系統程式設計
Systems Programming

鄭卜壬教授
臺灣大學資訊工程系

Tei-Wei Kuo, Chi-Sheng Shih, and Hao-Hua Chu©2007
Department of Computer Science and Information Engineering
Graduate institute of Multimedia and Networking, National Taiwan University

Contents

1. OS Concept & Intro. to UNIX
2. UNIX History, Standardization & Implementation
3. File I/O
4. Standard I/O Library
5. Files and Directories
6. System Data Files and Information
7. Environment of a Unix Process
8. Process Control
9. Signals
10. Inter-process Communication
11. Thread Programming
12. Networking

Interprocess Communication

- Except for communicating via the file system, processes can communicate via various IPCs. (Ch15)

IPC type	POSIX.1	XPG3	V7	SVR2	SVR3.2	SVR4	4.3BSD	4.3+BSD
pipes (half duplex)	●	●	●	●	●	●	●	●
FIFOs	●	●		●	●	●		
Stream pipes (full duplex)					●	●	●	●
named stream pipes					●	●	●	●
message queues		●		●	●	●		
semaphores		●		●	●	●		
shared memory		●		●	●	●		
sockets						●	●	●
streams					●	●		

Pipes (Ch15.2)

- **Pipes have two limitations:**
  - They are half-duplex.
  - They can be used between processes that have a common ancestor.
- **Stream pipes are duplex (Ch17.2)**
- **Named pipes/FIFOs and named stream pipes (Ch17.2) can be used between process not having the same ancestor.**
- **`pipe` function**

```
#include <unistd.h>
int pipe( int filedes[2] );
```

Unix Pipes

fd[0]: open for reading, file type is FIFO
fd[1]: open for writing, file type is FIFO

Pipes between Processes

Half-duplex pipe after a **`fork`**

Useful Pipe

Pipe from parent process to child process

Read from/Write to Pipes

- **When the write end is closed, after all the data has been read, reading from a pipe returns 0 (end of file)**
- **When the read end of a pipe is closed, writing to a pipe triggers `SIGPIPE` signal (Ch10)**
- **Kernel's pipe buffer size:**
  - Constant PIPE_BUF specifies the pipe buffer size.
  - Writing PIPE_BUF or less will not be interleaved.

Creating a pipe from the parent to the child and send data down the pipe

```
int
main(void)
{
    int     n;
    int     fd[2];
    pid_t   pid;
    char    line[MAXLINE];

    if (pipe(fd) < 0)
        err_sys("pipe error");
    if ((pid = fork()) < 0) {
        err_sys("fork error");
    } else if (pid > 0) {       /* parent */
        close(fd[0]);
        write(fd[1], "hello world\n", 12);
    } else {                    /* child */
        close(fd[1]);
        n = read(fd[0], line, MAXLINE);
        write(STDOUT_FILENO, line, n);
    }
    exit(0);
}
```

Connecting Pipes to Standard I/O

Pipe from parent process to child process

```
#define DEF_PAGER   "/bin/more"     /* default pager program */

int
main(int argc, char *argv[])
{
    int     n;
    int     fd[2];
    pid_t   pid;
    char    *pager, *argv0;
    char    line[MAXLINE];
    FILE    *fp;

    if (argc != 2)
        err_quit("usage: a.out <pathname>");

    if ((fp = fopen(argv[1], "r")) == NULL)
        err_sys("can't open %s", argv[1]);
    if (pipe(fd) < 0)
        err_sys("pipe error");

    if ((pid = fork()) < 0) {
        err_sys("fork error");
    } else if (pid > 0) {                           /* parent */
        close(fd[0]);       /* close read end */

        /* parent copies argv[1] to pipe */
        while (fgets(line, MAXLINE, fp) != NULL) {
            n = strlen(line);
            if (write(fd[1], line, n) != n)
                err_sys("write error to pipe");
        }
        if (ferror(fp))
            err_sys("fgets error");

        close(fd[1]);   /* close write end of pipe for reader */

        if (waitpid(pid, NULL, 0) < 0)
            err_sys("waitpid error");
        exit(0);
    } else {                                        /* child */
```

```
    } else {                                        /* child */
        close(fd[1]);   /* close write end */
        if (fd[0] != STDIN_FILENO) {
            if (dup2(fd[0], STDIN_FILENO) != STDIN_FILENO)
                err_sys("dup2 error to stdin");
            close(fd[0]);   /* don't need this after dup2 */
        }

        /* get arguments for execl() */
        if ((pager = getenv("PAGER")) == NULL)
            pager = DEF_PAGER;
        if ((argv0 = strrchr(pager, '/')) != NULL)
            argv0++;        /* step past rightmost slash */
        else
            argv0 = pager;  /* no slash in pager */

        if (execl(pager, argv0, (char *)0) < 0)
            err_sys("execl error for %s", pager);
    }
    exit(0);
}
```

Using Pipes for Parent-child Synchronization

Avoid Race Condition (Ch8)

```
static void charatatime(char *);

int main(void)
{
        pid_t pid;

+       TELL_WAIT();

        if ( (pid = fork()) < 0 ) return -1;
        else if ( pid == 0 ) {
+               WAIT_PARENT();          // parent gets first
                charatatime("output from child\n");
        } else {
                charatatime("output from parent\n");
+               TELL_CHILD( pid );
        }
        return 0;
}

static void charatatime(char *str)
{
        char *ptr;
        for ( ptr = str;  *ptr!='\0'; ptr++ )
                write( 1, ptr, 1 );
}
```

```
static int  pfd1[2], pfd2[2];

void
TELL_WAIT(void)
{
    if (pipe(pfd1) < 0 || pipe(pfd2) < 0)
        err_sys("pipe error");
}

void
TELL_PARENT(pid_t pid)
{
    if (write(pfd2[1], "c", 1) != 1)
        err_sys("write error");
}

void
WAIT_PARENT(void)
{
    char    c;

    if (read(pfd1[0], &c, 1) != 1)
        err_sys("read error");

    if (c != 'p')
        err_quit("WAIT_PARENT: incorrect data");
}
```

```
void
TELL_CHILD(pid_t pid)
{
    if (write(pfd1[1], "p", 1) != 1)
        err_sys("write error");
}

void
WAIT_CHILD(void)
{
    char    c;

    if (read(pfd2[0], &c, 1) != 1)
        err_sys("read error");

    if (c != 'c')
        err_quit("WAIT_CHILD: incorrect data");
}
```

FIFOs (Ch15.5)

- **FIFOs – named pipes for (un-)related processes to exchange data.**
  - A file type – st_mode (S_ISFIFO macro)
  - Data in a FIFO removed when the last referencing process terminates.

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

  - mode and usr/grp ownership = those of a file
  - Op's: open, close, read, write, unlink, etc.

FIFOs

- **O_NONBLOCK**
  - NO → an open for read-only blocks until some other process opens the FIFO for writing (write-only as well).
  - Yes → an open for read-only always returns, while that for write-only returns with an error (errno=ENXIO) if there is no reader.
- **Write for a no-reader FIFO → SIGPIPE**
  - Closing of the last writer → EOF
- **Atomic writes**
  - PIPE_BUF

FIFOs

tee: copies stdin to stdout, making a copy in files

- **How to pipe the standard output of a process to more than one file/process?**
- **Example – Stream Duplication**

```
mkfifo fifo1
prog3 < fifo1 &
prog1 < infile | tee fifo1 | prog2
```

FIFOs

- **Example – Client-Server Communication**
  - One server & multiple clients
    - Issues: replies, SIGPIPE, removing of all clients

Clientserver communication using FIFOs

Programming Assignment 2

- **Multiple Processes**
- **Inter-process communication**
- **Goal**
  - Process Control
  - Pipe