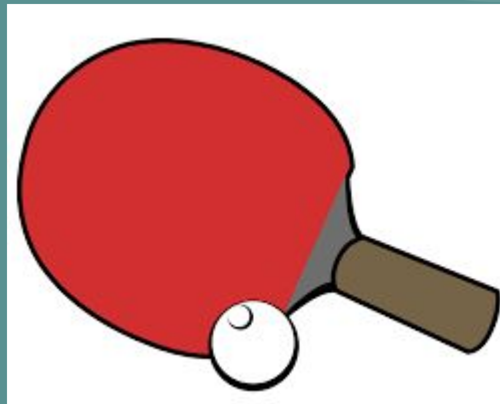
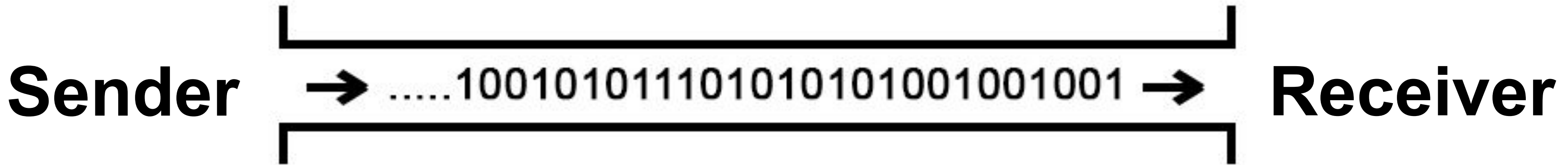
Ping Pong



KTH Kista
Stockholm
02.12.2019

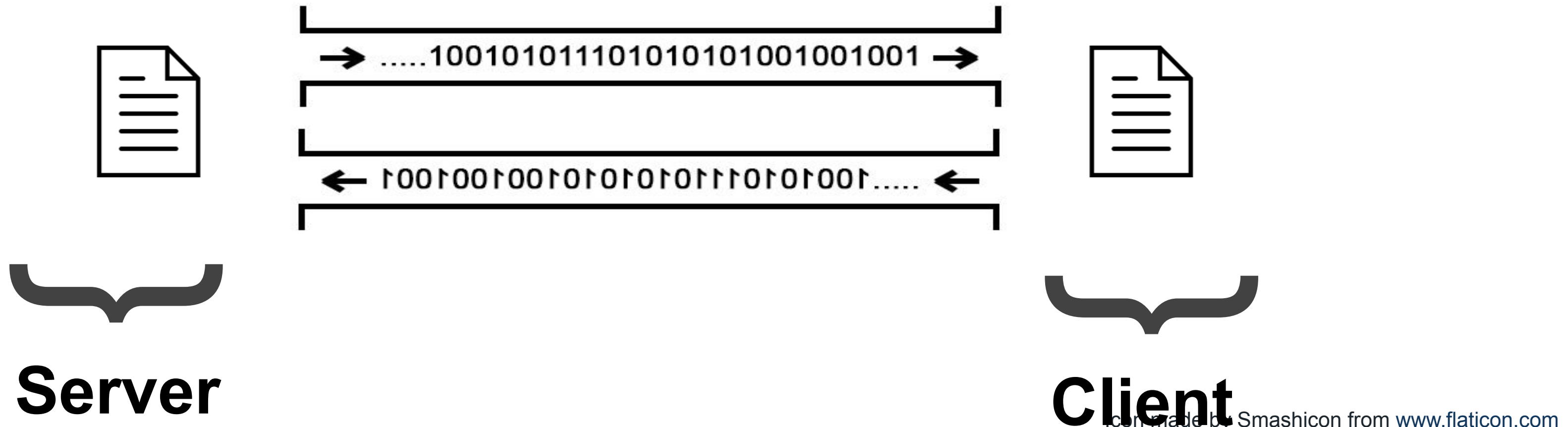
Pipes & Flow Control

- Pipes are one-directional streams of data that operates within a single machine.
- When the pipe is full, the sender will be suspended until the receiver has cleared space for new data
- Can be accessed either through a ***Shared memory (Fork)*** or a named ***File***



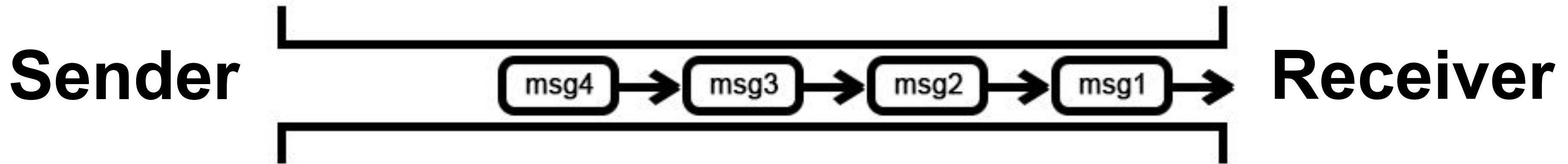
Sockets - Higher level abstraction

- One limitation of pipes is that they are **one way**
- Sockets provide abstraction for a **two way** connections and **location transparency** (you can send things to you friends!)
- Supports communication both locally and over a network
- Both stream (TCP) and datagrams (UDP) are options



Datagrams

- UDP
- Every messages includes the length of that actual message.
- Not reliable

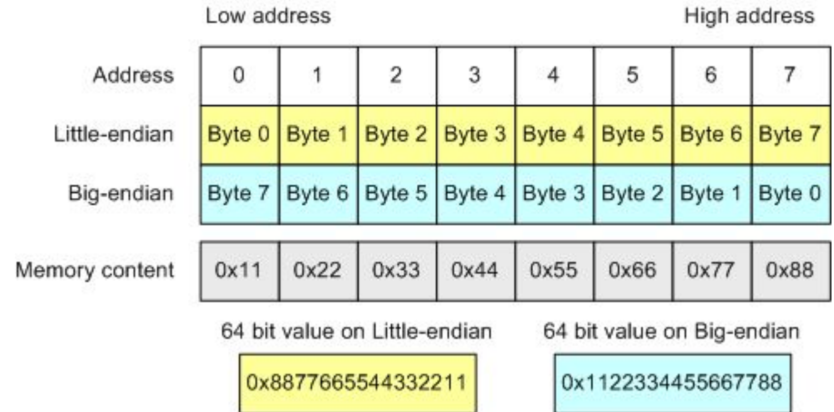


Sending data - Marshalling

- Encode data structure or other data as pure binary data for sending over network
- The way we encode and decode datatypes is known as **marshalling**
- Can be a problem when we are on different systems and compilers sender and receiver

Sending data - Endianness

- Network byte order! (Big endian)



Sending data - Where is the server?

- Two primary ways of identifying a socket in linux
- AF_INET vs AF_UNIX
- AF_INET is a socket “file” (same interface) but not in the namespace of the file system

Sockets and the file system

- Local sockets and named pipes are stored as special file types
- Other file types in linux: -, c, b, d, l

```
-rw-rw-r-- 1 hannesr hannesr  908 Oct 26 20:49 ping.c
srwxrwxr-x 1 hannesr hannesr    0 Dec  1 17:47 pingpong
-rwxrwxr-x 1 hannesr hannesr  13K Dec  1 17:47 pong
-rw-rw-r-- 1 hannesr hannesr  680 Oct 26 20:49 pong.c
prw-r--r-- 1 hannesr hannesr    0 Dec  1 17:41 sesame
```

Exam Questions

Exam Question 1

We can easily do a ***fork()*** and then set up stdin and stdout for the two processes to communicate through a so called **pipe**.
How can we achieve the same for two processes although they're not created using a fork()?
Meaning we still want one process create a pipe that another process can read from.

Exam Question 1

We can easily do a ***fork()*** and then set up stdin and stdout for the two processes to communicate through a so called **pipe**.
How can we achieve the same for two processes although they're not created using a fork()?
Meaning we still want one process create a pipe that another process can read from.

Answer:

One of the processes can register a pipe with a agreed upon file name, using the ***mkfifo*** command. Now the other pipe can open that pipe as a file, using the specified name.

Exam Question 2

A so called ***pipe*** is a simple way to send data from one process to another. It does have its limitations and a better way is to use so called ***sockets***. If we use a stream socket between two processes we will have several advantages.
Describe two advantages that a steam socket gives us that we will not have if we use a pipe.

Exam Question 2

A so called ***pipe*** is a simple way to send data from one process to another. It does have its limitations and a better way is to use so called ***sockets***. If we use a stream socket between two processes we will have several advantages.
Describe two advantages that a steam socket gives us that we will not have if we use a pipe.

Answer:

When using a stream socket we have a **two-way** communication, unlike the one-directional pipes.

Pipes are also limited to applications running on the same machine, unlike sockets that can be utilized to pass data over a network. They provide so called **location transparency**.

Exam Question 3

Below is code were we open a socket and use the name space `AF_INIT`. We will then be able to address a server using a port number and IP-address. There are other name spaces that we can use when working with sockets. Name one and decribe its advantages and disadvantages it might have.

```
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = inet_addr(SERVER_IP);
```

Exam Question 3

Below is code were we open a socket and use the name space **AF_INIT**. We will then be able to address a server using a port number and IP-address. There are other name spaces that we can use when working with sockets. Name one and decribe its advantages and disadvantages it might have.

```
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = inet_addr(SERVER_IP);
```

Answer:
For example we have the ***AF_UNIX*** domain socket, that's constrained to working with passing data between applications on the same machine. It works by binding a specific file on file system that can opened from other applications

This communication is faster, since we can skip the overhead of for example a TCP protocol. But as stated is constrained to only working on communication within the machine.

Exam Question 4

2.2 pipes [2 points]

The program below opens a pipe and iterates a number of times (ITERATIONS) where each iteration sends a number (BURST) of messages ("0123456789"). We need to handle the situation where the receiving process will not keep up with the sender; how do we implement ow-control to avoid buffer overflow?

```
int main() {
  int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  mkfifo("sesame", mode);
  // add flow control


  int flag = O_WRONLY;
  int pipe = open("sesame", flag);

  /* produce quickly */
  for(int i = 0; i < ITERATIONS; i++) {
    for(int j = 0; j < BURST; j++) {
      write(pipe, "0123456789", 10);
      // add flow control


    }
    printf("producer burst %d done\n", i);
  }
  printf("producer done\n");
}
```

Exam Question 4

2.2 pipes [2 points]

The program below opens a pipe and iterates a number of times (ITERATIONS) where each iteration sends a number (BURST) of messages ("0123456789"). We need to handle the situation where the receiving process will not keep up with the sender; how do we implement ow-control to avoid buffer overflow?

Answer: Pipes have built in flow control so we do not have to do anything.

```
int main() {
  int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  mkfifo("sesame", mode);
  // add flow control


  int flag = O_WRONLY;
  int pipe = open("sesame", flag);

  /* produce quickly */
  for(int i = 0; i < ITERATIONS; i++) {
    for(int j = 0; j < BURST; j++) {
      write(pipe, "0123456789", 10);
      // add flow control


    }
    printf("producer burst %d done\n", i);
  }
  printf("producer done\n");
}
```

Exam question 5

2.3 SOCK_WHAT [2 points*]

When you create a socket you can choose to create a *SOCK_STREAM* or *SOCK_DGRAM*. Which properties differ and when is it an advantage to choose one over the other.

Exam question 5

2.3 SOCK_WHAT [2 points*]

When you create a socket you can choose to create a *SOCK_STREAM* or *SOCK_DGRAM*. Which properties differ and when is it an advantage to choose one over the other.

Exam question 5

Answer: The big difference is that *SOCK_STREAM* is a double direction connection providing a sequence of bytes while *SOCK_DGRAM* is a one directional channel for messages of limited size.

The advantage *SOCK_DGRAM* is that the receiver will receive one message at a time and need not think about how to divide a sequence of bytes into messages. If the messages are of limited size it is almost always better to use

SOCK_DGRAM. The order is however not guaranteed nor that messages actually arrive. If this is important one has to implement a protocol to keep the order and request resending.