ISA 563: Fundamentals of Systems Programming

Inter-process Communication

April 3, 2012

Inter-process Communication (IPC)

- IPC is used to pass data among processes
- Different mechanisms for different levels of communication:
    - Between related processes
    - Between processes inside the same host
    - Between processes inside different hosts connection through network
- Some IPC mechanisms may require synchronization

IPC Mechanisms

- Shared files
- Pipes
- FIFOs
- Message queues
- Shared memory
- Sockets:
  - Local (Unix domain sockets)
  - Remote (TCP/UDP)
- Remote procedure calls

Persistence of IPC Objects

- process-persistent IPC:
  - Exists until last process with IPC object closes the object
- kernel-persistent IPC:
  - Exists until reboots or is explicitly deleted
- filesystem-persistent IPC:
  - Exists until IPC object is explicitly deleted

Pipes

- Pipes provide a communication mechanism between related processes (parent/child relationship)
  - Child inherits file descripters to communicate with parent
- Pipes can be accessed using normal file system functions:
  - `read()`
  - `write()`

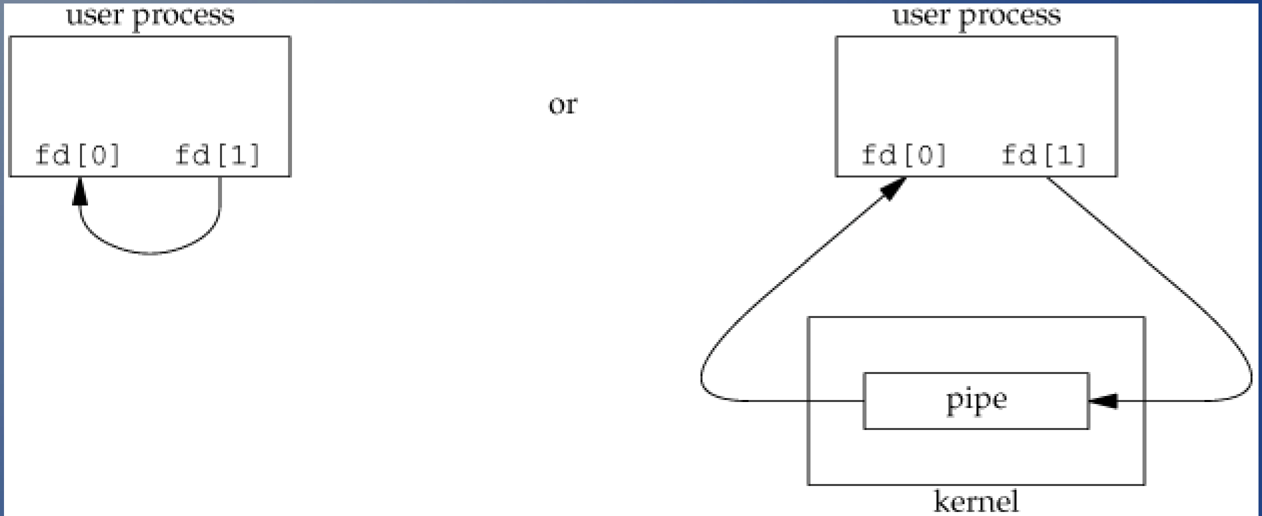
pipe()

```
#include <unistd.h>

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

- Two file descriptors are returned:
  - fd[0] – opened for reading
  - fd[1] – opened for writing

pipe() (cont'd)

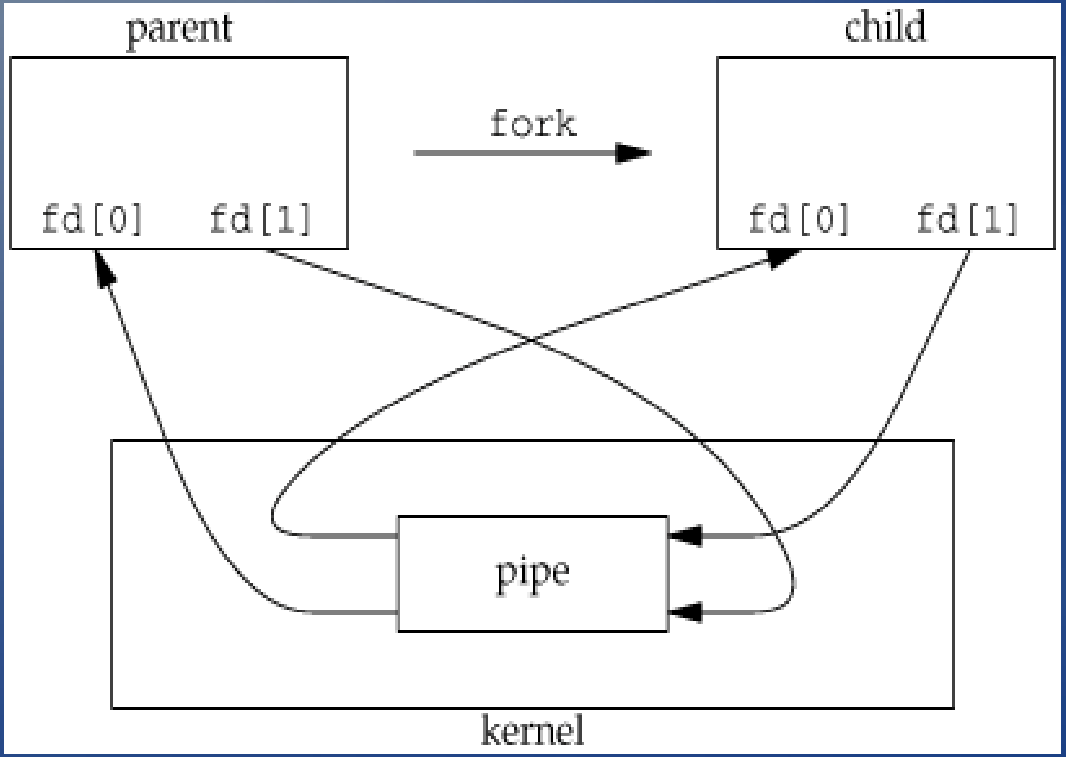
- View inside a single process:



(Figure Courtesy of Advanced Programming in the Unix Environment)

pipe() (cont'd)

- What happens when process forks after calling pipe(int fieldes[2])?

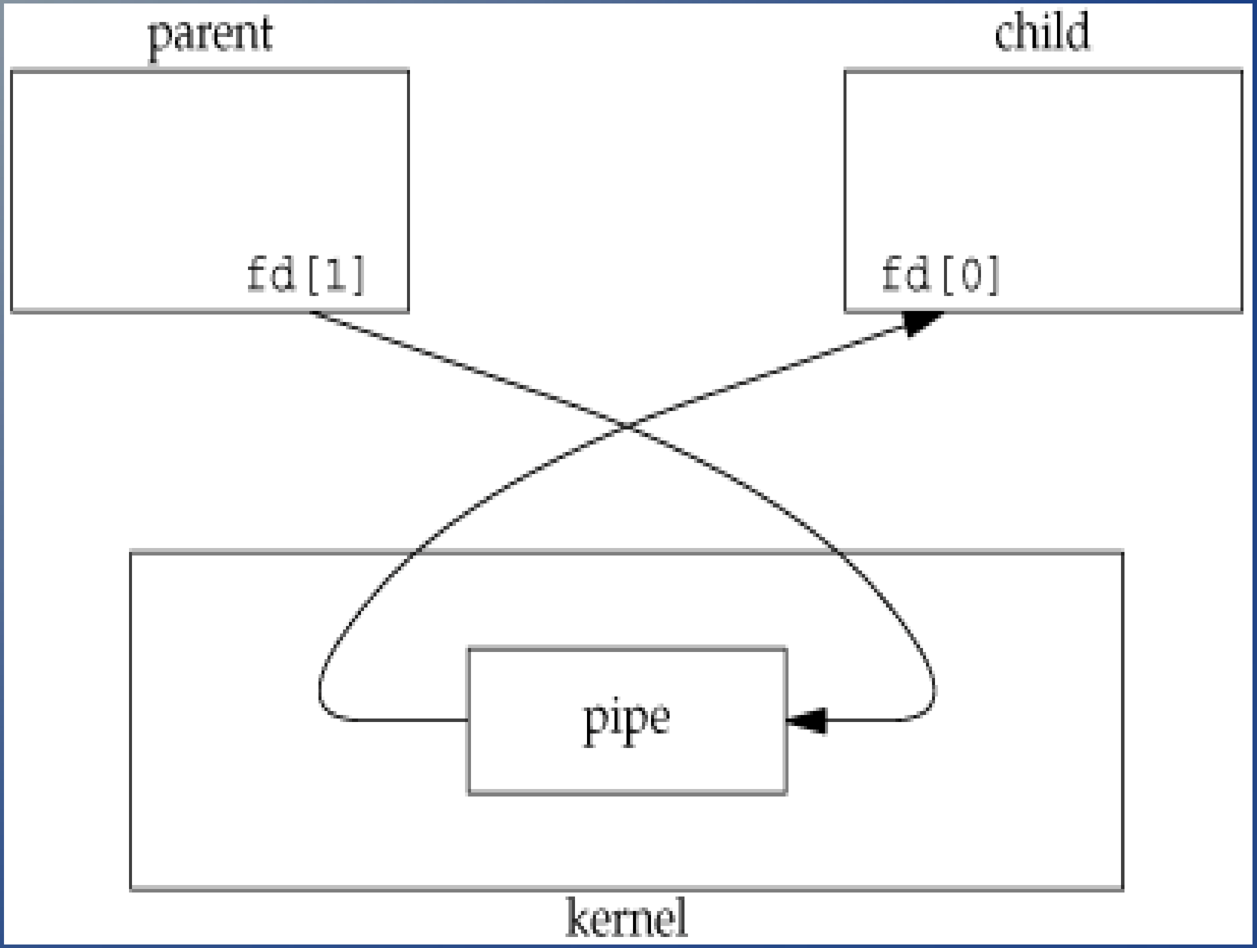


(Figure Courtesy of Advanced Programming in the Unix Environment)

Half-duplex Communication using Pipes

- Parent close one file descriptor and child closes the other depending desired direction of data flow:
    - parent → child:
        - parent closes fd[0]
        - child closes fd[1]
    - child → parent
        - parent closes fd[1]
        - child closes fd[0]

Parent → Child Half-duplex



(Figure Courtesy of Advanced Programming in the Unix Environment)

Demo

hello_pipe.c

Demo

pager.c

FIFOs

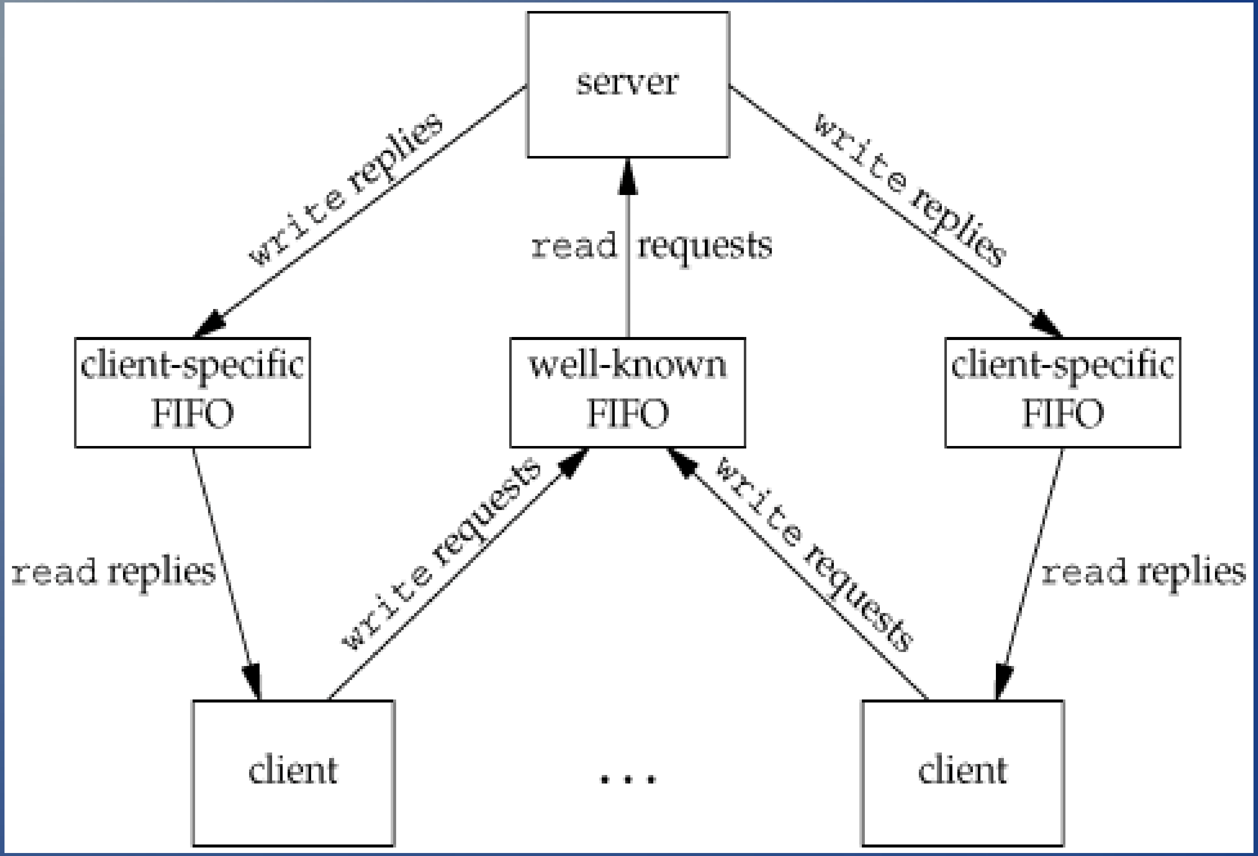
- FIFOs: first in, first out queues
  - Addresses pipe's limitations – allows two unrelated processes to communicate on the same host
  - Visible inside file system
- Common uses:
  - Used by shell to pass data from one process to another (through shell pipelines)
  - Used as rendezvous point between clients and servers

mkfifo

```
// mkfifo (3) system call

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

```
$ # mkfifo (1) command
$ mkfifo fifo1

$ cat fifo1

$ yes “hello” > fifo1   # in another terminal
```

FIFOs in Client/Server Interaction



(Figure Courtesy of Advanced Programming in the Unix Environment)

Message Queues

- Linked list of messages stored within the kernel

- APIs:
  - msgget – open an existing queue or create one
  - msgsnd – add a message to message queue
  - msgget – retrieve a message from message queue

Shared Memory

- Two or more processes share a piece of memory in user space
- No kernel involvement
- Fastest form of IPC available
- Read/write access has to be synchronized

Semaphores

- A protected variable used to controlling access to shared resources
- Similar to mutexes, but can have integer values associated:
    - process calls sem_wait:
        - if semaphore value is larger than 0, decrease value and return immediatelly
        - if semaphore value is 0, block until value is larger than 0
    - process call sem_post:
        - increase semaphore value and return immediately
- Can have “binary” and “counting” semaphores

Demo

shmem.c