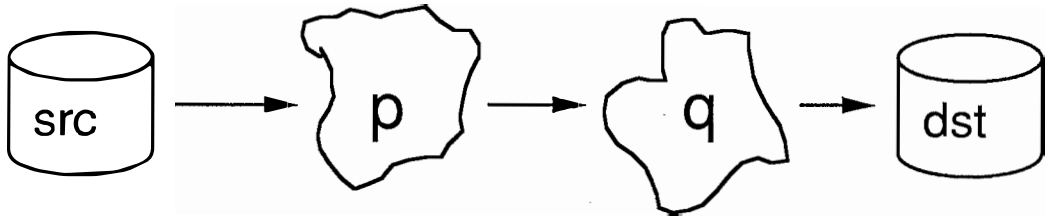
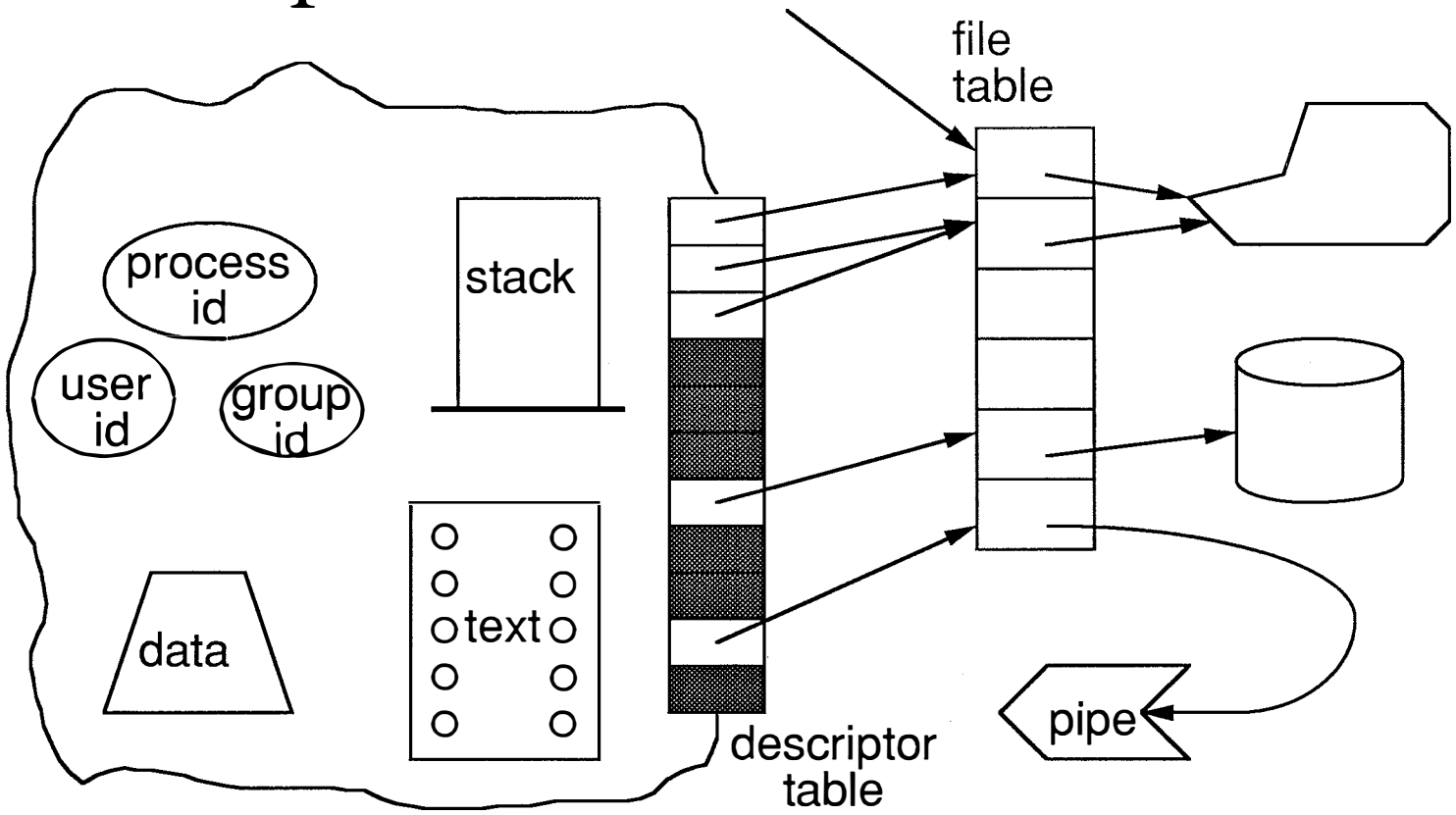
UNIX process model

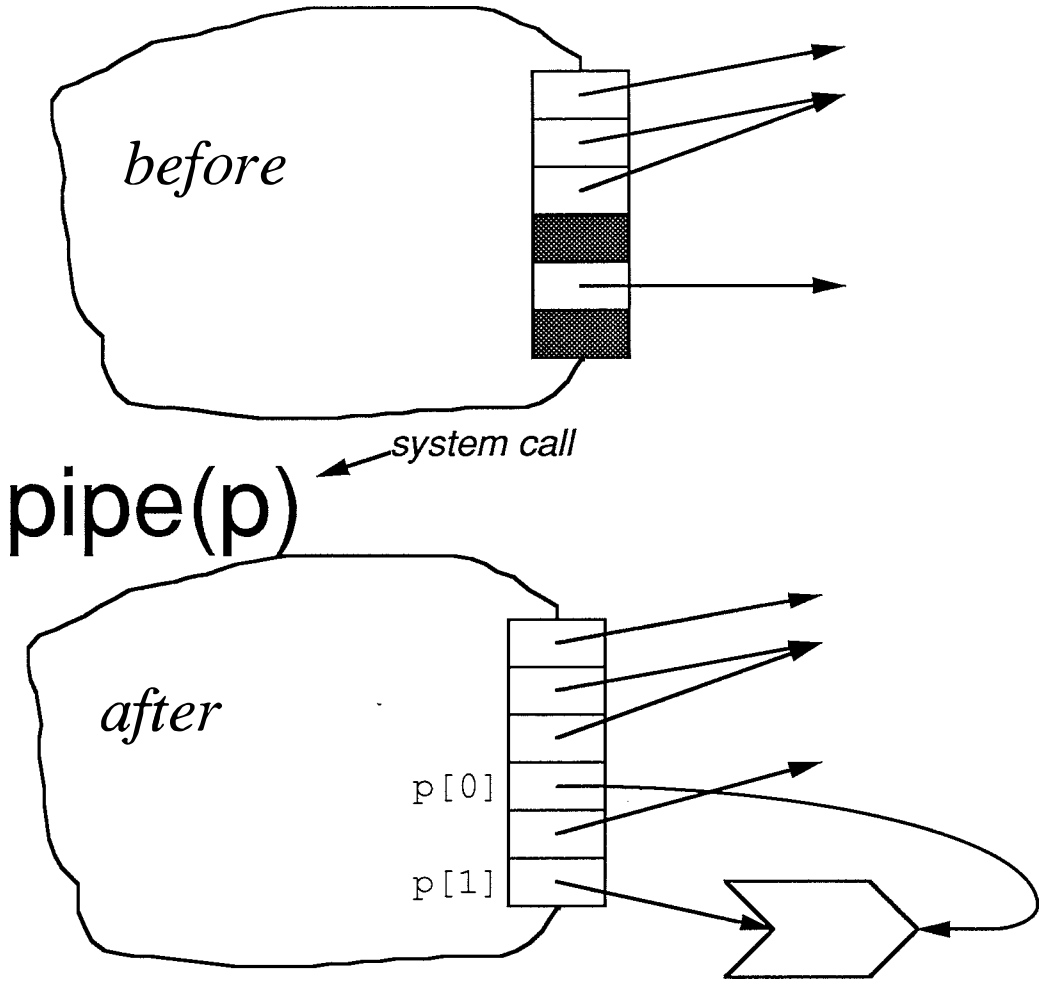
% p <src | q >dst



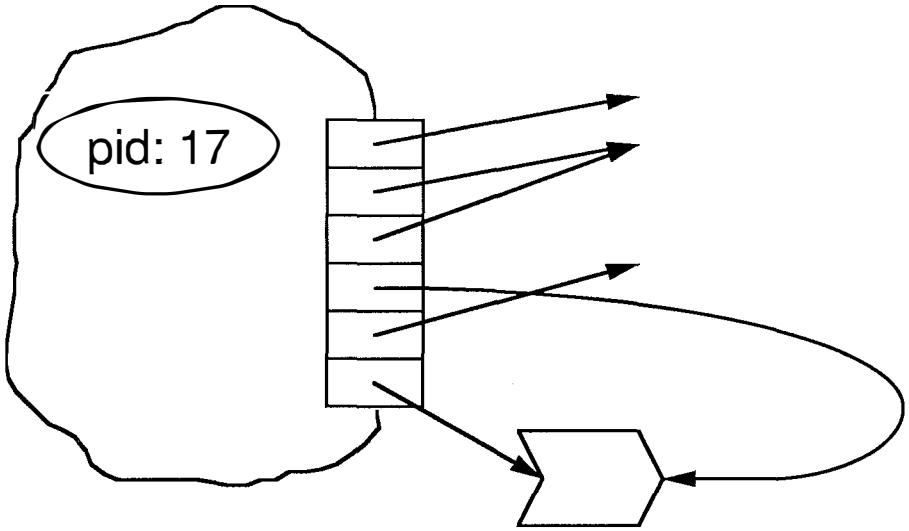
UNIX process



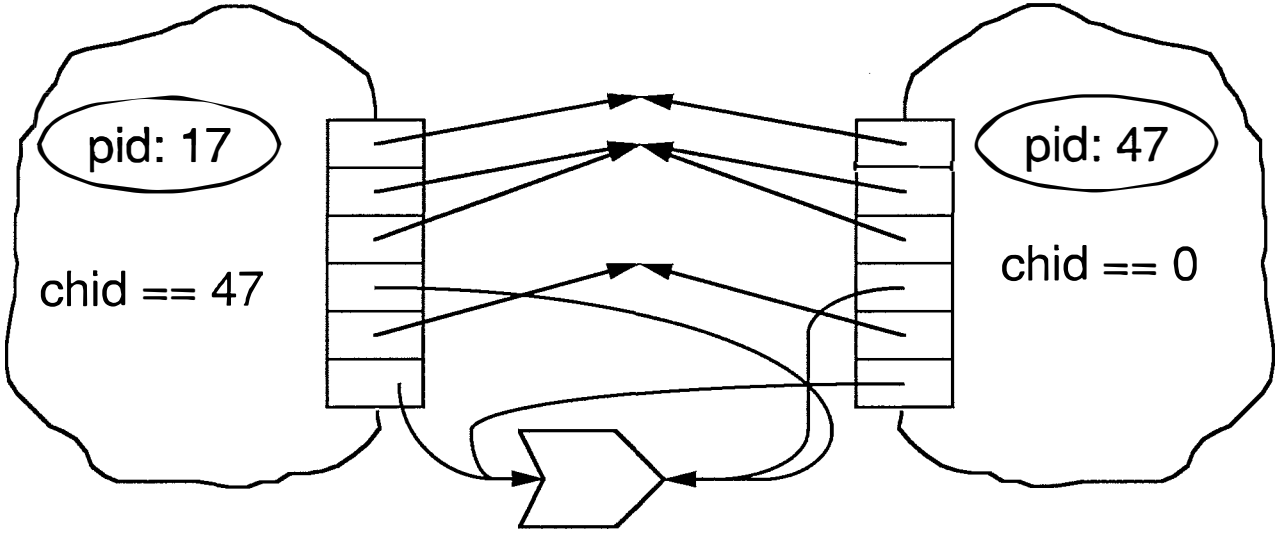
Creating a pipe in Unix



How two processes get a pipe



chid = fork();



How does a process invoke a program?

execl(*program*, *arguments*, ...);

- process is replaced by *program* running with *arguments*
- text, stack, and data are replaced
- file descriptors, process id remain
- if successful, does not return

For example:
execl("/bin/grep", "grep", "fudge", 0);

How does a process die?

exit(return-code);

- process becomes **ZOMBIE**

How is a ZOMBIE reaped?

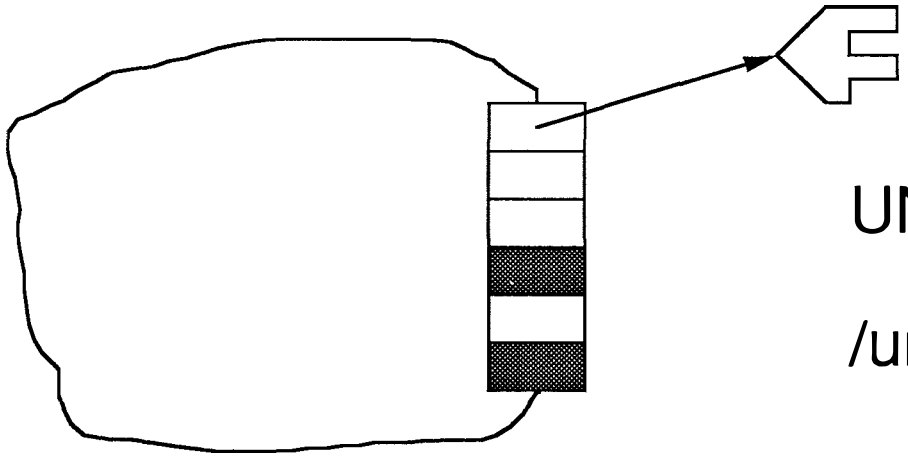
- parent must wait on dead children

chid = wait(&status);

Single-machine IPC

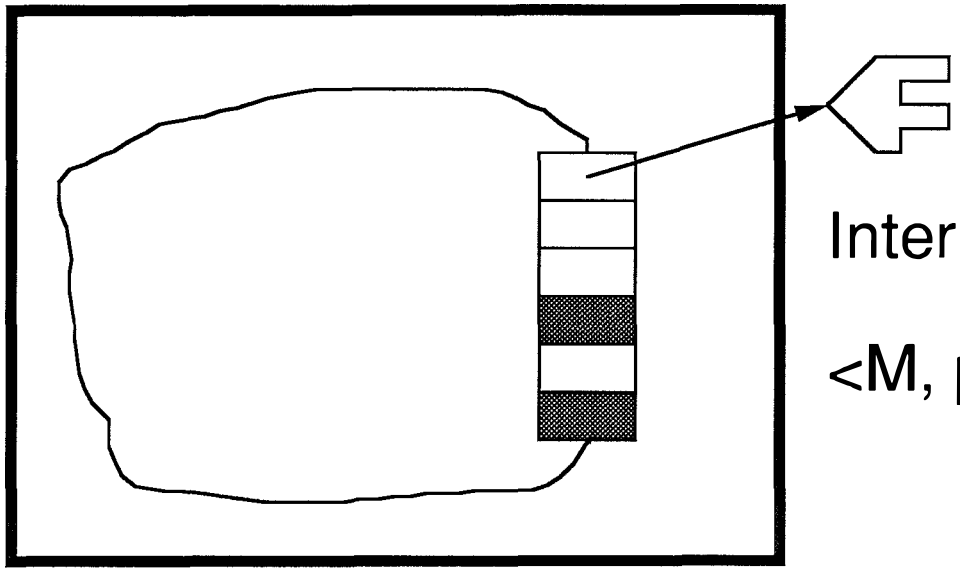
real UNIX	*System V*
pipes	named pipes
signals	shared memory
files	semaphores
file locks	message queues
	record locks

Socket domains



UNIX

/unc/foo/onu



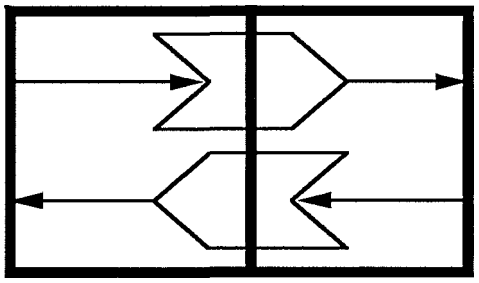
Internet

<M, port, protocol>

Socket types

SOCK_STREAM

reliable duplex communication



2 TCP sockets = 2 pipes

1 TCP socket ≠ 1 pipe

SOCK_DGRAM

unreliable message transmission

s = socket(*domain*, *type*, *protocol*);

creates an unnamed socket

domain could be Unix (AF_UNIX)
or internet (AF_INET)

type could be datagram (SOCK_DGRAM)
or virtual circuit (SOCK_STREAM)

protocol really ought to be 0

bind(*s*, &*name*, *name_size*);

gives the socket an external name

Parts of an Internet name

struct sockaddr_in s;

s.sin_family

socket family (AF_INET)

s.sin_port

port number
0, UNIX chooses -- 1..1023 privileged

s.sin_addr.s_addr

internet host number, e.g. 128.109.138.84
INADDR_ANY, UNIX chooses

Mappings

gethostbyname

string --> Internet addresses

getservbyname

string --> port, protocol of service
well-known (in /etc/services)

Stream sockets (TCP)

Uses a server-client paradigm, when initializing ---

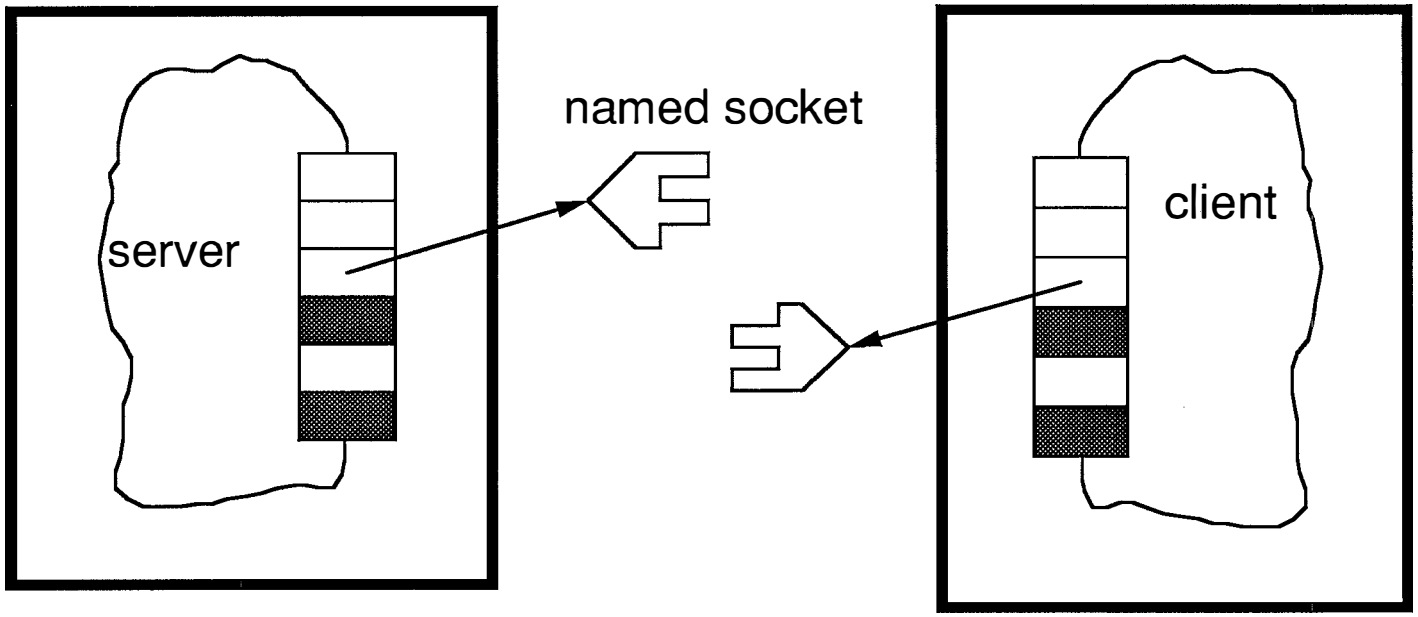
A server process
(1), creates an external socket
(2), accepts new connections
- socket -- creates socket
- bind -- gives external name
- listen(s, backlog)
  -- sets number of queueable requests

Clients (many)
- socket -- creates socket

The connection....

Server
  ns = accept(s, &client-address, &clnten);

client
  connect(s, &server-address, svrlen);



WOW!! regular I/O exists!

```
read(fd, buff, ...);
write(fd, buff, ...);
close(fd);
```

After the connection (usually)...

server creates a child
  child sets up standard I/O
  child exec's & provides real service

  parent accepts new connections

Datagrams (UDP)

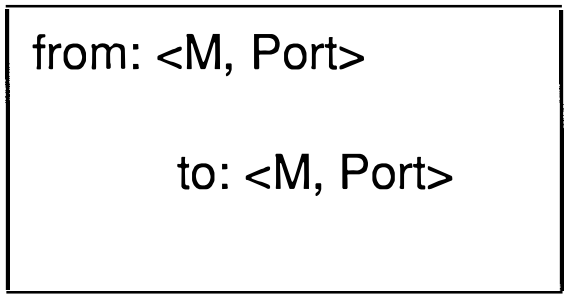
May be client-server. May be symmetric.

Both processes create a socket.
- s = socket(AF_INET, SOCK_DGRAM, 0);

One (or maybe just both) name the socket.
- bind(s, &name, ...);

Connectionless I/O

sending

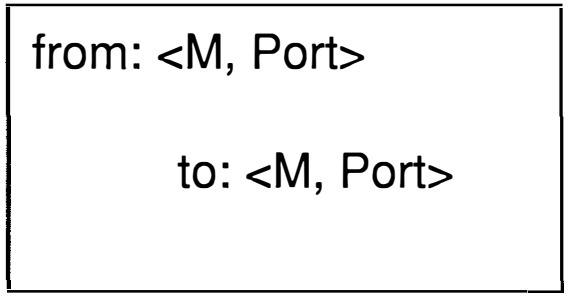


sendto(s, buf, buf-len, flags, to, to-len)

- s -- socket to send on (from addr)
- buf -- message to send
- flags -- 0, *if you're a reasonable person*
- to -- destination port

Connectionless I/O

receiving

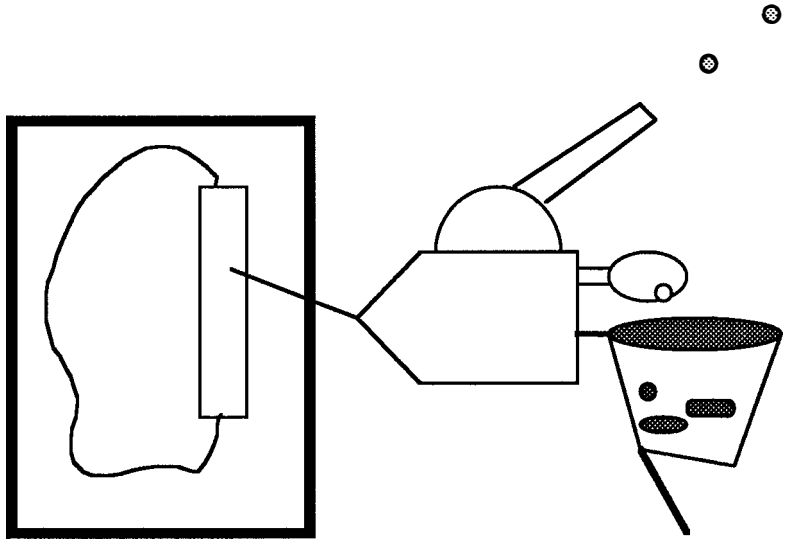


cc = recvfrom(s, buf, blen, flags, from, from-len)

- s - socket to receive on (to addr)
- buf - buffer for received message
- flags - 0 or MSG_PEEK, *if you can't wait*
- from - port address of sender

BSD 4.X (and UDP) datagrams

limited in length
not promised to be
- sequenced
- reliable
- unduplicated



Multiplexing I/O

```
ready = select(fds, &rm, &wm, &em, t);
```

- fds -- number of file descriptors
- ready -- number ready
- rm -- read mask
- wm -- write mask
- em -- exceptions mask
- t -- time out

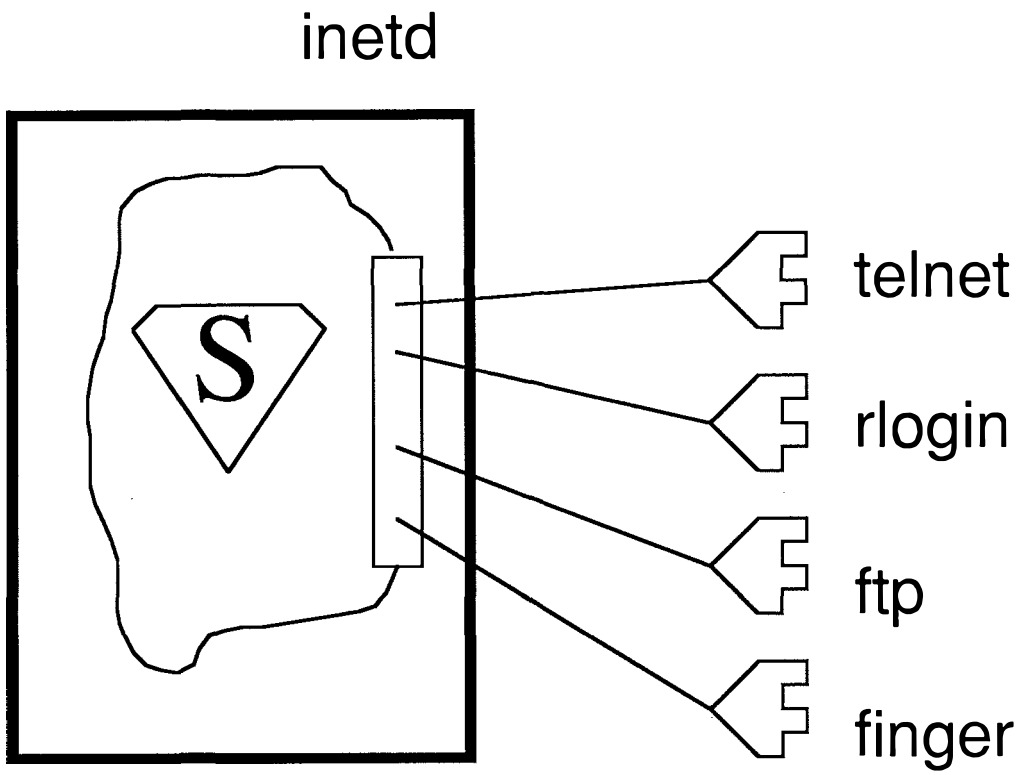
For example, awaiting input from standard input or socket s for one second:

```
Mask = set bits for 1 and s;
T = one second;
ready = select(30, &Mask, 0, 0, T);
```

ADA and BSD?

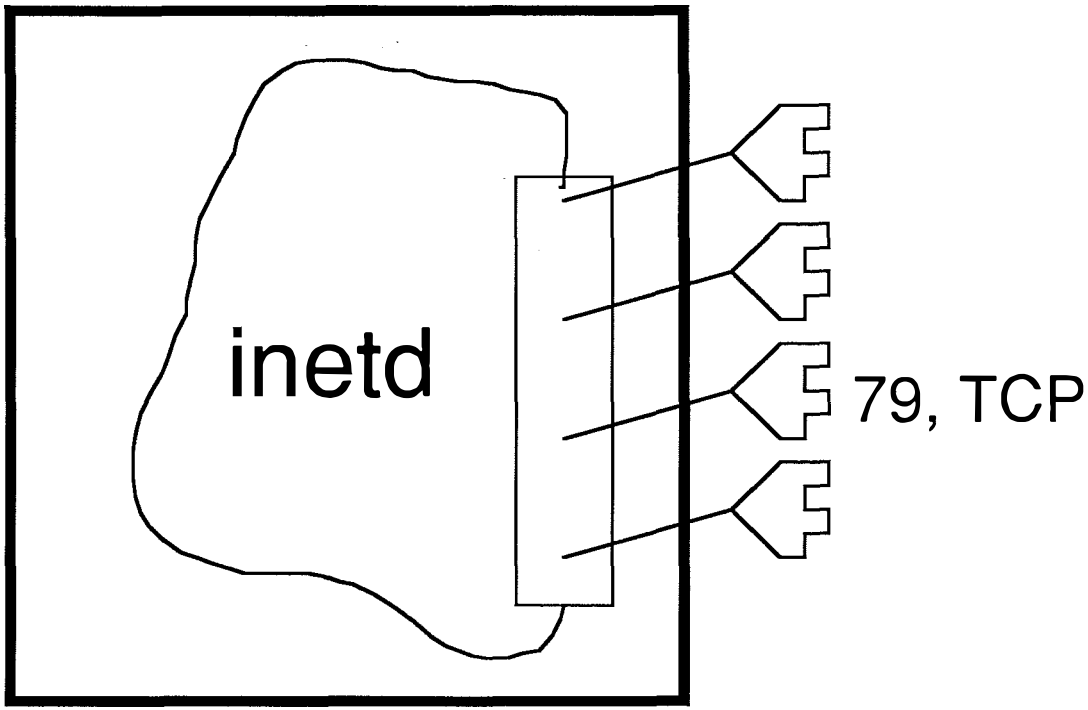
```
select
  when P ==>
    accept op1(...)
    ....
  or when Q ==>
    accept op2(....)
    ....
  or when R ==>
    delay T
    ....
end select
```

SUPERSERVER



provides services in /etc/servers
(or /etc/inetd.conf)

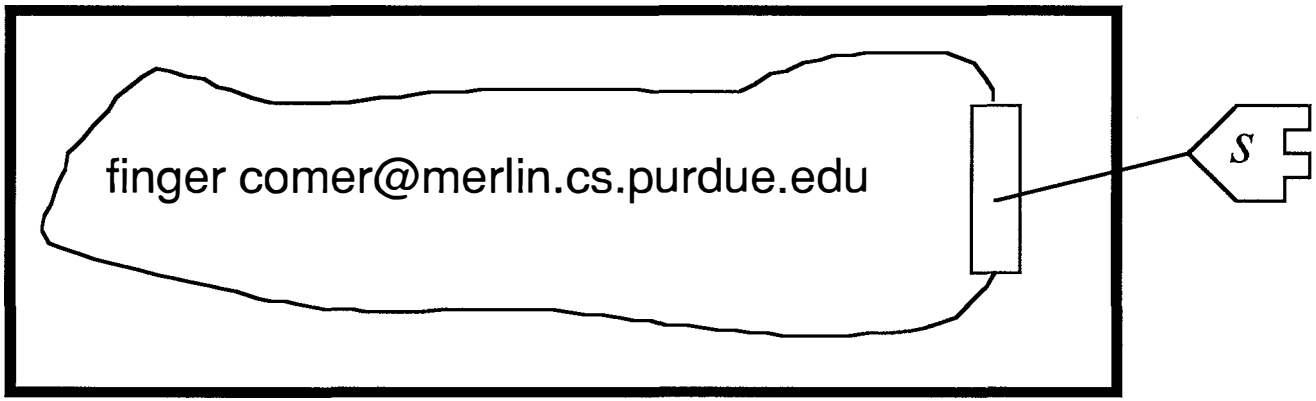
finger tcp /usr/etc/in.fingerd
from /etc/servers



View after booting

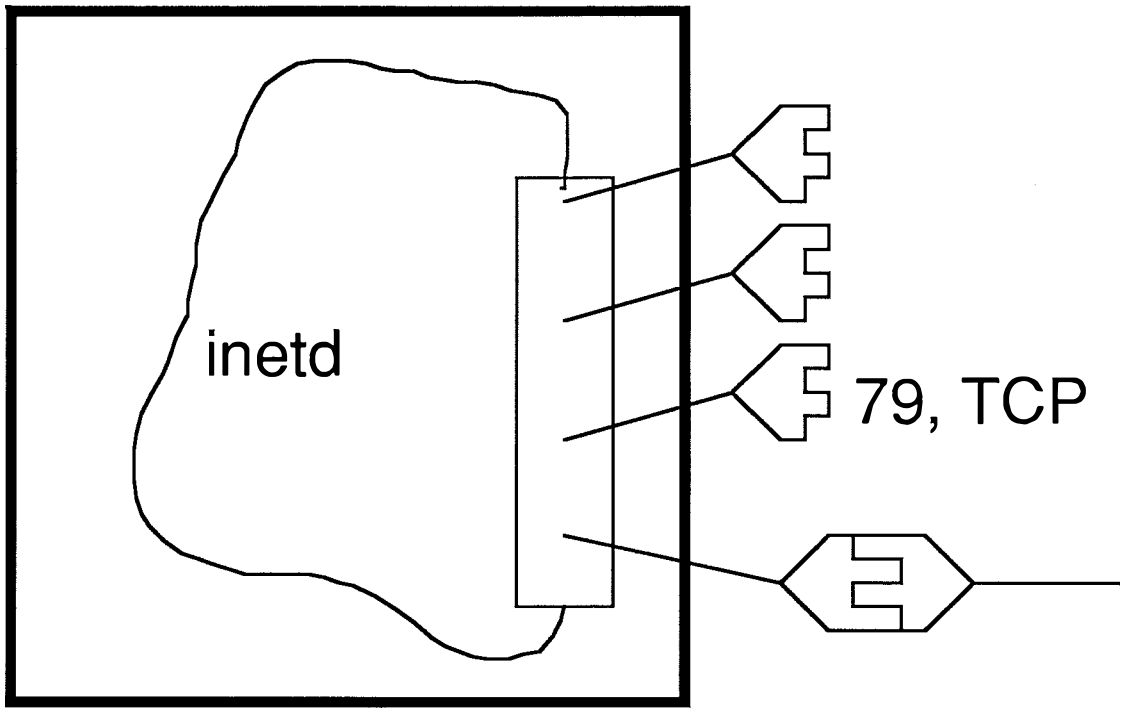
Typed at a client

% finger comer@merlin.cs.purdue.edu

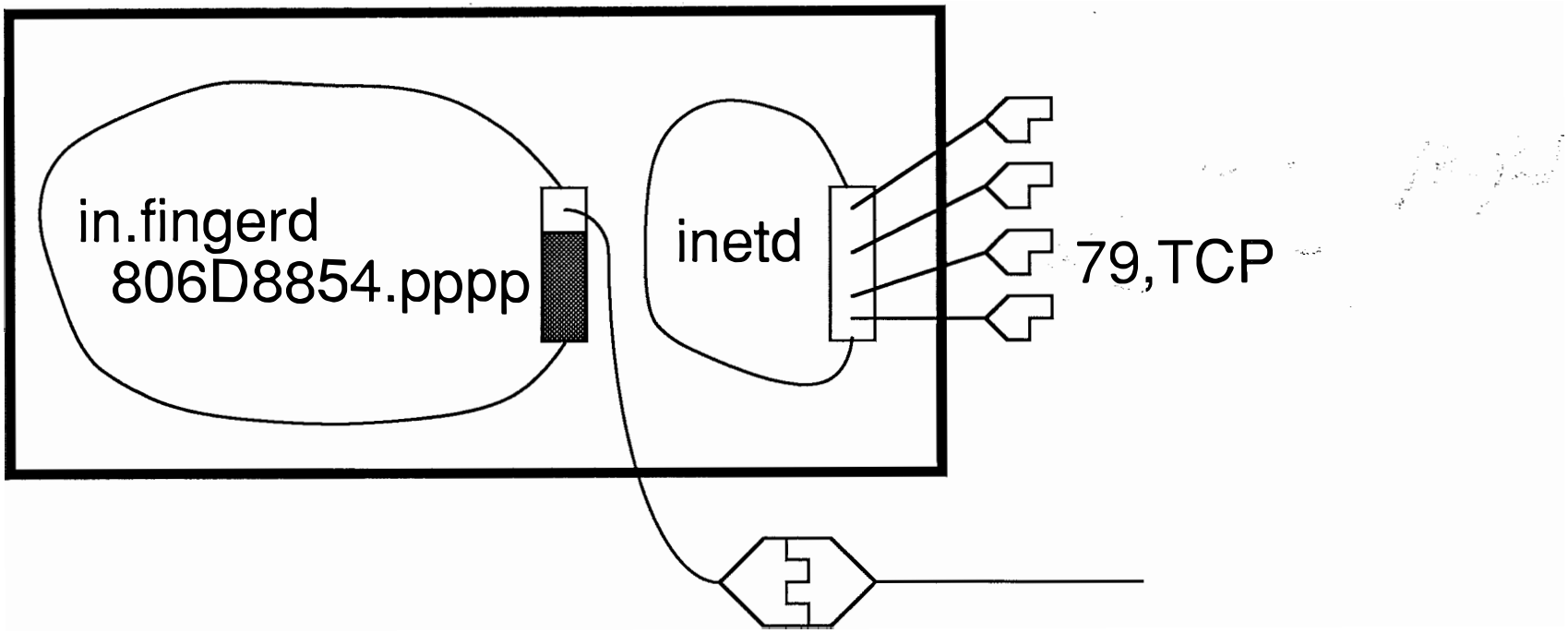


connect(s, 128.10.2.3, 79)

inetd accepts connection

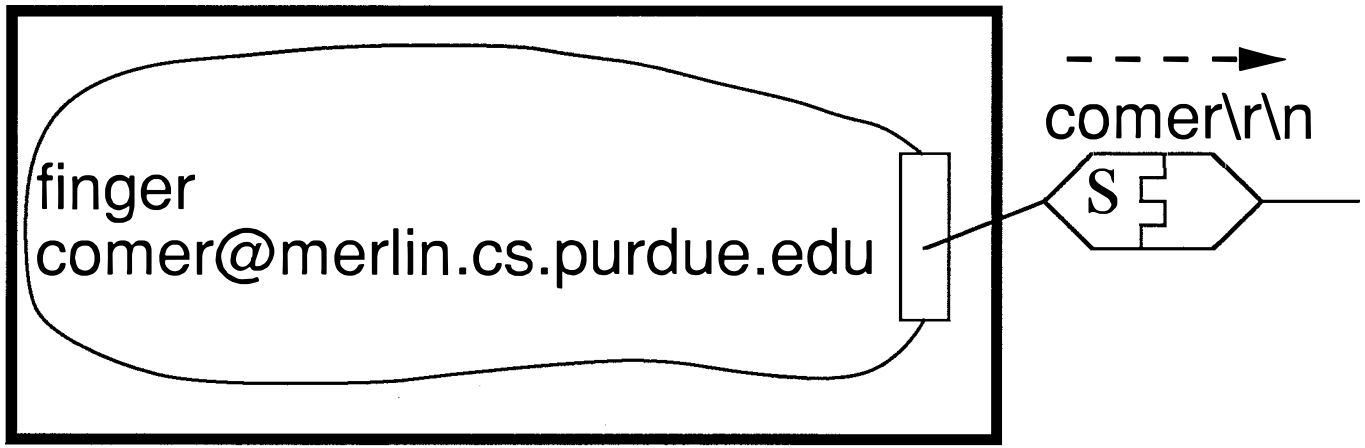


inetd calls finger daemon



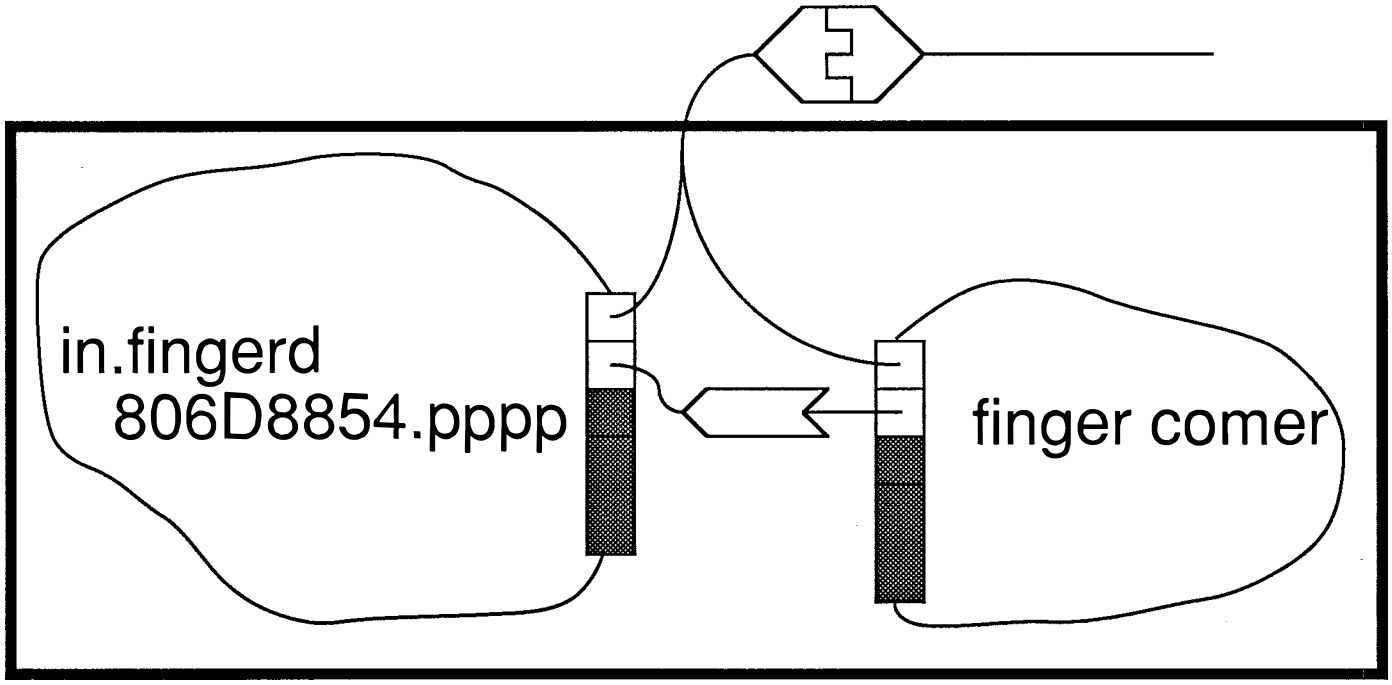
client sends fingered user

% finger comer@merlin.cs.purdue.edu



write(s, "comer\r\n");

in.fingerd invokes finger



in.fingerd
- creates pipes,
- dups pipes to right places,
- closes unused pipe ends,
- execs finger
- adds needed '\r'