

Unix Pipes (CSE 422S)

Ken Wong
Washington University

kenw@wustl.edu
www.arl.wustl.edu/~kenw

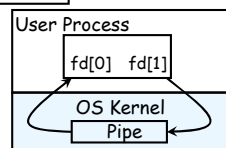
Topics

- Interprocess Communication (IPC) with named and unnamed pipes
 - » Unbuffered I/O system calls (*open*, *read*, *write*, *close*)
 - » *unlink* system call
- Pipes (Half Duplex)
 - » *pipe()*
- FIFOs (Named Pipes)
 - » *mkfifo* system call and *mkfifo* command
- Examples
 - » Simple FIFO
 - » Simple pipe
 - » Structured messages

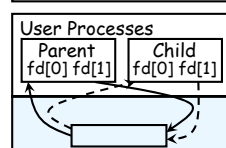
In A Nutshell

Unnamed Pipe

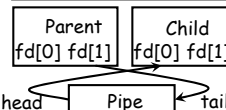
pipe



fork

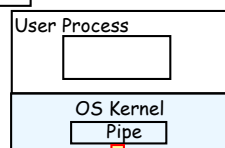


close, close

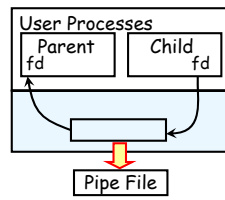


Named Pipe

mkfifo



open, open



Pipe (Unnamed FIFO) in the Shell

The Script 'xxx'

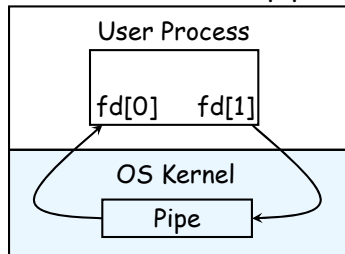
```
y | z & # y, z are shell scripts which just sleep
sleep 20
wait $! # $! is PID of most recently
        # backgrounded process ("/bin/sh ./z")
```

Output of 'ps -lx' while 'xxx' is sleeping

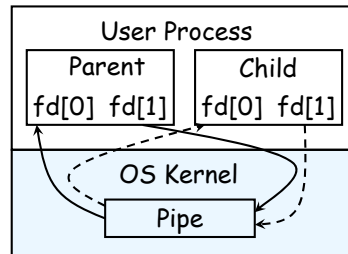
UID	PID	PPID	TIME	COMMAND
1024	17273	17271	0:03	-tcsh
1024	17648	17273	0:00	/bin/sh ./xxx
1024	17649	17648	0:00	/bin/sh ./z
1024	17650	17648	0:00	/bin/sh ./y
1024	17651	17650	0:00	sleep 20
1024	17652	17648	0:00	sleep 20
1024	17653	17649	0:00	sleep 20

Unix Pipe Movie

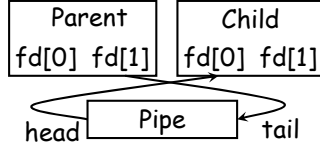
■ Parent creates pipe



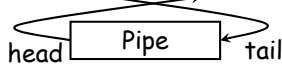
■ Parent forks child



■ Parent closes head



■ Child closes tail



5 - Ken Wong, 2/7/2007

Washington University in St. Louis

Unix Pipe Example 1 (1)

```

... Requisite includes and "unistd.h" ...
int main(void) {
    ... Variable definitions ...
    Pipe(fd);
    pid = Fork( );
    if (pid > 0) {           // parent
        close(fd[0]);       // close head of pipe
        Write (fd[1], msg, msgsz); // write msg to pipe
    } else {                // child
        close(fd[1]);       // close tail of pipe
        n = Read (fd[0], line, msgsz); // read msg from pipe
        printf("%s\n", line); // write msg to stdout
    }
    return 0;
}

```

6 - Ken Wong, 2/7/2007

Washington University in St. Louis

Unix Pipe Example 1 (2)

■ Variable Definitions

```

int      n, fd[2];
pid_t    pid;
char     line[MAXLINE];
char     *msg = "hello\n";
const int msgsz = sizeof(msg);

```

■ stdinc.h

» int Pipe(int fd)

- Error if pipe() returns negative

» ssize_t Write(int fd, const void *buf, size_t n)

- If write() returns negative: Error occurred and errno is set
 - Call perror() to display human-readable error message
- If write() doesn't return n: Error (unusual)

» ssize_t Read(int fd, const void *buf, size_t n)

- If read() returns negative: Error occurred and errno is set
- If read() returns 0: EOF (pipe was closed)
- If read() returns less than n: OK

7 - Ken Wong, 2/7/2007

Washington University in St. Louis

Unix Pipes

■ Limitations

- » They are half-duplex (data flows in one direction)
- » Can only be used between 2 processes that have a common ancestor
 - Process A creates a pipe P
 - Process A calls 'fork' to create child B
 - A pipe is used between processes A and B (i.e., "A | B")

■ Synopsis

- » #include <unistd.h>
- » int pipe (int FileDescriptor[2]);
- » Returns 0 if OK; -1 if error
- » FileDescriptor[0] is open for reading
- » FileDescriptor[1] is open for writing

8 - Ken Wong, 2/7/2007

Washington University in St. Louis

Pipe Operation

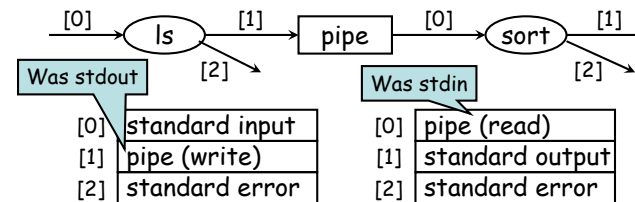
- Reading from a pipe whose write end has been closed
 - » 'read' returns 0 to indicate end of file (EOF) after all data has been read
- Writing to a pipe whose read end has been closed
 - » Generates a SIGPIPE signal
 - » 'write' returns an error with 'errno' set to EPIPE if we ignore the signal or catch it and return from the signal handler (See errno(3C) or perror(3C))
- The constant 'PIPE_BUF' specifies kernel's pipe buffer size
 - » Interleaving of messages from multiple writers will occur if we try to send more than 'PIPE_BUF' bytes in a message
- 'write' is atomic if the size is less than 'PIPE_BUF' bytes
- There are NO message boundaries
 - » 'n = read(fd, buff, 100)' will attempt to read 100 bytes and could return less than 100 (which might be OK)

9 - Ken Wong, 2/7/2007

Washington University in St. Louis

Pipelines

- Example (sort by non-decreasing file size)
 - » ls -l | sort -n +4



File Descriptor Tables

- Needed:
 - » Create pipe
 - » Replace STDIN/STDOUT with end of a pipe

10 - Ken Wong, 2/7/2007

Washington University in St. Louis

Duplicating a File Descriptor

```
#include <unistd.h>
int dup2 (int oldfd, int newfd);
```

- dup2 makes newfd be the copy of oldfd, closing newfd first if necessary
 - » Return new descriptor or -1 if error
- Example

```
int    fd[2];
Pipe(fd);    cpid = Fork();
if (cpid == 0) {
    Dup2(fd[1], STDOUT_FILENO);    Close(fd[0]);    Close(fd[1]);
    ... exec ls ...
}
Dup2(fd[0], STDIN_FILENO);    Close(fd[0]);    Close(fd[1]);
... exec sort ...
```

11 - Ken Wong, 2/7/2007

Washington University in St. Louis

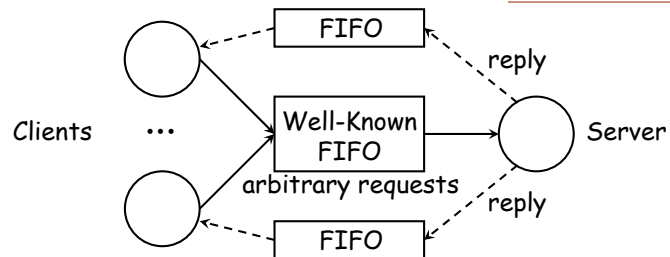
Pipeline Implementations

- Implementing "A | B | C" ???
 - » Different from one shell to another
- Method 1 (Forward forking)
 - » xssh forks child A which forks child B which forks child C
 - » A is child of xssh and rest are grandchildren
- Method 2 (Reverse forking):
 - » xssh forks child C which forks child B which forks child A
 - » C is child of xssh and rest are grandchildren
- Method 3
 - » xssh forks all children
 - » All processes are children of xssh

12 - Ken Wong, 2/7/2007

Washington University in St. Louis

FIFO Example (Client/Server)



- Client processes write requests (messages) into a FIFO with well-known pathname
- Server process reads requests in FIFO (First-In-First-Out) order
 - » Write replies into reply FIFO for sending process

13 - Ken Wong, 2/7/2007

Washington University in St. Louis

Unix FIFO (Named Pipe)

- A FIFO allows unrelated processes to exchange data
 - » A FIFO is a type of Unix file
 - » Creating a FIFO is similar to creating a file
- Synopsis
 - » `#include <sys/types.h>`
 - » `#include <sys/stat.h>`
 - » `int mkfifo(const char *PathName, mode_t Mode);`
- Equivalent Unix command

<ul style="list-style-type: none"> » <code>mkfifo myFifo -m 644</code> » <code>echo "hello" > myFifo &</code> » <code>cat myFifo</code> 	<ul style="list-style-type: none"> # make the FIFO # write to the FIFO # read entire FIFO
---	--

14 - Ken Wong, 2/7/2007

Washington University in St. Louis

mkfifo and File Modes

- Specifies access permissions of the file
 - » `S_IRUSR` user-read (i.e., 0400 or `r-- --- ---`)
 - » `S_IWUSR` user-write (i.e., 0200 or `-w- --- ---`)
 - » `S_IXUSR` user-execute (i.e., 0100 or `--x --- ---`)
 - » `S_[RWX]GRP` group-read, -write, -execute
 - » `S_[RWX]OTH` other-read, -write, -execute
- Usage
 - » `rc = mkfifo(pipe0, S_IRUSR | S_IRGRP);`
 - User+group read
 - » `rc = mkfifo(pipe0, 0440); // same`
 - » `rc = mkfifo(pipe0, S_IRUSR | S_IWGRP);`
 - User read+write (0400 or `rw- --- ---`)

15 - Ken Wong, 2/7/2007

Washington University in St. Louis

FIFO Operations

- Normal file I/O functions can be used on FIFO
 - » `FileDescriptor = open(FIFOname, Options)`
 - » `n = write(FileDescriptor, Buffer, Nbytes)`
 - » `n = read(FileDescriptor, Buffer, Nbytes)`
 - » `rc = close(FileDescriptor)`
 - » `rc = unlink(FIFOname)`
- Default Operation
 - » A read-only open will block until some other process opens the FIFO for writing
 - » A write-only open will block until some other process opens the FIFO for reading
 - » The nonblocking flag (`O_NONBLOCK`) affects what happens in the above two cases

16 - Ken Wong, 2/7/2007

Washington University in St. Louis

Example 2 (FIFO Reader)



```
int fd; // file descriptor of FIFO
char *rqFifo = "requests"; // name of FIFO
int n; // #bytes read from FIFO
int N; // input buffer
const int msgSz = sizeof(int);

Mkfifo(rqFifo, S_IRUSR | S_IWUSR);
fd = Open(rqFifo, O_RDONLY);
n = Read(fd, &N, msgSz);
while (n == msgSz) { // until EOF
    ... compute ...
    n = Read(fd, &N, msgSz);
}
Unlink(requestFifo);
```

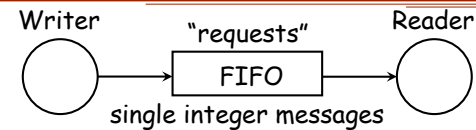
rw- - - - -
or O600

read-only

17 -Ken Wong, 2/7/2007

Washington University in St. Louis

Example 2 (FIFO Writer)



```
int fd; // file descriptor of FIFO
char *rqFifo = "requests"; // name of FIFO
int data; // output buffer
const int msgSz = sizeof(int);

fd = Open(rqFifo, O_WRONLY);
for (int i=0; i<M; i++) {
    ... generate data ...
    Write(fd, &data, msgSz);
}
Close(fd);
```

18 -Ken Wong, 2/7/2007

Washington University in St. Louis

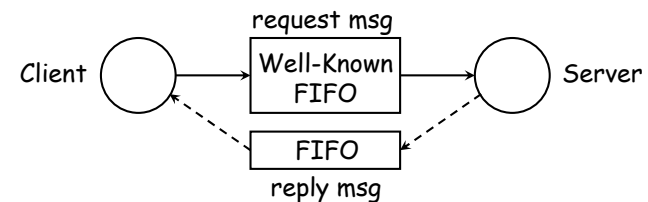
System Call Wrappers

- **Mkfifo (make a FIFO)**
 - » if (mkfifo(rqFifo, S_IRUSR | S_IWUSR) < 0) ... Error ...
- **Open (open a file)**
 - » if ((fd = open(rqFifo, O_RDONLY)) < 0) ... Error ...
 - » else return fd;
- **Read (read from a file)**
 - » if ((n = read(fd, &N, msgSz)) < 0) ... Error ...
 - » else return n;
- **Unlink (remove a file)**
 - » if (unlink(rqFifo) < 0) ... Error ...

19 -Ken Wong, 2/7/2007

Washington University in St. Louis

Example 3 (FIFO Messages)



- **Message Structures**
 - » Option 1: ASCII character stream separated by newline
 - » Option 2: (header, msg body)
 - Simple Header: Message length (Implied body format)
 - Complex Header: Message length, Body format
- **Who should create the FIFOs?**

20 -Ken Wong, 2/7/2007

Washington University in St. Louis

Option 2 (Simple Header, Body)

```
typedef struct request { // variable length
    int    bodySz; // n*sizeof(int), n=0..100
    int    op;
    int    operand[100]; // maximum
} request_t;
typedef struct reply { // constant length
    int    bodySz; // sizeof(int)
    int    value;
} reply_t
```

Static allocation of headers

```
request_t rq; // request allocation
reply_t rp; // reply allocation
```

Dynamic allocation of headers

```
requestHdr_t *rq; // request ptr
replyHdr_t *rp; // reply ptr
rqh = Malloc(sizeof(request_t));
rph = Malloc(sizeof(reply_t));
```

21 -Ken Wong, 2/7/2007

Washington University in St. Louis

Send/Recv Message

Sender

Note Wrappers

```
request_t requestMsg; // over allocates
reply_t replyMsg;
... Construct message ...
requestMsg.bodySz = ... something ...
Write(outfd, &requestMsg, sizeof(requestMsg));
rc = Read(infd, &replyMsg, sizeof(replyMsg));
```

Receiver

```
request_t requestMsg; // over allocates
reply_t replyMsg;
Read(infd, &requestMsg.bodySz,
    sizeof(requestMsg.bodySz)); // header
Read(infd, &requestMsg.op, requestMsg.bodySz); // body
... Handle request; Construct reply ...
Write(outfd, &replyMsg, sizeof(replyMsg));
```

22 -Ken Wong, 2/7/2007

Washington University in St. Louis

Comments

- What if EOF occurs on a FIFO?
- Server should create request FIFO
 - » Clients can't do anything until the request FIFO has been created
 - » Well-known means everyone knows the pathname of the FIFO
 - » Server opens for reading; Client opens for writing
- Client should create reply FIFO
 - » Must be done before server tries to write the reply
 - » Client opens for reading; Server opens for writing
- Can deadlock occur in the FIFO creation process?

23 -Ken Wong, 2/7/2007

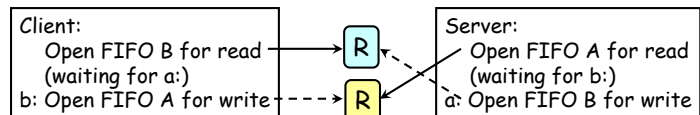
Washington University in St. Louis

Open FIFO is a Rendezvous

- Open one end for reading and one end for writing
- Open *blocks* until there is atleast one reader and one writer



Deadlock can occur



24 -Ken Wong, 2/7/2007

Washington University in St. Louis