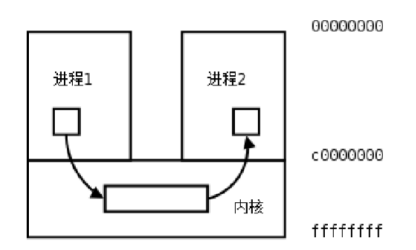
**IPC 方法**

Linux 环境下，进程地址空间相互独立，每个进程各自有不同的用户地址空间。任何一个进程的全局变量在另一个进程中都看不到，所以进程和进程之间不能相互访问，要交换数据必须通过内核，在内核中开辟一块缓冲区，进程1把数据从用户空间拷到内核缓冲区，进程 2 再从内核缓冲区把数据读走，内核提供的这种机



制称为进程间通信（IPC，InterProcess Communication）。

在进程间完成数据传递需要借助操作系统提供特殊的方法，如：文件、管道、信号、共享内存、消息队列、套接字、命名管道等。随着计算机的蓬勃发展，一些方法由于自身设计缺陷被淘汰或者弃用。现今常用的进程间通信方式有：

 ① 管道 (使用最简单)

 ② 信号 (开销最小)

 ③ 共享映射区 (无血缘关系)

 ④ 本地套接字 (最稳定)，用于网络通信

**管道（实现原理：内核借助环形队列机制，使用内核缓冲区实现）**

**管道的概念：**

 管道是一种最基本的 IPC 机制，作用于有血缘关系的进程之间，完成数据传递。调用 pipe 系统函数即可创建一个管道。有如下特质：

1. 其本质是一个伪文件(实为内核缓冲区)

2. 由两个文件描述符引用，一个表示读端，一个表示写端。

3. 规定数据从管道的写端流入管道，从读端流出。

管道的原理: 管道实为内核使用环形队列机制，借助内核缓冲区(4k)实现。

管道的局限性：

① 管道中数据不可反复读取。一旦读走，管道中不再存在。

② 采用半双工通信方式，数据只能在单方向上流动。

③ 只能在有公共祖先的进程间使用管道。（得有血缘关系）

常见的通信方式有，单工通信（电视与遥控器）、半双工通信（对讲机）、全双工通信（手机）。

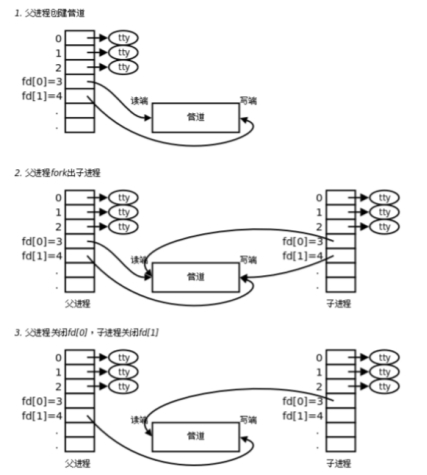
**pipe 函数**

创建管道

int pipe(int pipefd[2]);  成功：0；失败：-1，设置 errno

函数调用成功返回 r/w 两个文件描述符。无需 open，但需手动 close。规定：fd[0] → r； fd[1] → w，就像 0 对应标准输入，1 对应标准输出一样。向管道文件读写数据其实是在读写内核缓冲区。

管道创建成功以后，创建该管道的进程（父进程）同时掌握着管道的读端和写端。如何实现父子进程间通信呢？ 通常可以采用如下步骤：



1. 父进程调用 pipe 函数创建管道，得到两个文件描述符 fd[0]、fd[1]指向管道的读端和写端。

2. 父进程调用 fork 创建子进程，那么子进程也有两个文件描述符指向同一管道。

3. 父进程关闭管道读端，子进程关闭管道写端。父进程可以向管道中写入数据，子进程将管道中的数据读出。由于管道是利用环形队列实现的，数据从写端流入管道，从读端流出，这样就实现了进程间通信。

练习：父子进程使用管道通信，父写入字符串，子进程读出并，打印到屏幕。

int main(int argc, char\* argv[])

{

int fd[2];

char buf[1024];

int pr = pipe(fd);

if(pr < 0)

{

perror("pipe error");

exit(1);

}

pid\_t pid = fork();

if(pid == 0)

{

close(fd[0]);

write(fd[1], "123456789\n", 10);

close(fd[1]);

while(1);

}

if(pid > 0)

{

close(fd[1]);

int rr = read(fd[0], buf, 10);

printf("rr = %d\n", rr);

write(STDOUT\_FILENO, buf, rr);

close(fd[0]);

wait(NULL);

}

int pipe(int pipefd[2])

参数：

pipefd[2]：文件描述符，pipefd[0]:读端，pipefd[1]：写端

返回值：

成功：0

失败：-1 设置errno

**管道的读写行为：**

读管道：

管道里有数据：read返回读到的字节数

管道里没有数据：

有写端：阻塞等待

无写端：read返回0（好像是读到了文件的末尾）

写管道：

管道未满：

有读端：write返回写入的字节数

无读端：进程会异常终止（接收到SIGPIPE信号）

管道已满：

有读端：阻塞等待

无读端：进程会异常终止（接收到SIGPIPE信号）

使用管道实现父子进程间通信，完成：ls -l | wc –l。

练习：使用管道实现父子进程间通信，完成：ls | wc –l。假定父进程实现 ls，子进程实现 wc。

void sys\_err(char \* str)

{

perror(str);

exit(1);

}

int main(int argc, char\* argv[])

{

pid\_t pid;

int i;

int pr;

int fd[2];

pr = pipe(fd);

if(pr < 0)

{

sys\_err("pipe error");

}

for(i = 0; i < 2; i++)

{

pid = fork();

if(pid < 0)

{

sys\_err("fork error");

}

else if(pid == 0)

{

break;

}

}

if(i == 0)

{

printf("ls pid = %d\n", getpid());

close(fd[0]);

dup2(fd[1], STDOUT\_FILENO);

execlp("ls", "ls", "-l", NULL);

close(fd[1]);

sys\_err("execlp error");

}

else if(i == 1)

{

close(fd[1]);

dup2(fd[0], STDIN\_FILENO);

execlp("wc", "wc", "-l", NULL);

close(fd[0]);

sys\_err("execlp error");

}

else

{

close(fd[0]);

close(fd[1]);

int wr1 = wait(NULL);

printf("wait1 pid = %d\n", wr1);

int wr2 = wait(NULL);

printf("wait2 pid = %d\n", wr2);

}

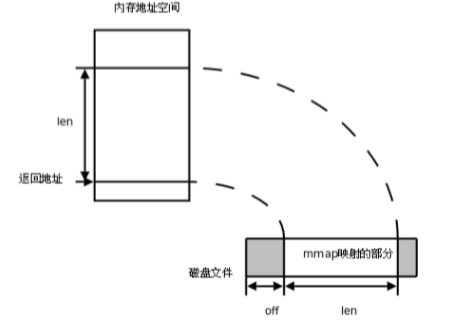
return 0;

}

**存储映射 I/O**

 存储映射 I/O (Memory-mapped I/O) 使一个磁盘文件与存储空间中的一个缓冲区相映射。于是当从缓冲区中取 数据，就相当于读文件中的相应字节。于此类似，将数据存入缓冲区，则相应的字节就自动写入文件。这样，就可在不适用 read 和 write 函数的情况下，使用地址（指针）完成 I/O 操作。

 使用这种方法，首先应通知内核，将一个指定文件映射到存储区域中。这个映射工作可以通过 mmap 函数来实 现。



mmap 函数

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

返回：成功：返回创建的映射区首地址；失败：MAP\_FAILED 宏

参数：

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

 length： 欲创建映射区的大小

 prot： 映射区权限 PROT\_READ、PROT\_WRITE、PROT\_READ|PROT\_WRITE

 flags： 标志位参数(常用于设定更新物理区域、设置共享、创建匿名映射区)

        MAP\_SHARED:  会将映射区所做的操作反映到物理设备（磁盘）上。

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<errno.h>

#include<fcntl.h>

#include<sys/mman.h>

void sys\_err(char\* str)

{

perror(str);

exit(1);

}

int main(int argc, char\* argv[])

{

char\* ptr = NULL;

int fd = open("./mmap.txt", O\_RDWR | O\_CREAT | O\_TRUNC, 0644);

//ftruncate(fd, 200);

lseek(fd, 199, SEEK\_END);

write(fd, "\0", 1);

ptr = (char\*)mmap(NULL, 100, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if(ptr == MAP\_FAILED)

{

sys\_err("mmap error");

}

strcpy(ptr, "123456789");

printf("ptr = %s\n", ptr);

int mur = munmap(ptr, 100);

if(mur < 0)

{

sys\_err("munmap error");

}

close(fd);

return 0;

}

注意：

1、用于创建映射区的文件大小为0，实际指定的映射区大小非0； 总线错误

2、用于创建映射区的文件大小为0，实际指定的映射区大小为0； 无效参数

3、用于创建映射区的文件为只读，实际指定的映射区权限为可读可写 权限被拒绝

4、当flags指定MAP\_SHARED时， 映射区的权限<=文件open权限，创建映射区时，需要文件的读权限，所以“只写”不行

5、当创建完映射区时，文件描述符就可关闭。

6、offset必须是4K（4096）的整数倍（0也是整数）。

7、对映射区内存的访问不能越界。

8、用于munmap的ptr必须是mmap的返回值，不能进行地址偏移

9、当flags指定MAP\_PRIVATE时，对共享内存的数据操作不会映射到文件上（物理磁盘）

10、当flags指定MAP\_PRIVATE时，open文件只需有读权限用与创建共享内存，共享内存的权限可以任意指定。

**mmap 父子进程通信**

 父子等有血缘关系的进程之间也可以通过 mmap 建立的映射区来完成数据通信。但相应的要在创建映射区的时 候指定对应的标志位参数 flags：

MAP\_PRIVATE:  (私有映射)  父子进程各自独占映射区；

MAP\_SHARED:  (共享映射)  父子进程共享映射区；

练习：父进程创建映射区，然后 fork 子进程，子进程修改映射区内容，而后，父进程读取映射区内容，查验是 否共享。

结论：父子进程共享：1. 打开的文件  2. mmap 建立的映射区(但必须要使用 MAP\_SHARED)

例子：

void sys\_err(char\* str)

{

perror(str);

exit(1);

}

//void \*mmap(void \*addr, size\_t length, int prot, int flags,

// int fd, off\_t offset);

int main(int argc, char\* argv[])

{

char\* ptr = NULL;

int fd = open("./mmap.txt", O\_RDWR | O\_CREAT | O\_TRUNC, 0644);

lseek(fd, 199, SEEK\_END);

write(fd, "\0", 1);

ptr = (char\*)mmap(NULL, 100, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE, fd, 0);

if(ptr == MAP\_FAILED)

{

sys\_err("mmap error");

}

pid\_t pid = fork();

if(pid < 0)

{

sys\_err("fork error");

}

else if(pid == 0)

{

strcpy(ptr, "\*\*\*\*\*-------");

printf("child ptr = %s\n", ptr);

}

else

{

usleep(10);

printf("parent ptr = %s\n", ptr);

int wr = wait(NULL);

printf("wait pid = %d\n", wr);

}

int mur = munmap(ptr, 100);

if(mur < 0)

{

sys\_err("munmap error");

}

close(fd);

return 0;

}

**mmap 无血缘关系进程间通信**

实质上 mmap 是内核借助文件帮我们创建了一个映射区，多个进程之间利用该映射区完成数据传递。由于内核 空间多进程共享，因此无血缘关系的进程间也可以使用 mmap 来完成通信。只要设置相应的标志位参数 flags 即可。

若想实现共享，当然应该使用 MAP\_SHARED 了。

值得注意的是：MAP\_ANON 和 /dev/zero 都不能应用于非血缘关系进程间通信。只能用于亲子进程间。

例子：

分为两个进程：

写端代码：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<errno.h>

#include<fcntl.h>

#include<sys/mman.h>

void sys\_err(char\* str)

{

perror(str);

exit(1);

}

//void \*mmap(void \*addr, size\_t length, int prot, int flags,

// int fd, off\_t offset);

int main(int argc, char\* argv[])

{

char\* ptr = NULL;

int fd = open("./mmap.txt", O\_RDWR | O\_CREAT | O\_TRUNC, 0644);

//ftruncate(fd, 200);

lseek(fd, 199, SEEK\_END);

write(fd, "\0", 1);

ptr = (char\*)mmap(NULL, 100, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if(ptr == MAP\_FAILED)

{

sys\_err("mmap error");

}

int n = 0;

while(1)

{

sprintf(ptr, "-------%d-------", n++);

printf(" write ptr = %s\n", ptr);

sleep(1);

}

int mur = munmap(ptr, 100);

if(mur < 0)

{

sys\_err("munmap error");

}

close(fd);

return 0;

}

读端代码：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<errno.h>

#include<fcntl.h>

#include<sys/mman.h>

void sys\_err(char\* str)

{

perror(str);

exit(1);

}

//void \*mmap(void \*addr, size\_t length, int prot, int flags,

// int fd, off\_t offset);

int main(int argc, char\* argv[])

{

char\* ptr = NULL;

int fd = open("./mmap.txt", O\_RDONLY);

ptr = (char\*)mmap(NULL, 100, PROT\_READ, MAP\_SHARED, fd, 0);

if(ptr == MAP\_FAILED)

{

sys\_err("mmap error");

}

while(1)

{

printf("read ptr = %s\n", ptr);

sleep(1);

}

int mur = munmap(ptr, 100);

if(mur < 0)

{

sys\_err("munmap error");

}

close(fd);

return 0;

}

**使用/dev/zero摇钱树文件（无穷大）进行映射：**

**例子：**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**#include<unistd.h>**

**#include<errno.h>**

**#include<fcntl.h>**

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

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

**void sys\_err(char\* str)**

**{**

**perror(str);**

**exit(1);**

**}**

**//void \*mmap(void \*addr, size\_t length, int prot, int flags,**

**// int fd, off\_t offset);**

**int main(int argc, char\* argv[])**

**{**

**char\* ptr = NULL;**

**int fd = open("/dev/zero", O\_RDWR | O\_CREAT | O\_TRUNC, 0644);**

**ptr = (char\*)mmap(NULL, 100, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

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

**{**

**sys\_err("mmap error");**

**}**

**pid\_t pid = fork();**

**if(pid < 0)**

**{**

**sys\_err("fork error");**

**}**

**else if(pid == 0)**

**{**

**strcpy(ptr, "\*\*\*\*\*-------");**

**printf("child ptr = %s\n", ptr);**

**}**

**else**

**{**

**usleep(10);**

**printf("parent ptr = %s\n", ptr);**

**int wr = wait(NULL);**

**printf("wait pid = %d\n", wr);**

**}**

**int mur = munmap(ptr, 100);**

**if(mur < 0)**

**{**

**sys\_err("munmap error");**

**}**

**close(fd);**

**return 0;**

**}**