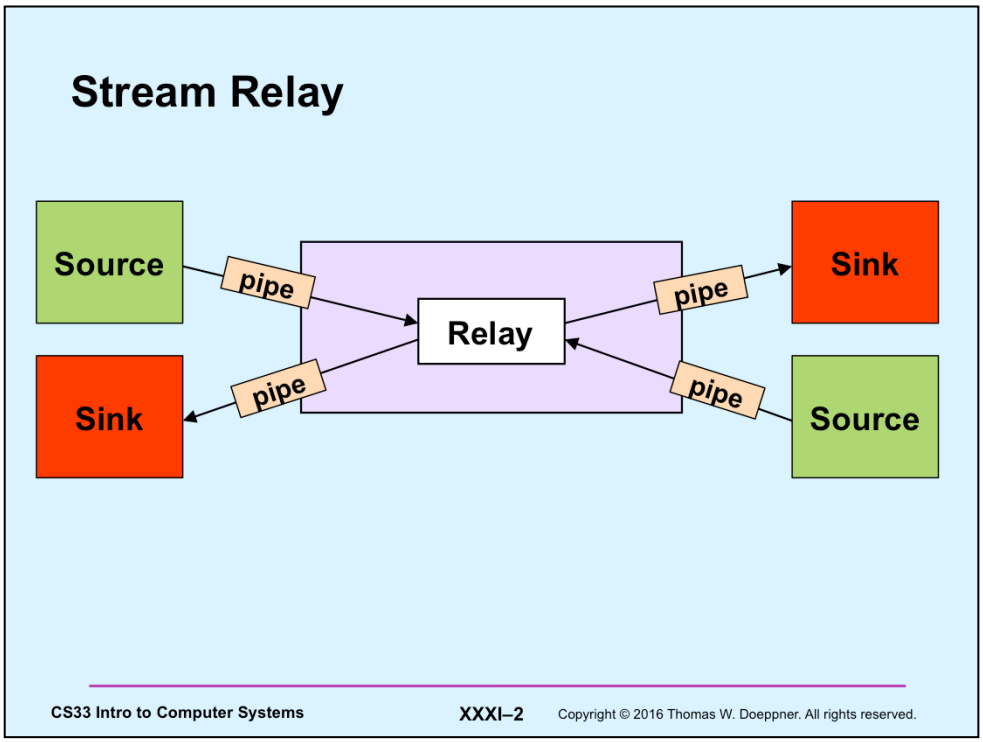
CS 33

More Network Programming



We start by looking at what's known as *event-based programming*: we write code that responds to events coming from a number of sources. As a simple example, before we use the approach in a networking example, we exam a simple relay: we want to write a program that takes data, via a pipe, from the left source and sends it, via a pipe, to the right sink. At the same time it takes data from the right source and sends it to the left sink.

Solution?

```
while(…) {
   size = read(left, buf, sizeof(buf));
   write(right, buf, size);
   size = read(right, buf, sizeof(buf));
   write(left, buf, size);
}
```


This solution is probably not what we'd want, since it strictly alternates between processing the data stream in one direction and then the other.

Select System Call

```
int select(
  int nfds,          // size of fd_sets
  fd_set *readfds,   // descriptors of interest
                     // for reading
  fd_set *writefds,  // descriptors of interest
                     // for writing
  fd_set *excpfds,   // descriptors of interest
                     // for exceptional events
  struct timeval *timeout
                     // max time to wait
);
```


Relay Sketch

```
void relay(int left, int right) {
   fd_set rd, wr;
   int maxFD = max(left, right) + 1;
   FD_ZERO(&rd); FD_SET(left, &rd); FD_SET(right, &rd);
   FD_ZERO(&wr); FD_SET(left, &wr); FD_SET(right, &wr);
   while (1) {
      select(maxFD, &rd, &wr, 0, 0);
      if (FD_ISSET(left, &rd))
         read(left, bufLR, BSIZE);
      if (FD_ISSET(right, &rd))
         read(right, bufRL, BSIZE);
      if (FD_ISSET(right, &wr))
         write(right, bufLR, BSIZE);
      if (FD_ISSET(left, &rd))
         write(left, bufRL, BSIZE);
   }
}
```


Here a simplified version of a program to handle the relay problem using *select*. An *fd_set* is a data type that represents a set of file descriptors. FD_ZERO, FD_SET, and FD_ISSET are macros for working with fd_sets; the first makes such a set represent the null set, the second sets a particular file descriptor to included in the set, the last checks to see if a particular file descriptor is included in the set.

Quiz 1

40 bytes have been read from the left-hand source. Select reports that it is ok to write to the right-hand sink.

a) You're guaranteed you can immediately write all 40 bytes to the right-hand sink

b) All that's guaranteed is that you can immediately write at least one byte to the right-hand sink

c) Nothing is guaranteed

Relay (1)

```
void relay(int left, int right) {
  fd_set rd, wr;
  int left_read = 1, right_write = 0;
  int right_read = 1, left_write = 0;
  int sizeLR, sizeRL, wret;
  char bufLR[BSIZE], bufRL[BSIZE];
  char *bufpR, *bufpL;
  int maxFD = max(left, right) + 1;

  // set up file descriptors so they won't
  // wait if I/O is not yet possible
  fcntl(left, F_SETFL, O_NONBLOCK);
  fcntl(right, F_SETFL, O_NONBLOCK);
```


This and the next three slides give a more complete version of the relay program.

The calls to fcntl are used to set up the file descriptors so if, for example, write is called but there currently is no room for all the data, the call won't wait until there is room, but will return the number of bytes that were able to be written, or if none were written, will report an error (EWOULDBLOCK);

Initially our program is prepared to read from either the left or the right side, but it's not prepared to write, since it doesn't have anything to write.

Relay (2)

```
while(1) {
  FD_ZERO(&rd);
  FD_ZERO(&wr);
  if (left_read)
    FD_SET(left, &rd);
  if (right_read)
    FD_SET(right, &rd);
  if (left_write)
    FD_SET(left, &wr);
  if (right_write)
    FD_SET(right, &wr);

  select(maxFD, &rd, &wr, 0, 0);
```


We set up the fd_sets *rd* and *wr* to indicate what we are interested in reading from and writing to (initially we have no interest in writing, but are interested in reading from either side).

Relay (3)

```
if (FD_ISSET(left, &rd)) {
  sizeLR = read(left, bufLR, BSIZE);
  left_read = 0;
  right_write = 1;
  bufpR = bufLR;
}
if (FD_ISSET(right, &rd)) {
  sizeRL = read(right, bufRL, BSIZE);
  right_read = 0;
  left_write = 1;
  bufpL = bufRL;
}
```


If there is something to read from the left side, we read it. Having read it, we're temporarily not interested in reading anything further from the left side, but now want to write to the right side.

In a similar fashion, if there is something to read from the right side, we read it.

Relay (4)

```
    if (FD_ISSET(right, &wr)) {
      if ((wret = write(right, bufpR, sizeLR)) == sizeLR) {
        left_read = 1; right_write = 0;
      } else {
        sizeLR -= wret; bufpR += wret;
      }
    }
    if (FD_ISSET(left, &wr)) {
      if ((wret = write(left, bufpL, sizeRL)) == sizeRL) {
        right_read = 1; left_write = 0;
      } else {
        sizeRL -= wret; bufpL += wret;
      }
    }
  }
  return 0;
}
```


Writing is a bit more complicated, since the outgoing pipe might not have room for everything we have to write, but just some of it. Thus we must pay attention to what write returns. If everything has been written, then we can go back to reading from the other side, but if not, we continue trying to write.