

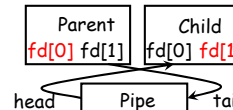
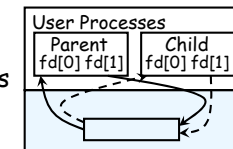
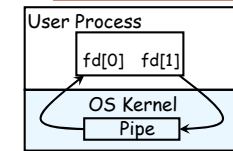
Pipes (Unnamed FIFO) (CSE 422S)

Ken Wong
Washington University

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

Pipe In A Nutshell

- pipe(fd)
 - » create pipe structure in kernel
 - » return 2 file descriptors (FDs)
 - one for each end of the pipe
 - non-negative integers (array indices)
 - » can insert bytes into either end
- fork()
 - » parent and child have same pipe FDs
- close(fd) in parent AND child
 - » close extra pipe ends →
 - remove ptr from FD table
 - can't use FD
 - » convention:
 - write to fd[1]; read from fd[0]



Unix Pipe Example 1 (1)

```
... include "unistd.h" and other headers ...
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;
}
```

Unix Pipe Example 1 (2)

- Variable Definitions
- | | |
|-----------|--------------------------|
| int | n, fd[2]; |
| pid_t | pid; |
| char | line[MAXLINE]; |
| char | *msg = "hello\n"; |
| const int | msgsz = 1 + sizeof(msg); |
- stdinc.h wrappers
 - » 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: might be OK

Unix Pipes

■ Limitations

- » 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")
- » Older implementations: Unidirectional, not bidirectional

■ Synopsis

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

5 - Ken Wong, 9/27/2007

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 as 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)

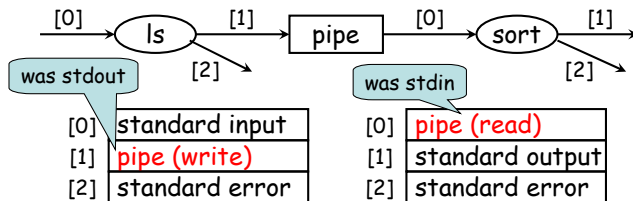
6 - Ken Wong, 9/27/2007

Washington University in St. Louis

Pipelines

■ Example (sort by non-decreasing file size)

- » ls -l | sort -n +4



File Descriptor Tables

■ Need to:

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

7 - Ken Wong, 9/27/2007

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

NOTE both

■ 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 ...
```

8 - Ken Wong, 9/27/2007

Washington University in St. Louis

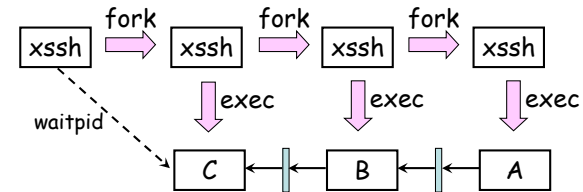
Pipeline Implementations

- 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
 - » **INCORRECT**
- 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 and 3'
 - » xssh forks all children
 - » All processes are children of xssh

9 - Ken Wong, 9/27/2007

Washington University in St. Louis

Method 2 (Reverse Forking)

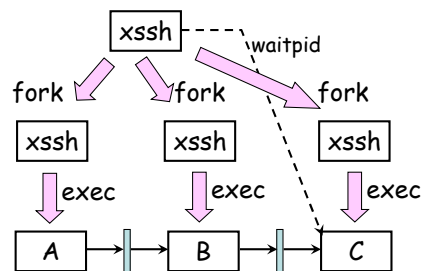


root process doesn't know about A and B

10 - Ken Wong, 9/27/2007

Washington University in St. Louis

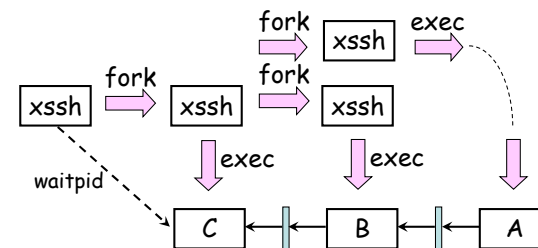
Method 3



11 - Ken Wong, 9/27/2007

Washington University in St. Louis

Method 3'



12 - Ken Wong, 9/27/2007

Washington University in St. Louis

Pipeline in the Shell (1)

■ 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 (edited) of 'ps -l' while 'xxx' is sleeping

UID	PID	PPID	TIME	COMMAND
1024	17273	17271	0:03	-tssh
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

13 - Ken Wong, 9/27/2007

Washington University in St. Louis

Pipeline in the Shell (2)

■ Different OS

■ Output of 'ps -o uid,pid,ppid,pgrp,cmd' while 'xxx' is sleeping

UID	PID	PPID	PGRP	CMD
1024	25467	25466	25467	-csh -i
1024	25548	25467	25548	/bin/sh xxx
1024	25549	25548	25548	/bin/sh ./y
1024	25551	25548	25548	/bin/sh ./z
1024	25552	25548	25548	sleep 20
1024	25554	25549	25548	sleep 20
1024	25555	25551	25548	sleep 20
1024	25558	25467	25558	ps -o uid,pid,ppid,pgrp,cmd

process group

14 - Ken Wong, 9/27/2007

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

15 - Ken Wong, 9/27/2007

Washington University in St. Louis