# 进程间通信（IPC，Inter Process Comunication）

## 一、管道（使用最简单）

### 1、本质

调用pipe函数可以创建一个管道，管道实质上是**内核中的一块缓冲区**（默认为4k，可由ulimit –a命令查看），有读、写两个文件描述符引用，使用**环形队列机制**，数据从管道的写端流入，从读端流出。

### 2、管道的局限性

1）只能用于有血缘关系进程之间的通信；

2）数据一旦被读走，便不在管道中存在，不可反复读取。

### 3、测试

首先父进程先写入让子进程读，然后子进程写入让父进程读。

**#include <stdio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**int main()**

**{**

**int fd[2];**

**int ret = pipe(fd); //创建管道**

**if (ret == -1)**

**{**

**perror("pipe error");**

**exit(1);**

**}**

**ret = fork(); //创建子进程**

**if (ret == -1)**

**{**

**perror("fork error");**

**exit(1);**

**}**

**else if (ret == 0) //子进程**

**{**

**usleep(1); //确保父进程先写**

**char buffer[1024];**

**ret = read(fd[0], buffer, sizeof(buffer)); //子进程 读**

**if (ret == -1)**

**{**

**perror("read error");**

**exit(1);**

**}**

**write(STDOUT\_FILENO, buffer, ret); //输出到屏幕**

**char\* str = "hello world --child\n";**

**write(fd[1], str, strlen(str)); //子进程 写**

**close(fd[0]);**

**close(fd[1]);**

**}**

**else //父进程**

**{**

**char\* str = "hello world --father\n";**

**write(fd[1], str, strlen(str)); //父进程 写**

**sleep(1); //确保子进程读取并重新写入**

**char buffer[1024];**

**ret = read(fd[0], buffer, sizeof(buffer)); //父进程写**

**if (ret == -1)**

**{**

**perror("read error");**

**exit(1);**

**}**

**write(STDOUT\_FILENO, buffer, ret); //输出到屏幕**

**close(fd[0]);**

**close(fd[1]);**

**}**

**return 0;**

**}**

/\* 程序运行结果：

hello world --father

hello world –child \*/

## 二、共享内存（mmap，非血缘关系进程间通信）

### 1、函数原型

void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);

函数说明：将磁盘上的文件映射到物理内存中，然后借助指针对映射区进行读写操作。

参数介绍：

addr：建立映射区的首地址，由Linux内核指定，使用时直接传入NULL；

length：创建映射区的大小；

prot：映射区的权限，主要掌握三种权限：PROT\_READ、PROT\_WRITE、PROT\_READ|PROT\_WRITE；

flags：MAP\_SHARED —— 会将映射区所做的修改反应到磁盘上，能够完成进程间通信；

MAP\_PRIVATE —— 不会将映射区所做的修改反应到磁盘上，不能完成进程间通信；

**注：父子进程共享：文件描述符、mmap建立的映射区（必须使用MAP\_SHARED）。**

fd：用来建立映射区的文件描述符；

offset：映射文件的偏移，可以从文件的某个位置开始映射（该值必须为4k的整数倍）。

返回值：成功返回映射区的首地址，失败返回MAP\_FAILED 宏（实质为 (void \*) -1）。

关闭映射区：int munmap(void \*addr, size\_t length);

addr为映射区首地址，length为映射区大小。

### 2、注意

1）length不能为0（即不能创建大小为0的映射区），且length必须小于等于文件大小。

2）创建映射区的过程中隐含着对文件的读操作，因此打开文件时必须有读权限。

3）当flags为MAP\_SHARED时，对映射区的操作权限必须小于等于打开文件的权限，而当flags为MAP\_PRIVATE时则无所谓。

4）offset必须为4k的整数倍。

5）在创建完映射区后，文件描述符已经没有用处，关闭后对指针操作映射区没有影响。

### 3、测试

①利用映射区给文件写入数据

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <sys/mman.h>**

**int main()**

**{**

**int fd = open("mmap\_test.txt", O\_CREAT|O\_RDWR|O\_TRUNC, 0644); //创建一个文件**

**if (fd == -1)**

**{**

**perror("open error");**

**exit(1);**

**}**

**int ret = ftruncate(fd, 4); //将文件扩大为4字节**

**if (ret == -1)**

**{**

**perror("ftruncate error");**

**exit(1);**

**}**

**char\* p = mmap(NULL, 4, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0); //创建映射区**

**if (p == MAP\_FAILED)**

**{**

**perror("mmap error");**

**exit(1);**

**}**

**strcpy(p, "abc\n"); //写数据**

**ret = munmap(p, 4); //关闭映射区**

**if (ret == -1)**

**{**

**perror("munmap error");**

**exit(1);**

**}**

**close(fd);**

**return 0;**

**}**

②利用mmap映射区完成非血缘关系进程间的通信

//分别写两个程序，一个负责往映射区中循环写数据，另一个负责循环读数据

//mmap\_w.c 负责写数据

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <sys/mman.h>**

**int main()**

**{**

**int fd = open("mmap\_test.txt", O\_CREAT|O\_RDWR|O\_TRUNC, 0644); //创建一个文件**

**if (fd == -1)**

**{**

**perror("open error");**

**exit(1);**

**}**

**int ret = ftruncate(fd, sizeof(int)); //将文件扩大**

**if (ret == -1)**

**{**

**perror("ftruncate error");**

**exit(1);**

**}**

**int\* p = mmap(NULL, sizeof(int), PROT\_WRITE, MAP\_SHARED, fd, 0); //创建映射区**

**if (p == MAP\_FAILED)**

**{**

**perror("mmap error");**

**exit(1);**

**}**

**for (int i = 0; i < 10; i++) //写数据**

**{**

**\*p = i;**

**sleep(2);**

**}**

**ret = munmap(p, 4); //关闭映射区**

**if (ret == -1)**

**{**

**perror("munmap error");**

**exit(1);**

**}**

**close(fd);**

**return 0;**

**}**

//mmap\_r.c 负责读数据

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <sys/mman.h>**

**int main()**

**{**

**int fd = open("mmap\_test.txt", O\_RDONLY); //打开一个文件**

**if (fd == -1)**

**{**

**perror("open error");**

**exit(1);**

**}**

**int\* p = mmap(NULL, sizeof(int), PROT\_READ, MAP\_SHARED, fd, 0); //创建映射区**

**if (p == MAP\_FAILED)**

**{**

**perror("mmap error");**

**exit(1);**

**}**

**for (int i = 0; i < 10; i++) //读数据**

**{**

**printf("%d\n", \*p);**

**sleep(2);**

**}**

**int ret = munmap(p, 4); //关闭映射区**

**if (ret == -1)**

**{**

**perror("munmap error");**

**exit(1);**

**}**

**close(fd);**

**return 0;**

**}**

/\*先运行写进程，再运行读进程，结果：

读进程循环打印映射区中由写进程不断修改的数据\*/

### 4、匿名映射

创建映射区时可以不依赖一个文件，而使用匿名映射来代替，方法：

length根据实际需要填写，flags为MAP\_SHARED|**MAP\_ANONYMOUS/MAP\_ANON**，fd填-1，offset填0。

注意：**MAP\_ANONYMOUS/MAP\_ANON**是Linux系统特有的宏，在其他的类Unix系统中无法使用，但是可以使用如下方式创建匿名映射区：

**fd = open(“/dev/zero”, O\_RDWR);**

**p = mmap(NULL, size, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0);**

测试：由匿名映射完成父子进程间的通信

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <sys/mman.h>**

**int main()**

**{**

**char\* p = mmap(NULL, 5, PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANON, -1, 0); //创建匿名映射区**

**if (p == MAP\_FAILED)**

**{**

**perror("mmap error");**

**exit(1);**

**}**

**pid\_t pid = fork(); //创建子进程**

**if (pid == -1)**

**{**

**perror("fork error");**

**exit(1);**

**}**

**else if (pid == 0) //子进程负责读数据**

**{**

**sleep(1); //确保父进程写完数据**

**write(STDOUT\_FILENO, p, 5);**

**}**

**else //父进程负责写数据**

**{**

**strcpy(p, "abc\n");**

**}**

**int ret = munmap(p, 5); //关闭映射区**

**if (ret == -1)**

**{**

**perror("munmap error");**

**exit(1);**

**}**

**return 0;**

**}**

三、信号（开销小）

四、本地套接字（稳定性好）