Advanced Programming in the UNIX Environment

Week 09, Segment 2:
socket(2) (PF_LOCAL)

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

socket(2)

```
#include <sys/socket.h>

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

Returns: fd if ok, -1 otherwise

socket(2) creates an endpoint for communication and returns a descriptor.

The *domain* specified selects the address- or name space of the socket, which selects the protocol family.

The *type* selects the semantics of communication; *protocol* selects specific rules / formats for this type. In practice, selecting the default protocol by specifying 0 is generally sufficient.

Sockets: Datagrams in the UNIX/LOCAL domain

Common *domains*:

Domain	**Description**
`PF_LOCAL`	local (previously UNIX) domain protocols
`PF_INET`	ARPA Internet protocols
`PF_INET6`	IPv6 protocols
...	see `socket(2)` / `sys/stat.h`

Common *types*:

Type	**Description**
`SOCK_STREAM`	sequenced, reliable, two-way connection based byte streams
`SOCK_DGRAM`	connectionless, unreliable messages of a fixed (typically small) maximum length
`SOCK_RAW`	access to internal network protocols and interfaces
...	see `socket(2)` / `sys/stat.h`

Sockets: Datagrams in the UNIX/LOCAL domain

Terminal — 130×24

```
            (struct sockaddr *)&name, sizeof(struct sockaddr_un)
) < 0) {
                perror("sending datagram message");
        }
        (void)close(sock);
        return EXIT_SUCCESS;
}
jschauma@apue$ cc -Wall -Werror -Wextra udgramsend.c -o send
jschauma@apue$ ./read
socket --> socket
--> The sea is calm tonight, the tide is full . . .
jschauma@apue$ ./read
socket --> socket
--> The sea is calm tonight, the tide is full . . .
jschauma@apue$ ./read
binding name to datagram socket: Address already in use
jschauma@apue$ ls -l socket
srwxr-xr-x  1 jschauma  users  0 Oct 25 20:29 socket
jschauma@apue$ rm socket
jschauma@apue$ ./read
socket --> socket
--> The sea is calm tonight, the tide is full . . .
jschauma@apue$
```

```
                perror("reading from socket");
        (void)printf("--> %s\n", buf);
        (void)close(sock);

        /* A UNIX domain datagram socket is a 'file'.  If you don
't unlink
         * it, it will remain in the file system. */
//      (void)unlink(NAME);
        return EXIT_SUCCESS;
}
"udgramread.c" 84L, 2896C written
jschauma@apue$ cc -Wall -Werror -Wextra udgramread.c -o read
jschauma@apue$ ./send socket
jschauma@apue$ ls -l socket
srwxr-xr-x  1 jschauma  users  0 Oct 25 20:29 socket
jschauma@apue$ ./send socket
sending datagram message: Connection refused
jschauma@apue$ ls -l socket
srwxr-xr-x  1 jschauma  users  0 Oct 25 20:30 socket
jschauma@apue$ ./send
usage: send <socket>: Undefined error: 0
jschauma@apue$ ./send socket
jschauma@apue$
```

0 sh | 1 sh

bind(2)

```
#include <sys/socket.h>

int bind(int s, const struct sockaddr *name, socklen_t namelen);
```

Returns: 0 if ok, -1 otherwise

bind(2) assigns a name to an unnamed socket.

Binding a name in the UNIX domain creates a socket in the filesystem. This file inherits permissions per the creating process's umask, but that is non-portable.

send(2) and recv(2)

```
#include <sys/socket.h>
ssize_t send(int s, const void *msg, size_t len, int flags);
ssize_t sendto(int s, const void *msg, size_t len, int flags,
                        const struct sockaddr *to, socklen_t tolen);


ssize_t recv(int s, const void *buf, size_t len, int flags);
ssize_t recvfrom(int s, void * restrict buf, size_t len, int flags,
                        struct sockaddr * restrict from, socklen_t fromlen);
```

Returns: number of bytes sent or received if ok, `-1` otherwise

Sockets: Datagrams in the UNIX/LOCAL domain

- create socket using socket(2)
- attach to a socket using bind(2)
- both processes need to agree on the name to use
- these files are only used for rendezvous, not for message delivery
- sockets are represented as file descriptors, so you can use `read(2)` and `write(2)`
- dedicated system calls like `recv(2)` and `send(2)` etc. offer specific functionality
- after communication, sockets must be removed using `unlink(2)`

Questions

- Change the program to become a generic "socket cat", a program that reads data from stdin and sends it into the specified socket, one line at a time.
- Experiment with the permissions on the socket after the server called `bind(2)`. Confirm or deny that they are honored on different operating systems as well as that binding the socket honors your `umask`.
- Change the programs to alternatively use `read(2)`/`write(2)` and `recv(2)`/`send(2)`.
- Can you have multiple processes using the same socket to send data to a single reader?
- Our example uses sockets of type `SOCK_DGRAM`; can we use `SOCK_STREAM` or any other type? What happens if the reader uses one type and the sender another?