Operating Systems

CSE 231

Instructor: Sambuddho Chakravarty

(Semester: Monsoon 2020)

Week 5: Oct 5 – Oct 8

IPC Using Message Passing

- Very similar to FIFOs.
- Relies on message queues (again FIFO).

- Acquire message queue ID

```
 int msgget(key_t key, int msgflg);
```

IPC Using Message Passing

- Associate the message queue to an ``key’’ much like other IPC mechanisms.

```
#include <sys/msg.h>
key = ftok("/home/beej/somefile", 'b');
msqid = msgget(key, 0666 | IPC_CREAT);
```

IPC Using Message Passing

- Message buffer type (defined in sys/msg.h)

```
struct msgbuf
{ long mtype;
char mtext[1]; };
```

Problem: mtext only single byte!

IPC Using Message Passing

- You can use any structure as long as the `mtype’ is type `long’.

```
struct pirate_msgbuf
{ long mtype; /* must be positive */
    struct pirate_info {
     char name[30];
    char ship_type;
     ..} info;
};
```

IPC Using Message Passing

- Sending the message:

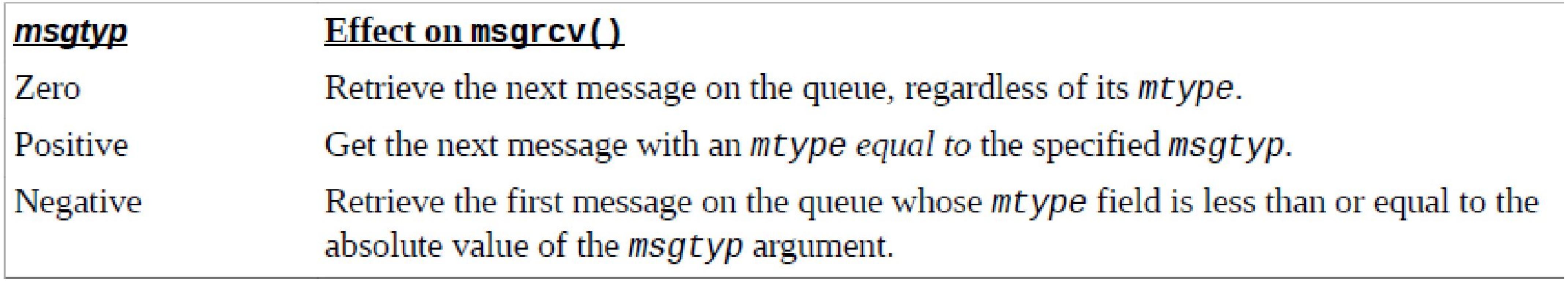
```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int
msgflg);
```

- Receiving the message:

```
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
int msgflg);
```

IPC Using Message Passing

- Msgtype used to determine which message to receive from the queue.



msgtyp	**Effect on `msgrcv()`**
Zero	Retrieve the next message on the queue, regardless of its *mtype*.
Positive	Get the next message with an *mtype equal to* the specified *msgtyp*.
Negative	Retrieve the first message on the queue whose *mtype* field is less than or equal to the absolute value of the *msgtyp* argument.

IPC Using Message Passing

- Delete the message queue:

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

```
#include <sys/msg.h>
.
.
 .
msgctl(msqid, IPC_RMID, NULL);
```

IPC Using Unix Domain Sockets

- Domain sockets: Much like INET sockets used for communication between clients and servers.
- Two-way FIFOs.
- Socket structure

```
struct sockaddr_un
{
unsigned short sun_family; /* AF_UNIX */
char sun_path[108];
}
```

IPC Using Unix Domain Sockets

- Steps to create a server socket:

socket() system call.

```
unsigned int s, s2;
struct sockaddr_un local, remote;
int len;
s = socket(AF_UNIX, SOCK_STREAM, 0);
```

IPC Using Unix Domain Sockets

- Steps to create a server socket:

bind() system call – associates a ``socket address’’ (a file path) to the said socket (much like what was done using shared memory)

```
local.sun_family = AF_UNIX; /* local is declared before socket() ^ */
strcpy(local.sun_path, "/home/beej/mysocket");
unlink(local.sun_path);
len = strlen(local.sun_path) + sizeof(local.sun_family);
bind(s, (struct sockaddr *)&local, len)
```

IPC Using Unix Domain Sockets

- Steps to create a server socket:

listen() system call – Set the socket ``state’’ to waiting.

```
listen(s, 5);
```

IPC Using Unix Domain Sockets

- Steps to create a server socket:

accept() system call – Accept incoming connections from a client. Returns a connected socket descriptor (different from the previously created socket descriptor)

```
len = sizeof(struct sockaddr_un);
s2 = accept(s, &remote, &len); /* s2 is the newly connected
socket descriptor which the server uses to communicate to
the client*/
```

IPC Using Unix Domain Sockets

- Steps to create a server socket:

send() or recv() to the connected socket descriptor.

```
while (len = recv(s2, &buf, 100, 0), len > 0)
        send(s2, &buf, len, 0);



/* loop back to accept() from here */
```

IPC Using Unix Domain Sockets

- Steps to create a server socket:

When done with the communication, you close() or shutdown() the socket

close(s);

Or

shutdown(s,how);
how == (SHUT_RD || SHUT_WR || SHUT_RDWR)

IPC Using Unix Domain Sockets

- Client part of the communication

- Initiate a client socket()

```
int s;
struct sockaddr_un remote;

if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1)
 {     perror("socket"); exit(1);
 }
```

IPC Using Unix Domain Sockets

- Connect() to the server

```
#define SOCK_PATH “/path/to/mysocket”

struct sockaddr_un remote;

remote.sun_family = AF_UNIX;
strcpy(remote.sun_path, SOCK_PATH);
 len = strlen(remote.sun_path) + sizeof(remote.sun_family);
if (connect(s, (struct sockaddr *)&remote, len) == -1)
{ perror("connect"); exit(1);
}
```

IPC Using Unix Domain Sockets

- send() / recv() to/from the socket just like you do for the server socket

Socketpair – Full-duplex pipe (socket + pipe)

```
int main(void)
{ int sv[2]; /* the pair of socket descriptors */
char buf; /* for data exchange between processes */
 if (socketpair(AF_UNIX, SOCK_STREAM, 0, sv) == -1)
{ perror("socketpair"); exit(1); }
if (!fork())
{  read(sv[1], &buf, 1);
printf("child: read '%c'\n", buf);
buf = toupper(buf);  /* make it uppercase */
write(sv[1], &buf, 1);
printf("child: sent '%c'\n", buf); }
else { /* parent */ write(sv[0], "b", 1);
printf("parent: sent 'b'\n");
 read(sv[0], &buf, 1); p
printf("parent: read '%c'\n", buf);
wait(NULL); /* wait for child to die */ }
return 0;}
```