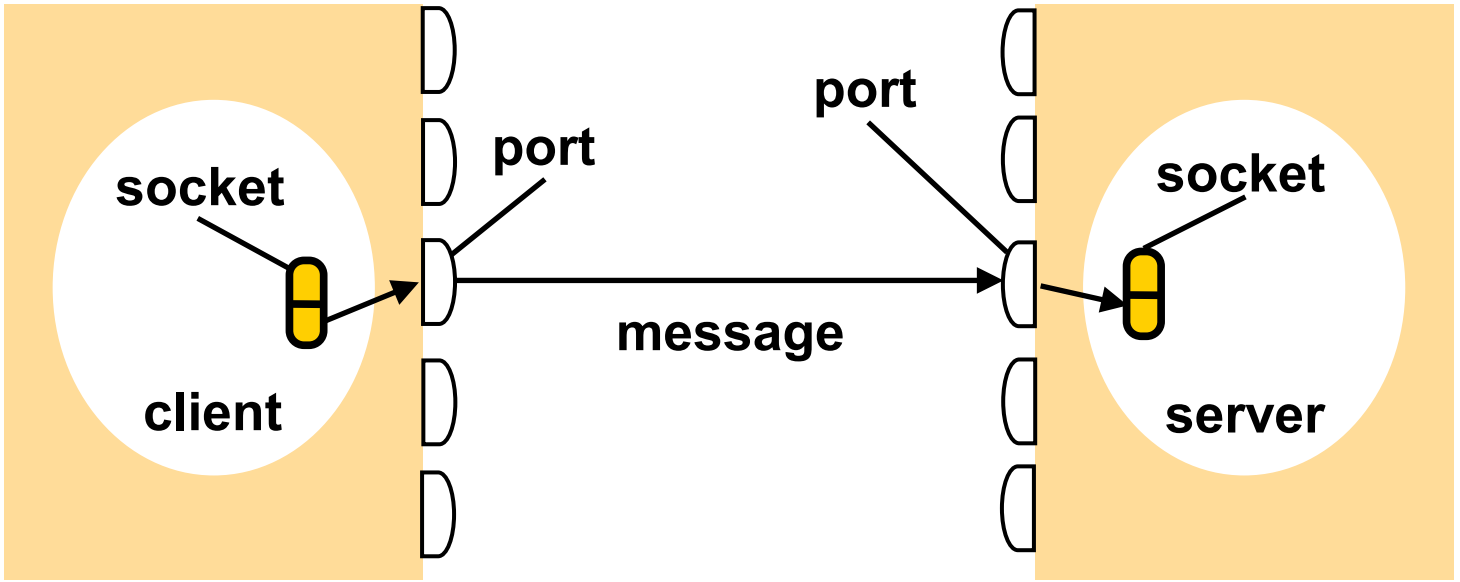
A Tutorial on Socket Programming

Sockets and Ports



- Socket: an endpoint for inter-process communication, which is bound to (Internet address, local port) pairs
- Interprocess communication: transmitting a message between a socket in one process and a socket in another process

Outline

- Socket Programming in C
- Socket Programming in Java

Sockets for C-Programmers on UNIX

- Sockets in BSD 4.x UNIX allow any process to communicate with itself or any other process
- System calls for UDP communication
  - socket: create a new socket and get its descriptor

    int socket (int domain, int type, int protocol);

    domain – communication domain (AF_INET)

    type – datagram or stream

    (SOCK_DGRAM or SOCK_STREAM)

    protocol – a particular protocol (normally 0)

    return value – a socket descriptor
  - close: destroy a socket when it is no longer needed

    int close(int s);

    s – socket descriptor

Sockets for C-Programmers on UNIX

- System calls (cont'd)
  - bind: bind a socket to an address (Internet address + port)

int bind (int s, struct sockaddr *socketAddr, int addrlength);

s – socket descriptor

socketAddr – socket address

addrlength – length (size) of socket address

return value – 0 means successful binding

```
struct sockaddr_in {
        short sin_family;
        u_short sin_port;
        struct in_addr sin_addr;
        ……
};
```

```
struct in_addr {
        union {
        ……
        u_long S_addr;
        } S_un;
};
```

Sockets for C-Programmers on UNIX

- System calls (cont'd)

```
void makeLocalSocketAddress(struct sockaddr_in *sa)
{  sa->sin_family = AF_INET;
   sa->sin_port = htons(0);
   sa->sin_addr.s_addr = htonl(INADDR_ANY); }
void makeRemoteSocketAddress(struct sockaddr_in *sa,
    char *hostname, int port)
{  struct hostent *host;
   sa->sin_family = AF_INET;
   if((host = gethostbyname(hostname))== NULL){
      printf("Unknown host name\n"); exit(-1); }
   sa->sin_addr = *(struct in_addr *) (host->h_addr);
   sa->sin_port = htons(port); }
```

any port

on local host

can be www.ntu.edu.sg or 123.124.125.126

a given port

Sockets for C-Programmers on UNIX

- System calls (cont'd)
  - sendto: send data to a remote socket

int sendto(int s, char *msg, int len, int flags, struct sockaddr *to, int tolen)

  - recvfrom: receive data from a remote socket

int recvfrom(int s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen)

s – local socket descriptor

msg/buf – message to be sent/buffer to receive message

len – length of message/buffer

flags – normally 0

to/from – address of remote socket

tolen/fromlen – length of remote socket address

return value – number of bytes sent/received

Sockets for C-Programmers on UNIX

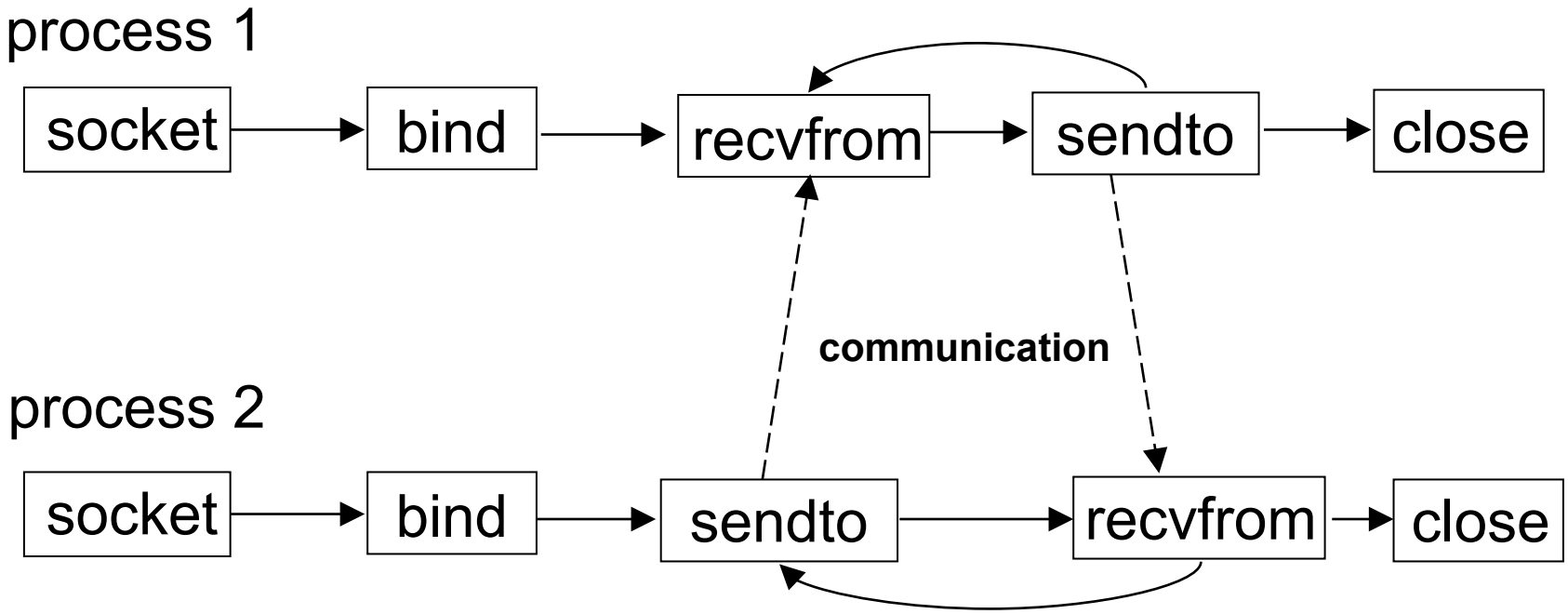
- System calls (cont'd)
  - select: detect whether a message has yet arrived (e.g., to test timeout)

```
#include <sys/time.h>
int anyThingThere(int s){
   unsigned long read_mask;
   struct timeval timeout;
   int n;
   timeout.tv_sec = 10; /*seconds wait*/
   timeout.tv_usec = 0; /*micro seconds*/
   read_mask = (1<<s);
   if((n = select(32, (fd_set *)&read_mask, 0, 0, &timeout))<0)
      perror("Select fail:\n");
   else printf("n = %d\n"); //n = 0 if no message has arrived
   return n;}
```

more details about select system call can be found at ftp://gaia.cs.umass.edu/cs653_1996/sock.ps

maximum value is FD_SETSIZE (sys/types.h)

UDP Communication using Sockets



- Need to bind sockets to addresses first
- Process knows where to respond (address of sending socket) after receiving a message

Example: Sockets for UDP Communication

Client: sending a message

```
s = socket(AF_INET, SOCK_DGRAM, 0)
//Internet communication domain
//datagram communication required
//0 stands for a protocol – UDP
…
bind(s,
   (struct sockaddr *)&clientAddress,
   sizeof(struct sockaddr_in))
…
sendto(s, message, strlen(message), 0,
   (struct sockaddr *)&serverAddress,
   sizeof(struct sockaddr_in))
…
```

Server: receiving a message

```
s = socket(AF_INET, SOCK_DGRAM, 0)
//Internet communication domain
//datagram communication required
//0 stands for a protocol – UDP
…
bind(s,
   (struct sockaddr *)&serverAddress,
   sizeof(struct sockaddr_in))
…
amount = recvfrom(s, buffer,
   buffersize, 0,
   (struct sockaddr *)&clientAddress,
   &addressLength)
…
```

Sockets for C-Programmers on UNIX

- Additional system calls for TCP communication
  - socket: create a new socket and get its descriptor
  - bind: bind a socket to an address
  - close: destroy a socket when it is no longer needed
  - sendto → write: send data over the connection
  - recvfrom → read: receive data over the connection

int write(int s, char *msg, int len)

int read(int s, char *buf, int len)

s – local socket descriptor

msg/buf – message to be sent/buffer to receive message

len – length of message/buffer

return value – number of bytes sent/received

Sockets for C-Programmers on UNIX

- Additional system calls (cont'd)
  - listen: (server) listen on its socket for client requests for connections

```
int listen(int s, int backlog);
```

s – socket descriptor

backlog – maximum number of connections that can be queued at the socket

return value – 0 for success and –1 for error

Sockets for C-Programmers on UNIX

- Additional system calls (cont'd)
  - accept: (server) accept a connection, and obtain a new socket for communication with the client

  int accept (int s, struct sockaddr *clientAddress, int *clientLength);

  s – listening socket descriptor

  clientAddress – address of remote socket

  clientLength – length of remote socket address

  return value – a new socket descriptor (already bound locally) for communication with the client

Sockets for C-Programmers on UNIX

- Additional system calls (cont'd)
  - connect: (client) binding + request a connection via the socket address of listening process

    int connect(int s, struct sockaddr *server, int addrLen)

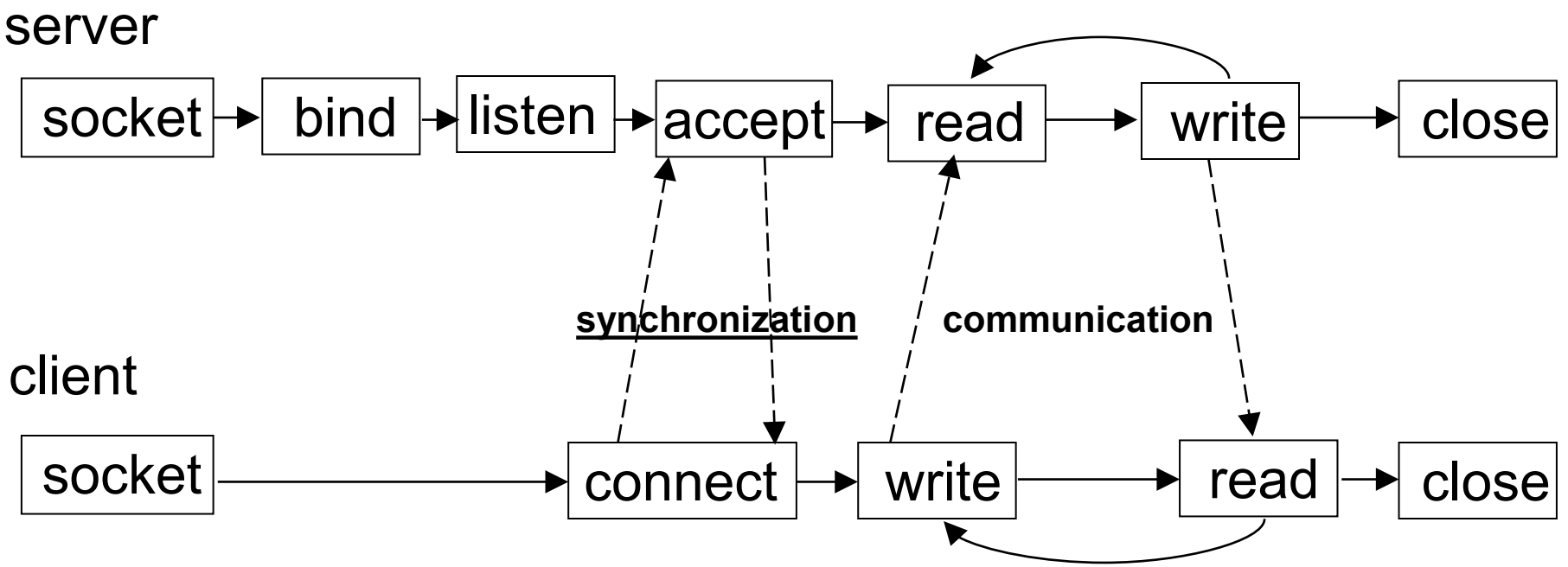
    s – local socket descriptor

    server – address of remote (server) socket

    addrLen – length of remote (server) socket address

    return value – 0 for success and –1 for error

TCP Communication using Sockets



- Need to establish a connection before communication
- No binding needed on the client side (done by connect call), server knows where to send after accepting a connection request
- Server normally forks a child process to handle the request, while the parent process continues to listen

Example: Sockets for TCP Communication

Client: requesting a connection

```
s = socket(AF_INET, SOCK_STREAM, 0)
//Internet communication domain
//stream communication required
//0 stands for a protocol – TCP
…
connect(s,
   (struct sockaddr *)&serverAddress,
   sizeof(struct sockaddr_in))
…
write(s, message, strlen(message))
…
```

Server: listening and accepting a connection

```
s = socket(AF_INET, SOCK_STREAM, 0)
…
bind(s,
   (struct sockaddr *)&serverAddress,
   sizeof(struct sockaddr_in))
listen(s, 5)
//specify max. number of connections
//that can be queued at a socket
…
sNew = accept(s,
   (struct sockaddr *)&clientAddress,
   &addressLength)
…
amount = read(sNew, buffer, buffersize)
…
```

Outline

- Socket Programming in C
- Socket Programming in Java

UDP Programming in Java

- UDP Programming in Java
  - DatagramPacket class
    - fields: data, length, the address of the destination host, and the port number that the host is listening on
    - methods: getData, getLength, getAddress, getPort
  - DatagramSocket class
    - send and receive methods: send and receive an instance of DatagramPacket
    - setSoTimeout method: set a timeout
      - If the timeout expires, the receive method will throw a SocketTimeoutException and exit

Example: UDP client sends to server and gets a reply

```
import java.net.*;
import java.io.*;
public class UDPClient{
   public static void main(String args[]){
      //args give message contents and server hostname
      DatagramSocket aSocket = null;
      try {
         aSocket = new DatagramSocket();  //use a free local port
         byte[] m = args[0].getBytes();  //a buffer for sending
         InetAddress aHost = InetAddress.getByName(args[1]);
         //translate user-specified hostname to Internet address
         int serverPort = 6789;
         //need a port number to construct a packet
```

Example: UDP client sends to server and gets a reply

```
            DatagramPacket request = new DatagramPacket(m,
                m.length, aHost, serverPort);
            aSocket.send(request);
            //send packet using socket method
            byte[] buffer = new byte[1000];  //a buffer for receive
            DatagramPacket reply = new DatagramPacket(buffer,
                buffer.length);  //a different constructor
            aSocket.receive(reply);  //from which port?
            System.out.println("Reply: "
                + new String(reply.getData()));
        }
        …… //handle exceptions
        } finally {if (aSocket != null) aSocket.close();}
    }
}
```

Example: UDP server repeatedly receives a request and sends it back to the client

```
import java.net.*;
import java.io.*;
public class UDPServer{
   public static void main(String args[]){
      DatagramSocket aSocket = null;
      try{
         aSocket = new DatagramSocket(6789);
         //bound to host and port
         byte[] buffer = new byte[1000];
         while(true){
            DatagramPacket request
               = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(request);  //blocked if no input
```

Example: UDP server repeatedly receives a request and sends it back to the client

```
                DatagramPacket reply = new DatagramPacket(
                    request.getData(), request.getLength(),
                    request.getAddress(), request.getPort());
                //to reply, send back to the same port
                aSocket.send(reply);
            }
        }
        ..... //handle exceptions
        } finally {if (aSocket != null) aSocket.close();}
    }
}
```

TCP Programming in Java

- TCP Programming in Java
  - Socket class
    - Used by a pair of processes with a connection
    - Client constructs a socket by specifying hostname and port of a destination host
    - Constructor creates a socket, binds to a local port and connects to the remote socket
    - Methods: getInputStream and getOutputStream (which return InputStream and OutputStream for reading and writing bytes respectively)
  - ServerSocket class
    - Used by a server to create a socket at a server port to listen to connect requests from clients
    - Method: accept (which returns a new socket for communicating with the client)

Example: TCP client makes connection to server, sends request and receives reply

```
import java.net.*;
import java.io.*;
public class TCPClient {
    public static void main (String args[]) {
        //args give message contents and server hostname
        Socket s = null;
        try{
            InetAddress aHost = InetAddress.getByName(args[1]);
            //translate user-specified hostname to Internet address
            int serverPort = 7896;
            s = new Socket(aHost, serverPort);
            //bind to remote server and port
            DataInputStream in =
                new DataInputStream(s.getInputStream());
                //for reading (receiving)
```

Example: TCP client makes connection to server, sends request and receives reply

```
            DataOutputStream out =
                new DataOutputStream(s.getOutputStream());
                //for writing (sending)
            out.writeUTF(args[0]);
            //string sent is encoded in Unicode Text Format
            String data = in.readUTF();  //read from server
            System.out.println("Received: " + data) ;
        }
        …… //handle exceptions
        } finally {if (s != null) s.close();}
    }
}
```

Example: TCP server makes a connection for each client and then echoes the client's request

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
        try{
            int serverPort = 7896;
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while(true) {
                Socket clientSocket = listenSocket.accept();
          //once accepts a connection, serversocket produces a socket
                Connection c = new Connection(clientSocket);
                //connection is a class defined on the next slide
            }
        }
        …… //handle exceptions
}}
```

Example: TCP server makes a connection for each client and then echoes the client's request

```
class Connection extends Thread {
    DataInputStream in;
    DataOutputStream out;
    Socket clientSocket;
    public Connection (Socket aClientSocket) {  //constructor
        try {
          clientSocket = aClientSocket;
          in =
            new DataInputStream(clientSocket.getInputStream());
          out =
            new DataOutputStream(clientSocket.getOutputStream());
          this.start();  //starts it as a thread
        }
        …… //handle exceptions
}
```

Example: TCP server makes a connection for each client and then echoes the client's request

```
public void run(){
        try {  //an echo server
            String data = in.readUTF();  //read in
            out.writeUTF(data);  //send back
        }
        ..... //handle exceptions
        } finally { if (clientSocket != null) clientSocket.close();}
    }
}
```

- Purpose of making a new thread to communicate with the client: not to miss connection setup requests from clients