Lab 2-1. Interprocess Communication - Pipes

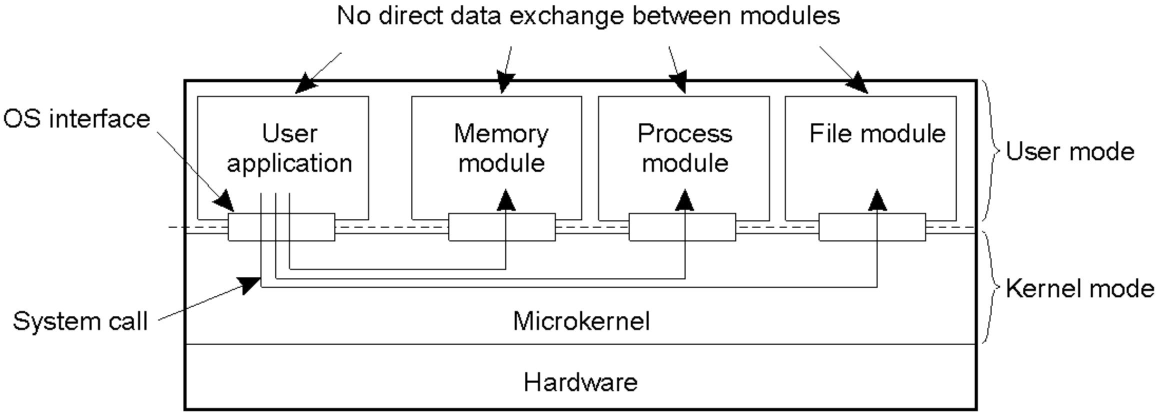
Objectives

- Classify the Interprocess Communication functions.
- Understand the structure of pipe, and implement the example of server-client program.

Background

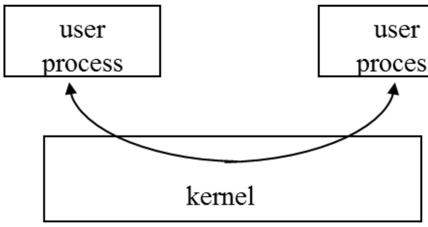
1. Uniprocessor Operating System and Communication

   Separate applications from operating system code through a microkernel.

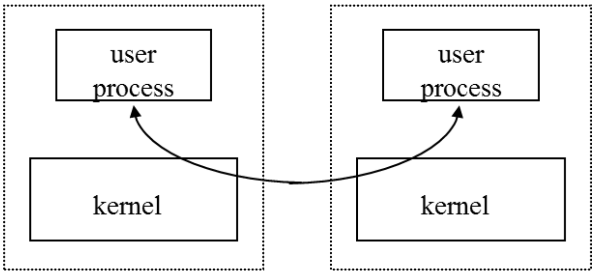


2. Interprocess communication has 2 forms:

□ IPC between Two Processes at the same system



□ IPC between Two Systems



IPC is used for 2 functions:

1) **Synchronization:** Used to coordinate access to resources among processes and also to oordinate the execution of these processes. They are

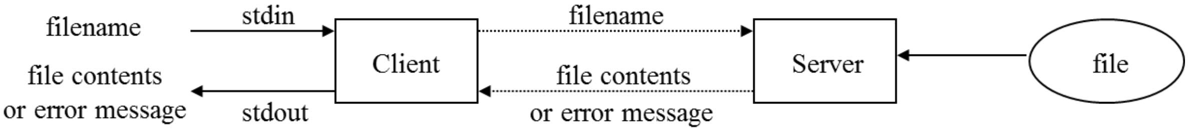
- Record locking,
- Semaphores,
- Mutexes and Condition variables.

2) **Message Passing:** Used when processes wish to exchange information. Message passing take several forms such as :

- Pipes,
- FIFOs,
- Message Queues,
- Shared Memory.

3. **Atomic operation:** one or more operations that are treated as a single operation. No other operation can be executed between the start and end of an atomic operation.
4. Simple Client-Server or IPC model:

□ Example of Client-Server Model



5. Pipe
   1) pipe

```
int pipe (int *filedes)
```

   - provides a one-way flow of data.
   - is created by the pipe system call.
   - Two file descriptors are returned
     - filedes[0] which is open for reading
     - filedes[1] which is open for writing

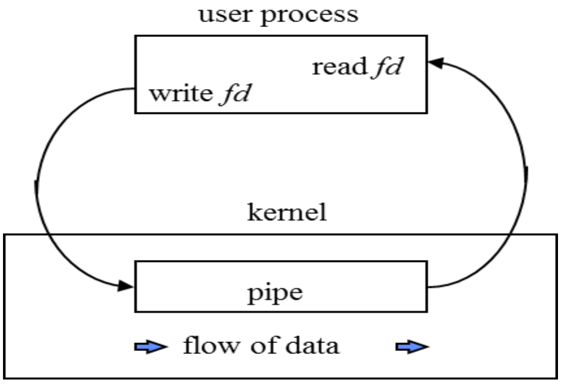
   2) Disadvantage of Pipe
   - only use for child processes created from same parent process

6. FIFO (First In, First Out)
   1) FIFO

```
int * mknod(char *parent, int mode, int *dev);
```

   - Similar to a pipe
   - But, FIFO has a name associated with it, allowing unrelated processes to access a single FIFO
   - Uni-directional data flow
   - Named pipe
     - One FIFO used for multiple individual processes
     - Available for System V
     - Generated by System call mknod
   - Created by int mknod(char *pathname, int mode, int dev)
     - pathname : a normal Unix pathname
     - mode : file access mode (read & write permissions for owner, group, world)
     - dev: ignore FIFO operation

Practice



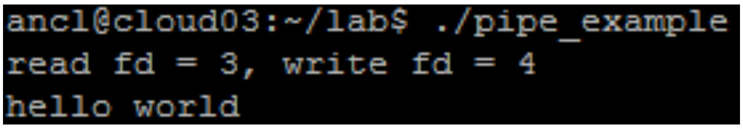
Pipes provide a one-way flow of data and are created by the pipe system call. Two file descriptors are returned: filedes[0] which is open for reading, filedes[1] which is open for writing.

```
int pipe(int *filedes);
```

Example to show how to create and use a pipe:

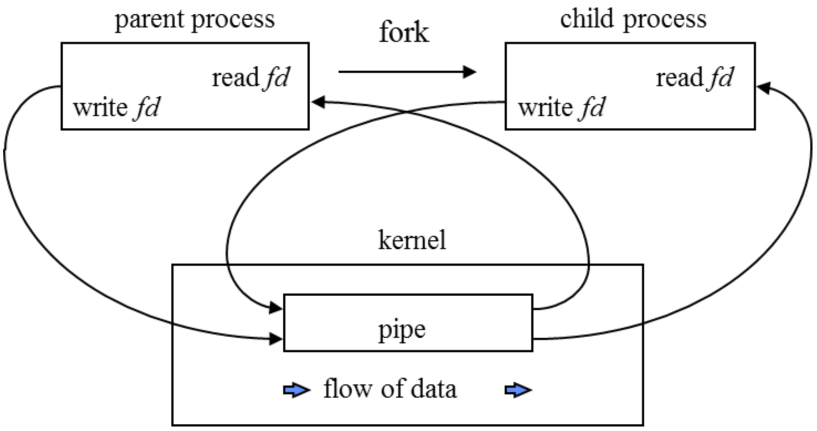
```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
        int pipefd[2], n;
        char buf[100];
        if(pipe(pipefd) < 0)
                exit(1);
        printf("read fd = %d, write fd = %d\n", pipefd[0], pipefd[1]);

        if(write(pipefd[1], "hello world\n", 12) != 12)
                exit(1);

        if((n = read(pipefd[0], buf, sizeof(buf))) <= 0)
                exit(1);
        write(1, buf, n);

        return 0;
}
```

Result:

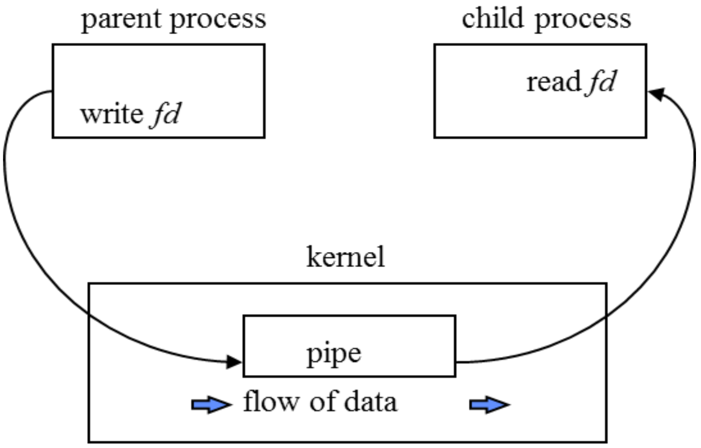


```
ancl@cloud03:~/lab$ ./pipe_example
read fd = 3, write fd = 4
hello world
```

Pipe in a single process, immediately after fork:



Pipes between two processes: unidirectuional

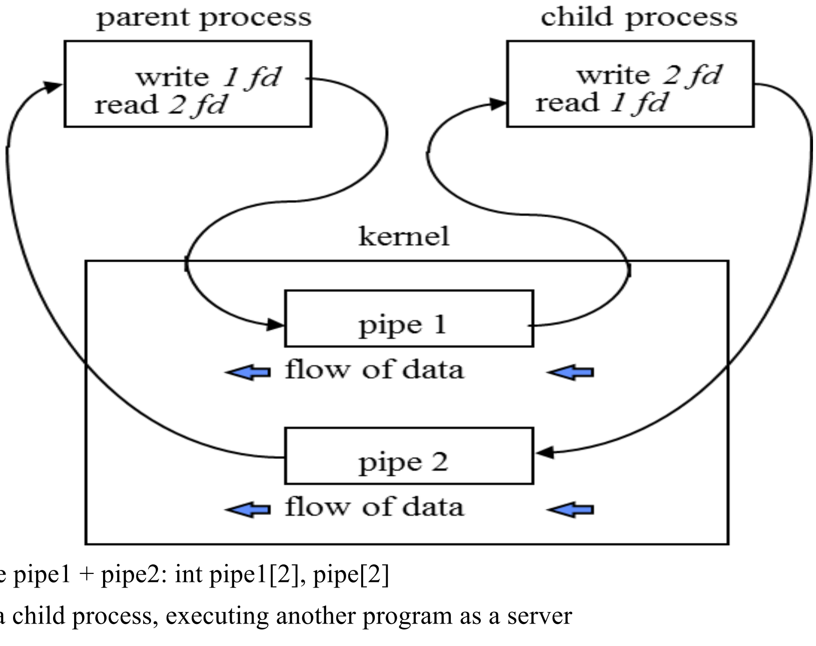


Steps: 1) Opening the pipe

2) Forking off another process

3) Closing the appropriate pipes on each end

Pipes between two processes: bidirectional



Steps: 1) Create pipe1 + pipe2: int pipe1[2], pipe[2]

2) Forking off a child process, executing another program as a server

3) Parent closes read end of pipe1 + write end of pipe2

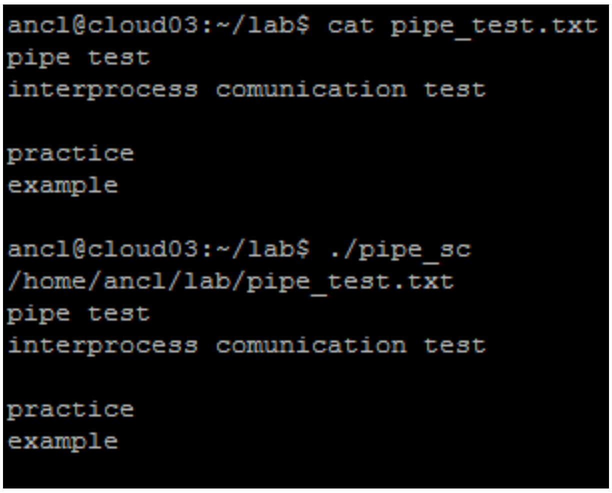
4) Child closes write end of pipe1 + read end of pipe2

Example of Simple Client-Server using PIPE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXBUF 1024

void client(int readfd, int writefd) {
    char buf[MAXBUF];
    int n;
    if(fgets(buf, MAXBUF, stdin) == NULL)
        exit(1);
    n = strlen(buf);
    if(buf[n-1] == '\n')
        --n;
    if(write(writefd, buf, n) != n)
        exit(1);
    while((n = read(readfd, buf, MAXBUF)) > 0) {
        if(write(1, buf, n) != n)
            exit(1);
    }
    if(n < 0)
        exit(1);
}

void server(int readfd, int writefd) {
    char buf[MAXBUF];
    int n, fd;

    if((n = read(readfd, buf, MAXBUF)) <= 0)
        exit(1);
    buf[n] = '\0';
    if((fd = open(buf, 0)) < 0) {
        sprintf(buf, "can't open file\n");
        n = strlen(buf);
        if(write(writefd, buf, n) != n)
            exit(1);
    }
    else {
        while((n = read(fd, buf, MAXBUF)) > 0) {
            if(write(writefd, buf, n) != n)
                exit(1);
        }
        if(n < 0)
            exit(1);
    }
}

int main(void) {
    int childpid, pipe1[2], pipe2[2];

    if(pipe(pipe1) < 0 || pipe(pipe2) < 0)
        exit(1);

    if((childpid = fork()) < 0)
        exit(1);
```

```
            else if(childpid > 0) {
                        close(pipe1[0]);
                        close(pipe2[1]);
                        client(pipe2[0], pipe1[1]);
                        while(wait((int*)0) != childpid);
                        close(pipe1[1]);
                        close(pipe2[0]);
            }
            else {
                        close(pipe1[1]);
                        close(pipe2[0]);
                        server(pipe1[0], pipe2[1]);
                        close(pipe1[0]);
                        close(pipe2[1]);
            }

            return 0;
}
```

Result:



```
ancl@cloud03:~/lab$ cat pipe_test.txt
pipe test
interprocess comunication test

practice
example

ancl@cloud03:~/lab$ ./pipe_sc
/home/ancl/lab/pipe_test.txt
pipe test
interprocess comunication test

practice
example
```