CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

Sockets

Today's Topics

- New Way of Communicating Between Processes
- Sockets

"Standard" Unix Processes/IPC

- IPC stands for Inter-Process Communication in this context
- Until know, we did communication through:
  - File system: File descriptors, read()/write() and pipes
  - IPC construct is shared through normal process hierarchy inheritance rules, pipes created through pipe() are nameless
  - Totally reliable *byte stream* between producer and consumer
  - Ties in to conventional UNIX semantics of process creation and termination

New Ideas

- We want to create a generalization of the pipe construct for network-based I/O
- That means we still want file descriptors and read()/write() calls to work
- We need to take some extra features into consideration:
  - Network Protocol Stacks
  - Network Naming Conventions
  - Requirements of Protocol-Specific Message Passing
- The BSD and UNIX solution is the `socket()` call. Most concisely, it can be described as a *communication endpoint*.
- The call returns a file descriptor.
- `int socket(int domain, int type, int protocol);`

Communication Domain

- This basically specifies a *protocol stack*.
- Some systems contain a richer set of communication domains than others
  - `AF_UNIX` or `AF_LOCAL`: The UNIX IPC domain, local to a single machine
  - `AF_INET`: The Internet domain, global in scope
  - `AF_INET6`: The Internet domain, using IPv6
- Once a domain is specified, we know how to associate a name with the socket
- As well as knowing the semantics of supported IPC mechanisms

Unix Domain Sockets (in brief)

- Let's start with the simpler (but less interesting) case of the `AF_UNIX` communication domain
- The header file `<sys/un.h>` defines addresses

```
#define UNIX_PATH_MAX 108

struct sockaddr_un {
  unsigned short sun_family;   /* AF_UNIX */
  char sun_path[UNIX_PATH_MAX]; /* Pathname */
};
```

- Some examples of Unix domain sockets can be found under the `/dev` directory

Types of Sockets in Unix Domain

- We'll be looking at two *types* of sockets available in Unix
  - `SOCK_DGRAM` provides *datagram* communication semantics
  - Only promises best-effort delivery
  - Unix may discard datagrams in times of buffer congestion
  - Connectionless!

Types of Sockets in Unix Domain

- We'll be looking at two *types* of sockets available in Unix
  - `SOCK_STREAM` implements *virtual circuit* communication semantics
  - Reliable FIFO point-to-point communications
  - Appears as a byte stream to applications
  - This is actually how some later UNIX systems implement pipes!
- Socket type should be chosen according to the needs of the application, and should be programmed in accordance with well-specified delivery semantics of chosen type.

Operations on Sockets

- Binding a name to a socket:

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

- Sending datagram on a socket (asynchronous):

```
int sendto(int s, const void *msg, int len, unsigned int flags
        const struct sockaddr *to, int tolen);
```

- Receiving datagram from a socket (synchronous, blocking):

```
int recvfrom(int s, void *buf, int len, unsigned int flags,
        struct sockaddr *from, int *fromlen);
```

Server I

```
#include <errno.h>
#include <strings.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>

main() {
   short p_len;
   int socket_fd, cc, h_len, fsize, namelen;
   void printsun();
   struct sockaddr_un s_un, from;
   size_t addrlength;

   struct {
      char head;
      u_long body;
      char tail;
   } msg;

   socket_fd = socket (AF_UNIX, SOCK_DGRAM, 0);

   s_un.sun_family = AF_UNIX;
```

Server II

```
    strcpy(s_un.sun_path, "udgram");
    addrlength = sizeof(s_un.sun_family) + sizeof(s_un.sun_path);
    unlink("udgram");

    bind(socket_fd, (struct sockaddr *)&s_un, addrlength);

    for(;;) {
      fsize = sizeof(from);
      cc = recvfrom(socket_fd, &msg, sizeof(msg), 0,
        (struct sockaddr *)&from, &fsize);
      printsun(&from, "unix_rdgram: ", "Packet from");
      printf("Got data ::%c%ld%c\n", msg.head,msg.body,msg.tail);
      fflush(stdout);
    }
}

void printsun(Sun, s1, s2)
struct sockaddr_un *Sun; char *s1, *s2;
{
    printf("%s %s\n",s1,s2);
    printf("       family <%d> addr <%s>\n",
      Sun->sun_family, Sun->sun_path);
}
```

Client I

```
#include <errno.h>
#include <strings.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>

main() {
  int socket_fd, cc;
  //long getpid();
  struct sockaddr_un dest;

  struct {
    char head;
    u_long body;
    char tail;
  } msgbuf;

  socket_fd = socket (AF_UNIX, SOCK_DGRAM, 0);
  dest.sun_family = AF_UNIX;
  strcpy(dest.sun_path, "udgram");

  msgbuf.head = '<';
```

Client II

```
    msgbuf.body = (u_long) getpid();
    msgbuf.tail = '>';

    cc = sendto(socket_fd, &msgbuf, sizeof(msgbuf),0,
      (struct sockaddr *)&dest,sizeof(dest));
}
```

Sockets and the Internet (IPv4)

- `AF_INET` communication domain
- SOCK_DGRAM - Same as before! **UDP/IP**
- SOCK_STREAM - Same as before! **TCP/IP**
- We need a way to associate names with sockets to be able to do network I/O through a socket file descriptor

Sockets and the Internet (IPv4)

- The header file <netinet/in.h> defines a 32-bit for an Internet host.
- This actually identifies a specific network interface on a specific system on the Internet.
- It's represented by a 32 bit unsigned number

```
struct in_addr {
    __u32 s_addr;
}
```