The Design Of The
Unix Operating system

Apurba Sarkar

Department of Computer Science and Tech.
Bengal Engg. and Sc. University Shibpur.

May 16, 2012

What is pipe?

- A pipe is used for one-way communication of a stream of bytes.
- The command to create a pipe is `pipe()`, which takes an array of two integers.
- It fills in the array with two file descriptors that can be used for low-level I/O. The following code shows how to crate a pipe,

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

What is pipe? contd

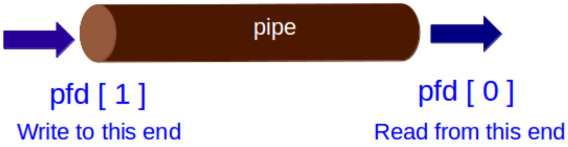

Figure 1: A pipe

I/O with pipe

- File descriptors created by pipe system call can be used for block I/O.
- Following code snippet shows how to write to and read from a pipe. `write(pfd[1], buf, SIZE);`
  `read(pfd[0], buf, SIZE);`

pipe and fork

- A single process would not use a pipe.
- They are used when two processes wish to communicate in a one-way fashion.
- A process splits in two using fork().
- A pipe opened before the fork becomes shared between the two processes.

I/O with pipe

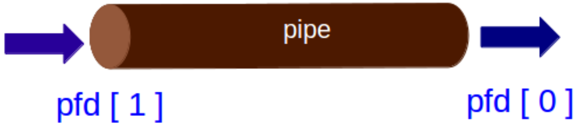

Figure 2: pipe before fork()

pipe and fork contd

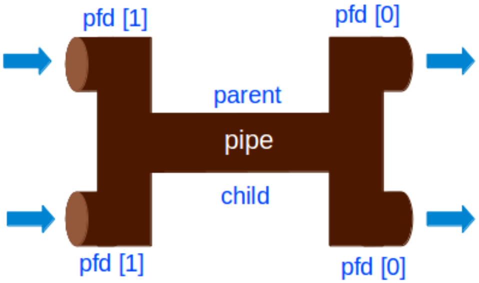

Figure 3: pipe after fork()

pipe and fork contd

- This gives two read ends and two write ends.
- The read end of the pipe will not be closed until both of the read ends are closed, and the write end will not be closed until both the write ends are closed.
- Either process can write into the pipe, and either can read from it.Which process will get what is not known.
- For predictable behaviour, one of the processes must close its read end, and the other must close its write end to make it a simple pipeline again.

pipe and fork contd

- Suppose the parent wants to write down a pipeline to a child.
- The parent closes its read end, and writes into the other end.
- The child closes its write end and reads from the other end.
- When the processes have ceased communication, the parent closes its write end.
- This means that the child gets eof on its next read, and it can close its read end. This is shown in the next slide.

pipe and fork contd

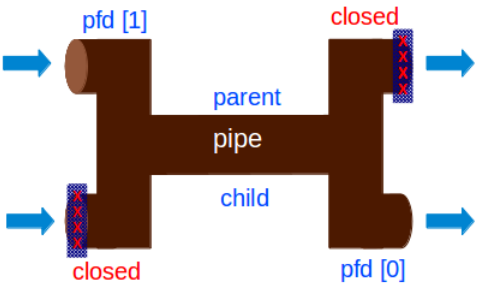

Figure 4: simple pipe line

An example

```
#include <stdio.h>
#define SIZE 1024
int main(int argc, char **argv)
{
  int pfd[2],nread,pid;
  char buf[SIZE];
  if (pipe(pfd) == -1){
  perror("pipe failed");
    exit(1);
  }
  if ((pid = fork()) < 0){
    perror("fork failed");
    exit(2);
  }
```

An example

```
  if (pid == 0) /* child */
  {close(pfd[1]);
    while ((nread = read(pfd[0], buf, SIZE))!= 0)
      printf("child read %s\n", buf);
      close(pfd[0]);
  } else {
    close(pfd[0]); /* parent */
    strcpy(buf, "hello...");
    /* include null terminator in write */
    write(pfd[1], buf,strlen(buf)+1);
    close(pfd[1]);
  }
  exit(0);
}
```

dup system call

- A pipeline works because the two processes know the file descriptor of each end of the pipe.
- Each process has a stdin (0), a stdout (1) and a stderr (2).
- The file descriptors will depend on which other files have been opened, but could be 3 and 4, say.
- If one of the processes replaces itself by an “exec”. The new process will have files for descriptors 0, 1, 2, 3 and 4 open.

How will it know which are the ones belonging to the pipe?

pipe A real example "ls|wc"

- To implement "ls | wc" the shell will have created a pipe and then forked.
- The parent will exec to be replaced by "ls", and the child will exec to be replaced by "wc"
- The write end of the pipe may be descriptor 3 and the read end may be descriptor 4.
- "ls" normally writes to 1 and "wc" normally reads from 0.

pipe A real example "ls|wc"

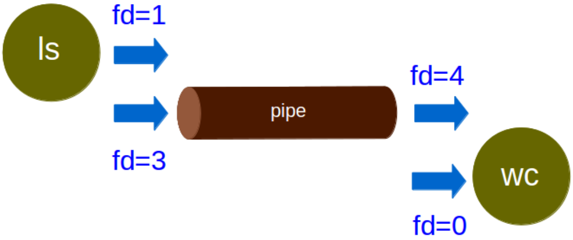

Figure 5: Before dup system call

pipe A real example "ls|wc"

- The dup2() system call takes an existing file descriptor, and another one that it "would like to be".
- Here, fd=3 would also like to be 1, and fd=4 would like to be 0.
- So we dup2 fd=3 as 1, and dup2 fd=4 as 0. Then the old fd=3 and fd=4 can be closed as they are no longer needed.

pipe A real example “ls|wc”

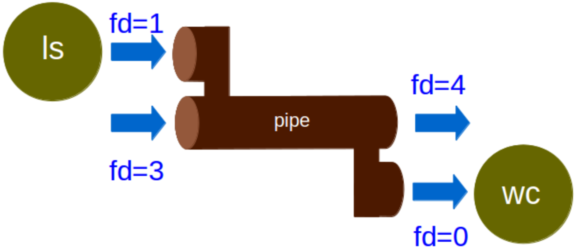

Figure 6: After dup system call

pipe A real example “ls|wc”

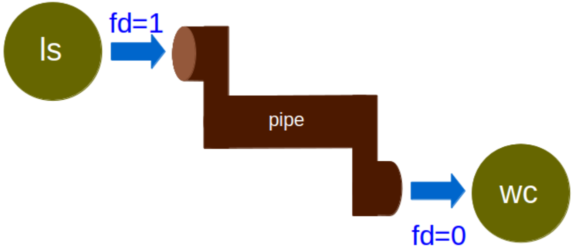

Figure 7: After close system call

pipe and fork example

Without any error checks, the program to do this is

```
int main(void)
{ int pfd[2],pipe(pfd);
  if (fork() == 0) {
    close(pfd[1]);
    dup2(pfd[0], 0);
    close(pfd[0]);
    execlp("wc", "wc",(char *) 0);
  } else {
    close(pfd[0]);
    dup2(pfd[1], 1);
    close(pfd[1]);
    execlp("ls", "ls",(char *) 0);
  }
}
```

Summery

- pipe system call is always followed by a fork system call.
- As an effect of the `fork` the ends of the pipe are shared.
- We generally close one read end and one write end of the pipe to create a one way channel.
- pipe can be used for one way communication between two processes only when they have parent-child relationship.
- A pipeline may consist of three or more process (such as a C version of `ps | sed 1d | wc -l`).
- A process may want to both write to and read from a child. In this case it creates two pipes.

Why Share memory?

Why Share memory?

- pipe is used for communication between two related process.

Why Share memory?

- pipe is used for communication between two related process.
- What if the they are not related and still want to communicate?

Why Share memory?

- pipe is used for communication between two related process.
- What if the they are not related and still want to communicate?
- shared memory comes handy in such situation.

Systeam calls needed

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key_t key, int size, int shmflg);

char *shmat(int shmid, char *shmaddr, int shmflg);

int shmdt(char *shmaddr);
```

shmget()

- It creates or locates a piece of shared memory.
- The pieces are identified by a key.
- The key is an integer agreed upon by all the communicating processes.
- It also specifies the size of the memory and the permission flags (eg -rwx——).