CU CS 4995-03: Internet Systems Programming

Unix Interprocess Communications

Overview

- Unix sockets
- Unix pipes
- memory mapping and shared memory

already discussed other IPC: signals, network sockets

Unix sockets

- named, local, possibly with credentials

```
struct sockaddr_un {
  uint8_t     sun_len;
  sa_family_t sun_family;  /* AF_LOCAL or AF_UNIX */
  char        sun_path[104];
}
```

- don't use `sun` as variable name...

```
srwxr-xr-x 1 hgs faculty 0 Nov 15 15:12 /tmp/unixsocket=
```

`getsockname` and **`getpeername`**

```
int  getsockname(int s, struct sockaddr *name,
  socklen_t *namelen);
int getpeername(int s, struct sockaddr *name,
  socklen_t *namelen);
```

- for both Unix/local and network sockets
- `getsockname`: local socket name (e.g., port if set to 0)
- `getpeername`: address of party connected to socket

Unix sockets

- `AF_UNIX` $\longrightarrow$ `AF_LOCAL` (Posix)
- `SOCK_STREAM` and `SOCK_DGRAM`
- datagram clients must `bind()`
- no raw sockets
- server can't reuse existing socket ("address already in use")
- must `connect()` to *open* socket – not a mailbox
- can't send reply unless client socket is bound ($\neq$ network socket – no *ephemeral* ports!)
- `write()` blocks until client has read data

socketpair

```
#include <sys/socket.h>
int socketpair(int family, int type,
  int protocol, int sockfd[2]);
socketpair(AF_UNIX, SOCK_STREAM, 0, fd);
```

- *unnamed* connection
- bidirectional (“full-duplex”)

pipe

```
int pipe(int fd[2]);
```

- simplified version of `socketpair`
- BSD: half-duplex, Solaris: full duplex
- `fstat` returns number of bytes available for reading

popen

```
FILE *popen(const char *command, const char *mode);
pclose(f);
```

- pipe between calling process and command
- mode = ”r” or ”w”

FIFO: `mkfifo`

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *path, mode_t mode);
```

- roughly, pipe with implicit attributes

Passing file descriptors

- child sharing open file descriptor with parent
- keeping file descriptors across an `exec`
- but: want child to pass pack descriptor to parent
- Unix: generalized $\rightarrow$ from any process to any other
- uses `sendmsg`

Passing file descriptors

1. create Unix domain socket, datagram or stream
2. get descriptor: `open`, `pipe`, `mkfifo`, `socket`, `accept`, ...
3. create `msghdr` structure, use `msg_control`; increments reference count by one
4. receiver uses `recvmsg` to get descriptor – number may differ

SysV shared memory

- mapped into distinct per-process regions
- persists even after process dies
- deleted with `shmctl()` or `ipcrm`
- identified by "key" and "shared memory identifier"
- permissions bits similar to flags, but no execute permission
- similar: semaphores, message queues
- careful: shared segments can only be about 128 kB unless system parameter is changed

Create shared memory

Options:

1. private segment

   ```
   shmid = shmget(IPC_PRIVATE, size, SHM_R|SHM_W);
   shm = (char *)shmat(shmid, /*addr*/0, /*flag*/0);
   ```

   Need to convey *identifier* via file, socket, ...

2. named segment ⟶ pick some (integer) key

   ```
   shmid = shmget(key, size, SHM_R|SHM_W|IPC_CREAT);
   ```

   Share *key* with other users of segment.

Mapping IPC to a memory location

```
char *shm;
shmid = shmget(...);
shm = (char *)shmat(shmid, /*addr*/0, /*flag*/0);
```

Can also specify fixed address. Flag can be `SHM_RDONLY` or specify MMU properties.

Reference shared memory

- if created with `IPC_PRIVATE`, just
- if explicit key, use `shmget()`
- once done, call `shmdt(shmid)` to detach

SysV IPC: list active

```
$ ipcs
IPC status from <running system> as of
  Wed Nov 17 15:04:27 EST 1999
Message Queue facility inactive.
T         ID      KEY         MODE          OWNER    GROUP
Shared Memory:
m         11    0          --rw-------         hgs  faculty
m         12    0xb1       --rw-------         hgs  faculty
Semaphore facility inactive.
```

Remove via `ipcrm -m` *id*

Memory mapping

- map

```
void  *mmap(caddr_t addr, size_t len, int prot,
  int flags, int fildes, off_t off);
int munmap(caddr_t addr, size_t len);
```

addr: use this address; usually zero

prot: protection: `PROT_READ`, `PROT_WRITE`, `PROT_EXEC`;

flags: `MAP_SHARED` or `MAP_PRIVATE` (copy on write; e.g., for debugger)

fildes: file descriptor for file to be mapped

off: offset into the file

Using /dev/zero for shared memory

- `/dev/zero` = infinite source of zeros when read
- map to memory
- share via normal `fork()` semantics

Message queues

- linked lists of messages in a “queue”
- `msgget(key, flag)` creates queue or opens existing one
- `msgsnd()` appends messages
- messages: type field, length, data
- get messages: `msgrcv()`
- can get messages by type

```
struct msg {
  long mtype;      /* message type */
  char mtext[0];   /* message body */
}
int msgsnd(int msqid, void *ptr, size_t nbytes, int flag);
int msgrcv(int msqid, void *ptr, size_t nbytes,
  long type, int flag);
```

Other shared memory

Shared memory as a *file* rather than memory:

```
#include <sys/mman.h>
int shm_open(const char *name, int oflag, mode_t mode);
```

Link with `-lrt`)

SysV IPC issues

- system wide
- no reference count: not deleted upon process exit
- not part of file system
- can't use multiplexed I/O (`select`, `poll`), e.g., wait for two message queues

IPC summary

IPC	CL?	reliable?	flow?	records	msg. types?
msg. queues	no	yes	yes	yes	yes
streams	no	yes	yes	yes	yes
Unix stream socket	no	yes	yes	no	no
Unix dgram socket	yes	yes	no	yes	no
FIFOs	no	yes	yes	no	no