

深圳大学 计算机与软件学院

College of Computer Science and Software Engineering of Shenzhen University



系统编程

基于TaiShan服务器/openEuler OS 的实践

第二讲：进程间通信 – 管道

进程间通信

■ What is Inter-Process Communication (IPC)

- 进程间传送数据的机制

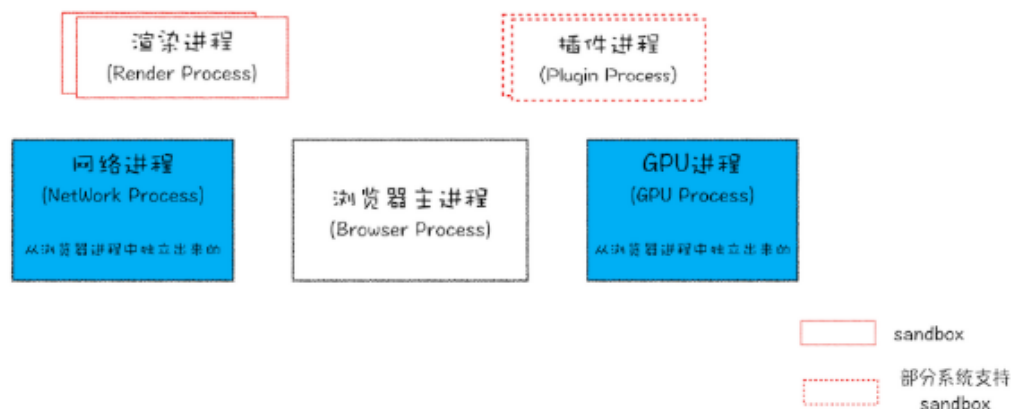
■ Why is it needed?

- 所有任务都由一个进程完成

- ◆ 串行
- ◆ 效率低
- ◆ 可用性差
- ◆ 安全性差

- 多进程分工协作完成

- ◆ 并行/并发
- ◆ 数据/资源隔离
- ◆ 通信同步

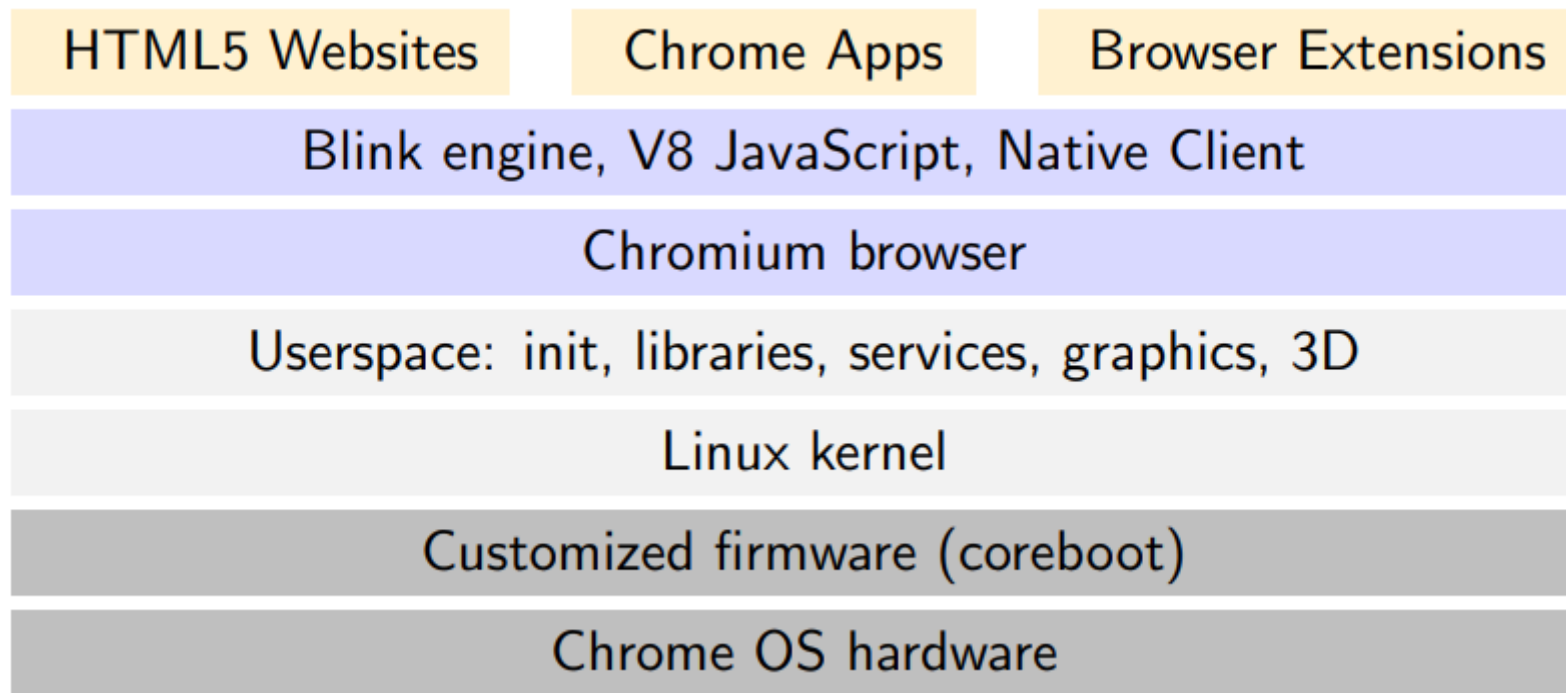


最新的 Chrome 进程架构图

图片出自：

<https://www.cnblogs.com/linm/p/12598933.html>

Google Chrome 体系结构

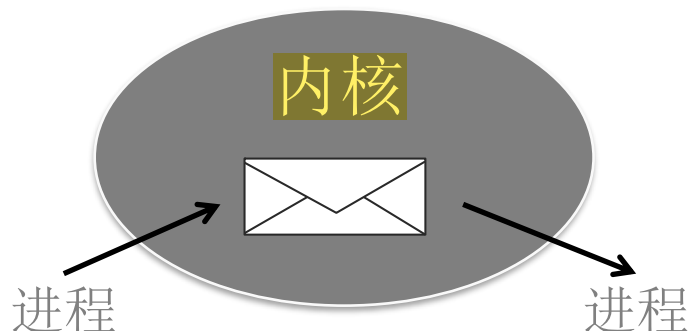


图片出自：

<https://events.static.linuxfound.org/sites/events/files/slides/chrome.pdf>

两大类 IPC

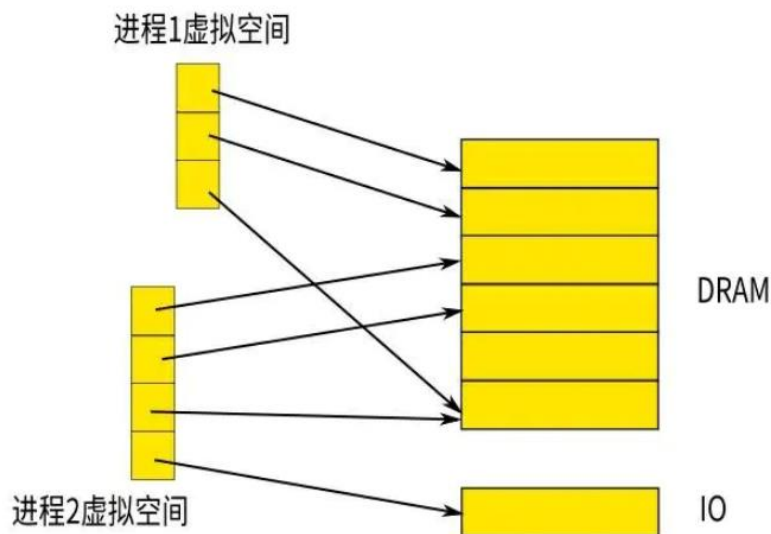
“Intermediary” - 中介



消息通过内核从一个地址空间传到另一个

- 管道
 - 命名管道
 - 信号
 - 信号量
 - 消息队列
 - 套接字(socket)
- 课堂学习内容

“Mind meld” (融合)



linux进程共享内存 (图片来源于网络)

共享地址空间

- 共享内存
- 内存映射文件

图片来源: <https://www.jianshu.com/p/18b71feba27b>

进程间通信 - 管道

■ Shell管道命令

```
$ls | more
```

```
$cat firstpipe.c | grep "pipe"
```

■ 半双工方式的通信

■ 只用于创建管道的进程的子孙进程间（含创建管道的进程）的通信

■ 管道也是文件



图片来源:

http://img.alicdn.com/imgextra/i3/1993033942/T2xI5gXGtXXXXXXXXXX_%21%211993033942.jpg

pipe() 函数

■ 创建管道

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

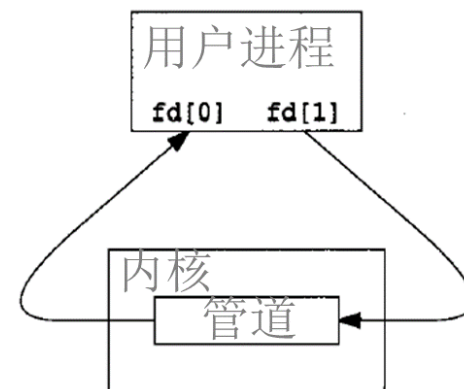
pipefd数组用来返回两个文件描述符,分别指向管道的两端

- pipefd[0]: 管道的读端
- pipefd[1]: 管道的写端
- 先进先出

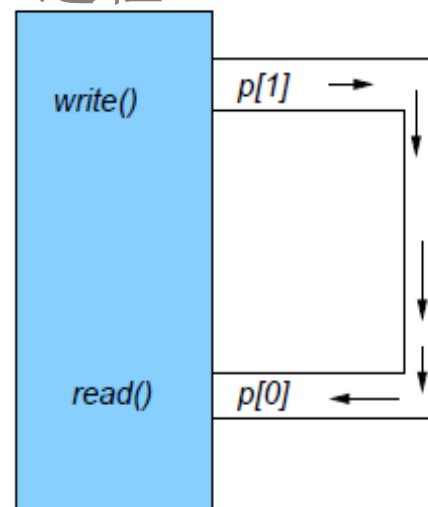
单个进程中管道的使用

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#define READ 0
#define WRITE 1
#define BUFSIZE 100
#define NUM 3
char *msgs[NUM] = {"Good morning.", "Good afternoon.", "Good night."};
int main(int argc, char *argv[])
{
    pid_t cid;
    int pipefd[2], readbytes;
    char msg[BUFSIZE];
    if (pipe(pipefd) == -1) {perror("pipe\n"); exit(EXIT_FAILURE);}
    for (int i = 0; i < NUM; i++){
        write(pipefd[WRITE], msgs[i], strlen(msgs[i])+1);
    }
    for (int i = 0; i < 3; i++){
        memset(msg, '\n', sizeof(msg));
        readbytes = read(pipefd[READ], msg, strlen(msgs[i])+1);
        printf("%s\n", readbytes, msg);
    }
}
```

```
[szu@taishan02-vm-10 pipe]$ gcc -o msgselfbypipe_1 msgselfbypipe_1.c
[szu@taishan02-vm-10 pipe]$ ./msgselfbypipe_1
Good morning.
Good afternoon.
Good night.
```



进程



父/子进程通过管道进行通信

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#define READ 0
#define WRITE 1
#define BUFSIZE 100
```

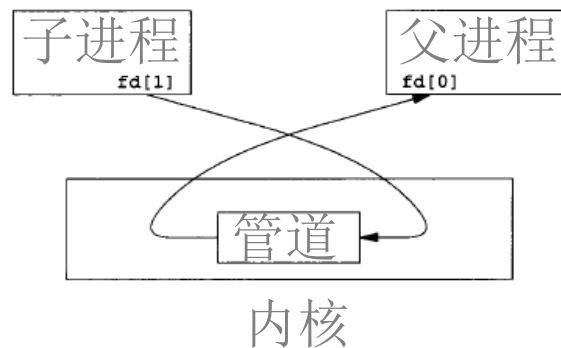
```
char *str = "Welcome home, child.";
```

```
int main(int argc, char *argv[])
{
```

```
    pid_t cid;
    int pipefd[2], readbytes;
    char msg[BUFSIZE];
```

```
    if (pipe(pipefd) == -1) { perror("pipe\n"); exit(EXIT_FAILURE);}
    if ((cid = fork()) < 0) { perror("fork\n"); exit(EXIT_FAILURE);}
```

```
    if (cid > 0) { //Parent, sender
        write(pipefd[WRITE], str, strlen(str)+1);
    } else { //Child, receiver
        readbytes = read(pipefd[READ], msg, sizeof(msg));
        printf("Receive %d bytes from Parent: %s\n", readbytes, msg);
    }
    close(pipefd[WRITE]); close(pipefd[WRITE]);
}
```



```
[szu@taishan02-vm-10 pipe]$ ./p2cbypipe
Receive 21 bytes from Parent: Welcome home, child.
```


习题一、请问如下程序运行结果是什么？

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#define BUFSIZE 10

int main(void){
    char bufin[BUFSIZE]="empty";
    char bufout[BUFSIZE]="hello";
    int bytesin;
    pid_t childpid;
    int fd[2];

    if (pipe(fd) == -1 ){
        perror("failed to create pipe"); exit(23);
    }

    bytesin=strlen(bufin);
    childpid = fork();
    if (childpid == -1) { perror("failed to fork"); exit (23);}
    if (childpid) // parent code
        write(fd[1],bufout,strlen(bufout)+1);
    else // child code
        bytesin=read(fd[0],bufin,strlen(bufin)+1);

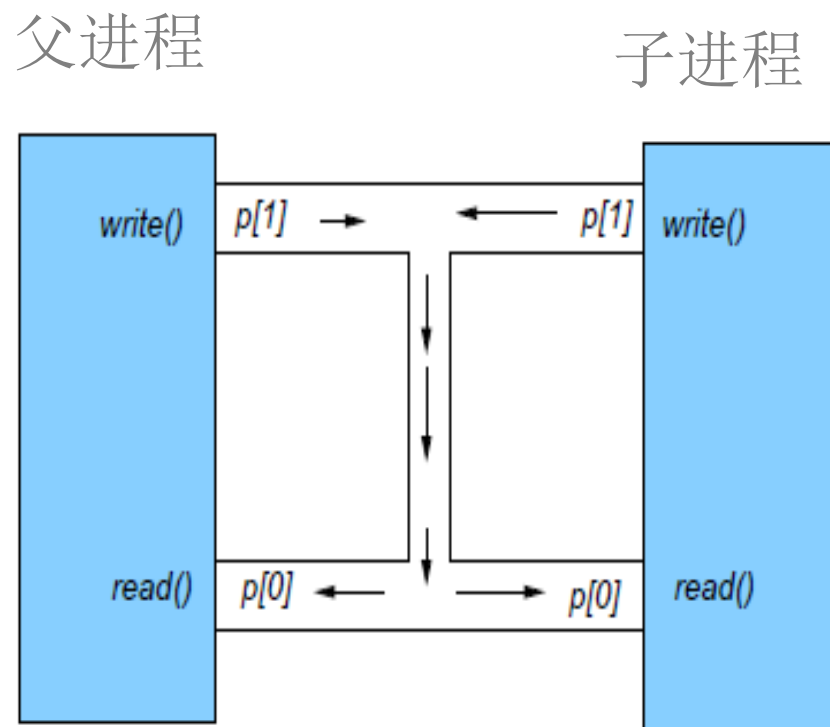
    printf("[%ld]: my bufin is {%.*s}, my bufout is {%s} (parent process %ld)\n",
        (long)getpid(), bytesin, bufin, bufout, (long)getppid());
    return 0;
}
```

[6679]: my bufin is {empty}, my bufout is {hello} (parent process 3420)

[6680]: my bufin is {hello}, my bufout is {hello} (parent process 6679)

然而,这将发生问题...

- 父or子进程都可通过 `p[1]` 写入数据
- 父or子进程都可从 `p[0]` 读数据



父子进程无差别读/写管道, 难以区分特定信息的接收者:
父? 子?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#define READ 0
#define WRITE 1
#define BUFSIZE 100
char *str = "Welcome home, child.";
int main(int argc, char *argv[])
{
```

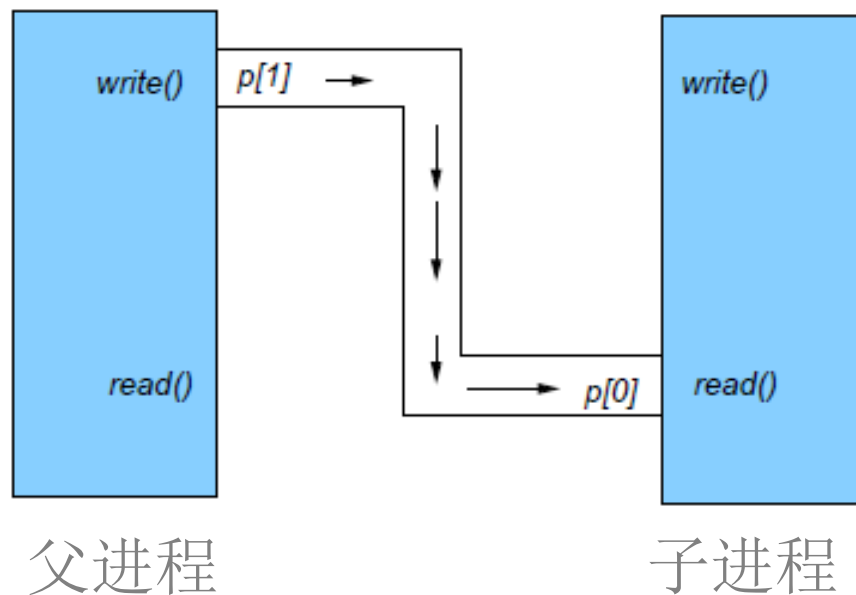
```
    pid_t cid;
    int pipefd[2], readbytes;
    char msg[BUFSIZE];

    if (pipe(pipefd) == -1) { perror("pipe\n"); exit(EXIT_FAILURE); }
    if ((cid = Fork()) < 0) { perror("Fork\n"); exit(EXIT_FAILURE); }

    if (cid > 0) { //Parent, sender
        close(pipefd[READ]);
        write(pipefd[WRITE], str, strlen(str)+1);
        close(pipefd[WRITE]);
    } else { //Child, receiver
        close(pipefd[WRITE]);
        readbytes = read(pipefd[READ], msg, sizeof(msg));
        printf("Receive %d bytes from Parent: %s\n", readbytes, msg);
    }
}
```

代码该做些什么改变?...

确定读者/写者角色,
关闭不再需要写/读端



父进程

子进程

管道的读与写（一）

场景1：写者进程打开了管道，还没写，此时，读者进程读管道，会发生什么？

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

#define MSGSIZE 16

char *msg1 = "Buenos Dias! #1";
char *msg2 = "Buenos Dias! #2";
char *msg3 = "Buenos Dias! #3";
```

```
main(){
    char inbuf[MSGSIZE];
    int p[2], i = 0, rsize = 0;
    pid_t pid;

    time_t t;

    if (pipe(p) == -1) {
        perror("pipe call");
        exit(1);
    }

    switch (pid=fork()){
        case -1: perror("fork call");exit(2);
        case 0:
            sleep(5);
            write(p[1],msg1,MSGSIZE);//if child then write!
            sleep(5);
            write(p[1],msg2,MSGSIZE);
            sleep(5);
            write(p[1],msg3,MSGSIZE);
            break;
        default: for (i=0; i < 3; i++){
            printf("trying to read at time: %ld\n", (long)time(&t));
            rsize = read(p[0],inbuf,MSGSIZE);//if parent then read!
            printf("read data %.*s at time %ld\n", rsize, inbuf, (long)time(&t));
        }
        wait(NULL);
    }
    exit(0);
}
```

```
trying to read at time: 1302364256
read data Buenos Dias! #1 at time 1302364261
trying to read at time: 1302364261
read data Buenos Dias! #2 at time 1302364266
trying to read at time: 1302364266
read data Buenos Dias! #3 at time 1302364271
```

管道的读与写（二）

场景2：写者进程已退出、尚未启动或尚未打开管道，此时，读者进程打开空管道进行读操作，会发生什么？

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

#define MSGSIZE 16

char *msg1 = "Buenos Dias! #1";
char *msg2 = "Buenos Dias! #2";
char *msg3 = "Buenos Dias! #3";

main(){
    char inbuf[MSGSIZE];
    int p[2], i = 0, rsize = 0;
    pid_t pid;

    time_t t;

    if (pipe(p) == -1) {
        perror("pipe call");
        exit(1);
    }

    switch (pid=fork()){
        case -1: perror("fork call");exit(2);
        case 0:
            break;
        default:
            close(p[1]);
            for (i=0; i < 3; i++){
                printf("trying to read at time: %ld\n", (long)time(&t));
                rsize = read(p[0], inbuf, MSGSIZE); //if parent then read!
                printf("read data %.*s at time %ld\n", rsize, inbuf, (long)time(&t));
            }
            close(p[0]);
            wait(NULL);
    }
    exit(0);
}
```

```
trying to read at time: 1302364352
read data  at time 1302364352
trying to read at time: 1302364352
read data  at time 1302364352
trying to read at time: 1302364352
read data  at time 1302364352
```

管道的读与写（三）

场景3：写者进程已退出，此时，读者进程打开非空管道进行读操作，会发生什么？

管道的读与写（四）

场景4：读者进程打开管道，忙，未进行读操作。写者进程持续写管道，将会发生什么？

管道的读与写

■ 管道的一端关闭时

- 写端关闭，读管道

- ◆ 管道非空，读取数据，并返回读取数据的长度

- ◆ 管道为空，返回0

- 读端关闭，写管道

- ◆ 引发信号SIGPIPE

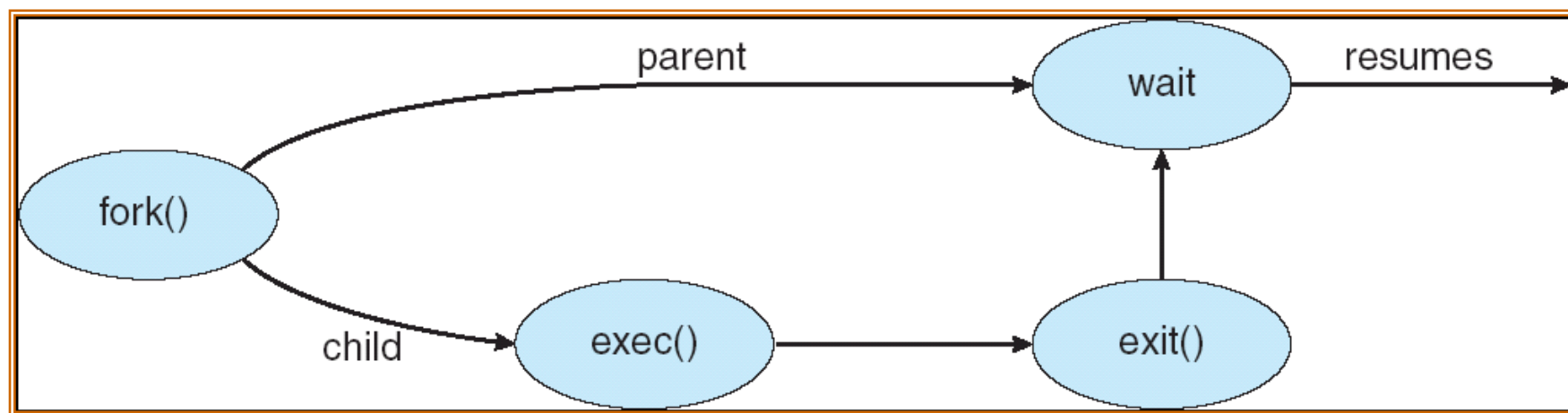
■ 常数PIPE_BUF设定内核中管道缓存器的大小

```
[szu@taishan02-vm-10 pipe]$ sudo find / -name "*.h" -exec grep "PIPE_BUF" {} \; -print | more
[sudo] password for szu:
/* Define if the system reports an invalid PIPE_BUF value. */
/* #undef HAVE_BROKEN_PIPE_BUF */
/usr/include/python2.7/pyconfig-64.h
/* Define if the system reports an invalid PIPE_BUF value. */
/* #undef HAVE_BROKEN_PIPE_BUF */
/usr/include/python2.7-debug/pyconfig-64.h
#define PIPE_BUF 4096 /* # bytes in atomic write to a pipe */
/usr/include/linux/limits.h
#define _POSIX_PIPE_BUF 512
# define _POSIX_HIWAT _POSIX_PIPE_BUF
/usr/include/bits/posix1_lim.h
```


子进程调用`exec()`家族函数后, 还能使用从父进程继承来的管道与父进程通信吗?

exec函数

- 有些应用程序需要子进程执行与其父进程不同的代码
 - **exec**族函数用新的代码覆盖调用进程代码段
 - **fork—exec**是父子进程配合工作的方式之一
 - ◆ 子进程调用**exec**族函数加载并执行新的二进制代码
 - ◆ 父进程继续执行原来的代码。



exec() 族函数

```
#include <unistd.h>
```

```
int execl(const char *path, const char *arg0, ...,  
          NULL)
```

- path: 可执行文件的路径名
- arg0, ... : 所有参数
- NULL指针: 参数终结

exec() 族函数 (一)

场景1: 用户态缓冲区资源会被子进程继承吗?

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

int main(int argc, char *argv[])
{
    int ret = 0;
    printf("My Id is %ld", getpid());
    if ((ret=fork())>0){
        printf("...\n");
    } else {
        ret = execl("/bin/ls", "ls", "-l", ".", NULL);
    }
}
```

```
[szu@taishan02-vm-10 execdemo]$ vi execls.c
[szu@taishan02-vm-10 execdemo]$ gcc execls.c
[szu@taishan02-vm-10 execdemo]$ ./a.out
My Id is 386369...
总用量 20
-rwxrwxr-x. 1 szu szu 71344  9月 16 21:43 a.out
-rw-rw-r--. 1 szu szu   224  9月 16 21:43 execls.c
[szu@taishan02-vm-10 execdemo]$ vi execls.c
[szu@taishan02-vm-10 execdemo]$
```

exec() 族函数 (二)

场景2：打开的文件资源会被子进程继承吗？

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

#define READ 0
#define WRITE 1

int main(int argc, char *argv[]){
    int p1[2], p2[2], filedesc = -1;
    char myinputparam[20];
    pid_t pid;

    if ( (filedesc = open("MytestFile", O_WRONLY|O_CREAT, 0666)) == -1){
        perror("file creation");exit(1);
    }

    if (pipe(p1) == -1) {
        perror("pipe call");exit(1);
    }

    if (pipe(p2) == -1){
        perror("pipe call");exit(1);
    }

    if ( (pid = fork()) == -1){
        perror("fork");exit(1);
    }

    if (pid != 0){
        close(filedesc);close(p1[READ]);close(p1[WRITE]);close(p2[READ]);close(p2[WRITE]);
        close(0);close(1);close(2);
        if (wait(NULL) != pid){
            perror("Waiting for child\n");exit(1);
        }
    } else {
        printf("filedesc=%d\n",filedesc);
        printf("p1[READ]=%d, p1[WRITE]=%d,\n",p1[READ],p1[WRITE]);
        printf("p2[READ]=%d, p2[WRITE]=%d,\n",p2[READ],p2[WRITE]);
        dup2(p2[WRITE],11);
        execlp(argv[1],argv[1],"11",NULL);
        perror("execlp");
    }
}
```

\$gcc -o parent parent.c

exec() 族函数 (二)

场景2: 打开的文件资源会被子进程继承吗?

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>

#define READ 0
#define WRITE 1

int main(int argc, char *argv[]){
    char message[]="Hello there!";

    printf("Operating after the execlp invocation!\n");
    if (write(3,message,strlen(message)+1)==-1)
        perror("Write to 3-file\n");
    else printf("Write to file with file descriptor 3 succeeded\n");
    if (write(5,message,strlen(message)+1)==-1)
        perror("Write to 5-pipe");
    else printf("Write to pipe with file descriptor 5 succeeded\n");
    if (write(7,message,strlen(message)+1)==-1)
        perror("Write to 7-pipe");
    else printf("Write to pipe with file descriptor 7 succeeded\n");
    if (write(11,message,strlen(message)+1)==-1)
        perror("Write to 11-dup2");
    else printf("Write to dup2ed file descriptor 11 succeeded\n");
    if (write(13,message,strlen(message)+1)==-1)
        perror("Write to 13-invalid");
    else printf("Write to invalid file descriptor 13 not feasible\n");
    return 1;
}
```

```
filedesc=3
p1[READ]=4, p1[WRITE]=5,
p2[READ]=6, p2[WRITE]=7,
Operating after the execlp invocation!
Write to file with file descriptor 3 succeeded
Write to pipe with file descriptor 5 succeeded
Write to pipe with file descriptor 7 succeeded
Write to dup2ed file descriptor 11 succeeded
Write to 13-invalid: Bad file descriptor
```

```
$gcc -o child child.c
$./parent ./child
```

exec() 族函数 (三)

场景3：进程成功调用exec()后会返回执行下一条指令吗？

2. 172.31.234.200 (szu)

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

int main(int argc, char *argv[])
{
    int ret = 0;
    ret = execl("/bin/ls", "ls", "-l", ".", NULL);
    printf("My Id is %ld\n, exiting...", getpid());
}
```

```
[szu@taishan02-vm-10 execdemo]$ ./execls0
总用量 40
-rwxrwxr-x. 1 szu szu 71344  9月 16 21:43 a.out
-rwxrwxr-x. 1 szu szu 71248  9月 17 10:19 execls0
-rw-rw-r--. 1 szu szu  183  9月 17 10:19 execls0.c
-rw-rw-r--. 1 szu szu  224  9月 16 21:43 execls.c
[szu@taishan02-vm-10 execdemo]$
```

exec() 族函数

■ 更多成员函数

`int execl (.....)`

`int execlp (.....)`

`int execv (.....)`

`int execve (.....)`

`int execvp (.....)`

若调用不成功，返回-1，并设置errno

■ 进程终止, 操作系统执行的操作

- 取消挂起的定时器和信号
- 释放资源：内存、锁等
- 关闭打开的文件
- 发送SIGCHLD信号给父进程

进程调用exec()族函数

■ 新的进程映像

- 新的堆栈段
- 新的数据段
- 新的代码段

■ 打开的文件资源

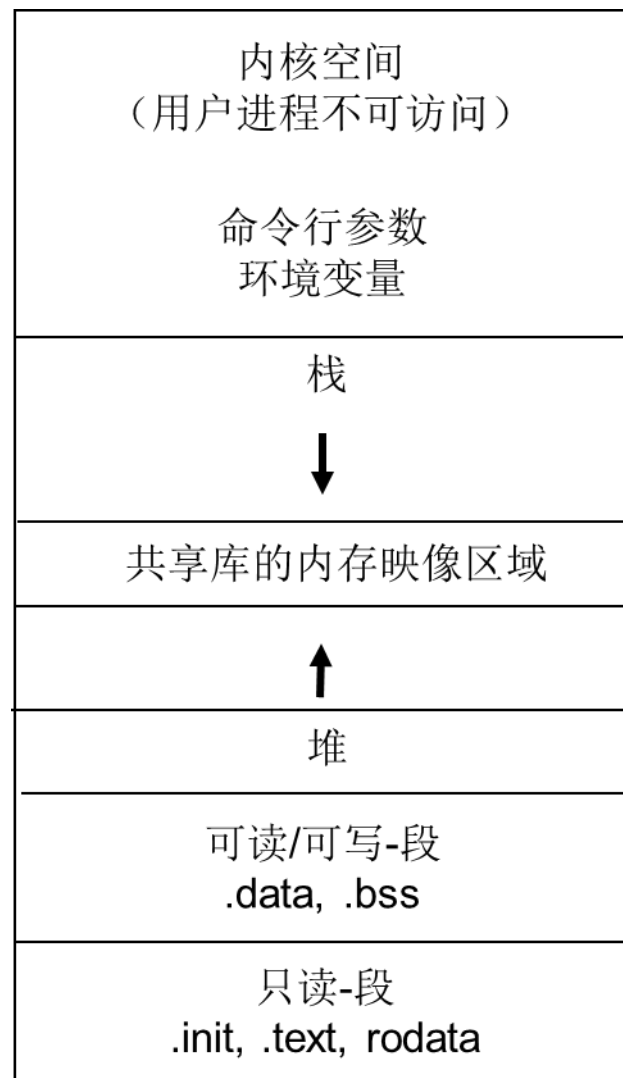
- 默认情况下可通过文件描述符访问
 - ◆ fcntl设置后可以改变
- 相应的逻辑符号名不可访问

■ 执行成功

- 不返回下一条指令
- 进程退出

■ 其他

- 进程ID不变
- 信号状态不变
- 调度参数等不变



回想：管道最初的作用

```
ls | wc -l
```

```
cmd1 | cmd2
```

如何通过系统调用`pipe()`实现命令行管道？

要解决的问题：

- 将`cmd1`的标准输出重定向到`pipe[1]`
- 将`cmd2`的标准输入重定向到`pipe[0]`

创建文件描述符副本

```
#include <unistd.h>
```

```
int dup(int oldfd);
```

将文件描述符oldfd复制到第一个未被使用的文件描述符

■ 返回值:

- 返回值 ≥ 0 : 成功, 返回一个新的文件描述符
- 返回值 = -1: 失败, 具体原因查看 [errno](#)

■ 参数:

- [oldfd](#): 已打开的文件描述符

创建文件描述符副本

```
#include <unistd.h>
```

```
int dup2(int oldfd, int newfd);
```

- 将文件描述符oldfd复制到文件描述符newfd, 如果newfd指向已打开的文件, 则先关闭它
- 返回值:
 - 返回值 ≥ 0 : 成功, 返回一个新的文件描述符
 - 返回值 = -1: 失败, 具体原因查看 [errno](#)
- 参数:
 - [oldfd](#): 要创建副本的文件描述符
 - [newfd](#): 指向oldfd指向的文件

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

```
int main(void)
```

```
{
```

```
    int pipefd[2];
```

```
    pipe(pipefd);
```

```
    if (!fork()) {
```

```
        close(1);    /* 关闭标准输出 */
```

```
        dup(pipefd[1]); /* 将pipefd[1]重定向到标准输出 */
```

```
        close(pipefd[0]);
```

```
        execlp("/bin/ps", "ps", "axj", NULL);
```

```
    } else {
```

```
        close(0);    /* 关闭标准输入 */
```

```
        dup(pipefd[0]); /* 将pipefd[0]重定向到标准输入 */
```

```
        close(pipefd[1]);
```

```
        execlp("/bin/grep", "grep", "systemd", NULL);
```

```
    }
```

```
    return 0;
```

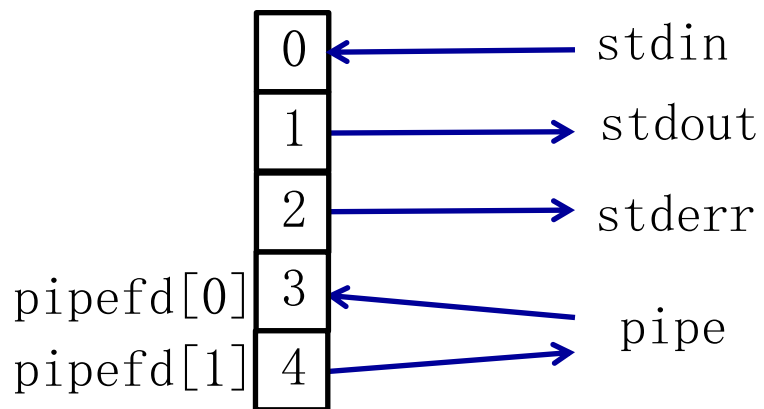
```
}
```

编码实现

```
ps axj | grep systemd
```

ps axj | grep systemd

父进程文件描述符

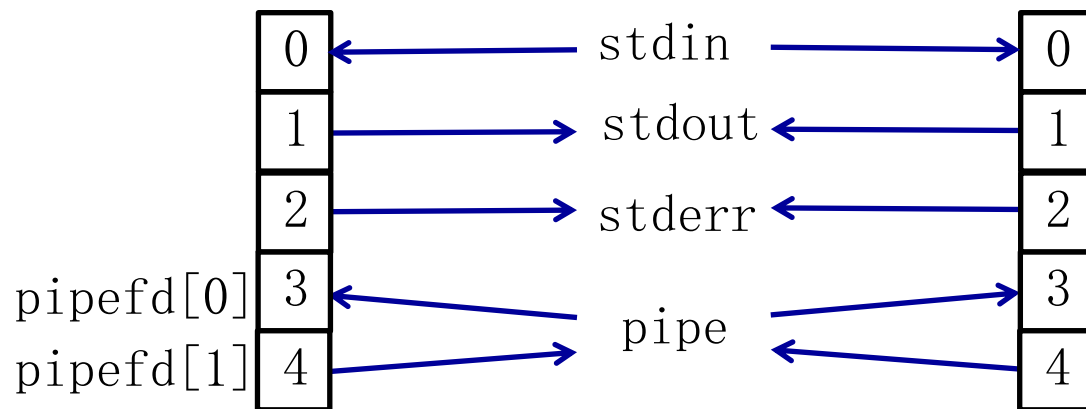


```
pipe(pipefd);
```

ps axj | grep systemd

父进程文件描述符

子进程文件描述符



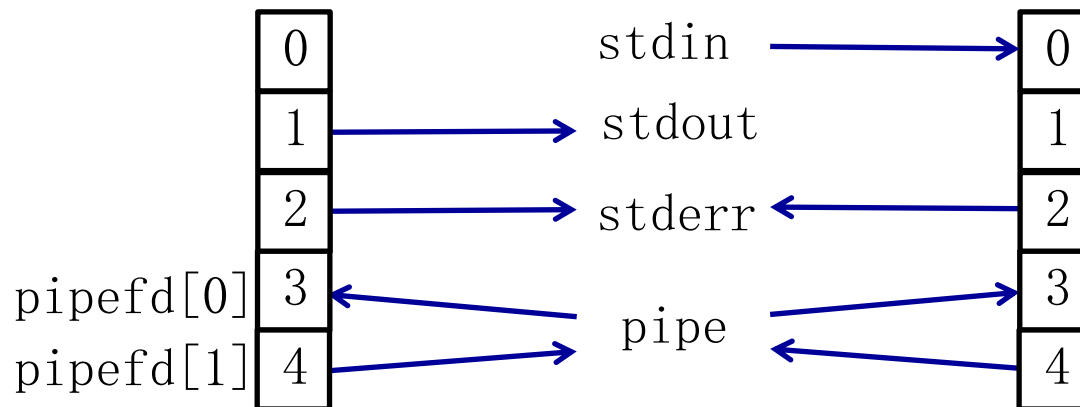
```
pipe(pipefd);  
fork();
```



ps axj | grep systemd

父进程文件描述符

子进程文件描述符



```
pipe(pipefd);  
fork();  
close(0);
```

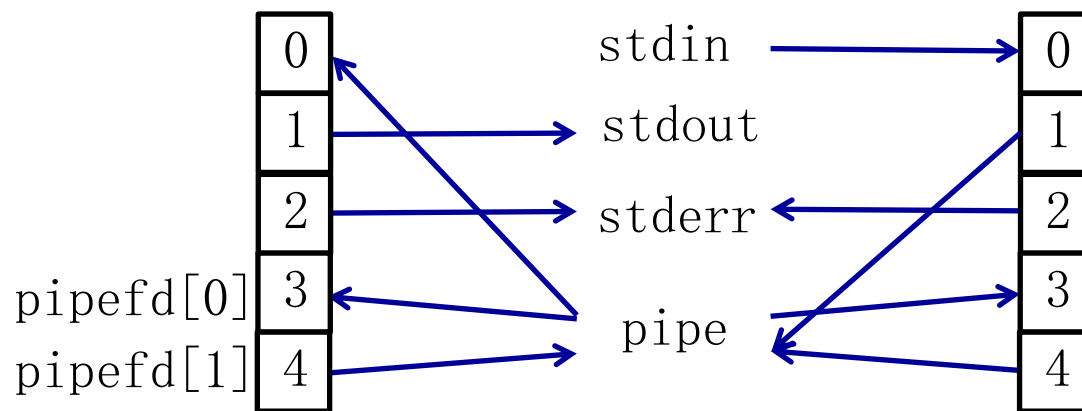


```
close(1);
```


ps axj | grep systemd

父进程文件描述符

子进程文件描述符



```
pipe(pipefd);  
fork();  
close(0);  
dup(pipefd[0]);
```

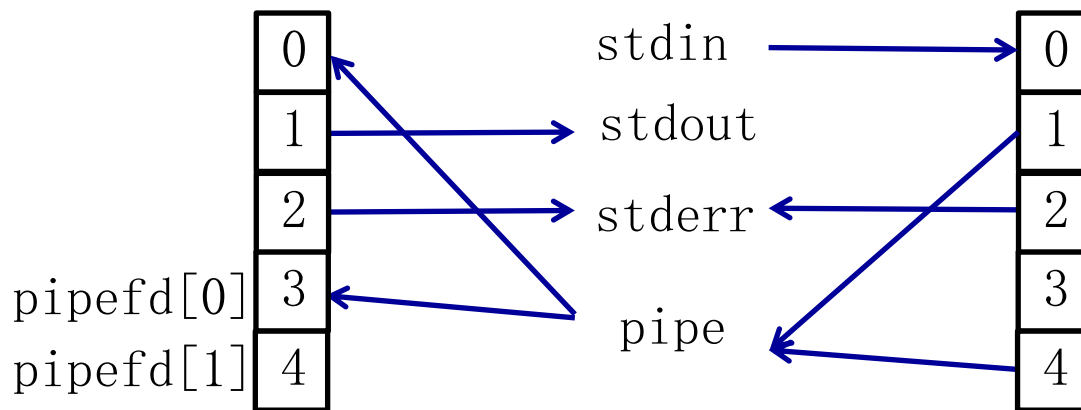


```
close(1);  
dup(pipefd[1]);
```

ps axj | grep systemd

父进程文件描述符

子进程文件描述符



```
pipe(pipefd);  
fork();  
close(0);  
dup(pipefd[0]);  
close(pipefd[1]);  
execlp("/bin/ps",  
        "ps",  
        "axj",  
        NULL);
```

```
close(1);  
dup(pipefd[1]);  
close(pipefd[0]);  
execlp("/bin/grep",  
        "systemd",  
        NULL);
```

ps axj | grep system
如果用dup2()或dup3()实现？

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

```
#define MSGSIZE 16
#define READ 0
#define WRITE 1
```

\$gcc -o main main.c

```
int main(int argc, char *argv[])
{
    int p[2], bytes;
    pid_t pid;

    if (pipe(p) == -1) {
        perror("pipe call");
        exit(1);
    }

    if ( (pid = fork()) == -1){
        perror("fork");
        exit(1);
    }

    if (pid != 0){
        close(p[READ]);
        dup2(p[WRITE], 1);
        close(p[WRITE]);
        execlp(argv[1], argv[1], NULL);
        perror("execlp");
    } else {
        close(p[WRITE]);
        dup2(p[READ], 0);
        close(p[READ]);
        execlp(argv[2], argv[2], NULL);
    }
}
```

文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <sys/types.h>
#define MSGSIZE 20
#define READ 0
#define WRITE 1
```

\$gcc -o param param.c
\$./main ./param ./param

```
int main(int argc, char const *argv[])
{
    int p[2], bytes, res, c;
    char inbuf[10240];
    int pid;

    if (pipe(p) == -1){
        perror("Fail to create a pipe!\n");exit(1);
    }
    pid = fork();
    if (pid != 0){
        close(p[READ]);
        dup2(p[WRITE], 1);
        printf("123\n");
        fflush(stdout);
        wait(NULL);
    } else {
        close(p[WRITE]);
        dup2(p[READ], 0);
        while ((c=getchar())!='\n'){
            printf("%c-", c);
        }
        printf("\n");
        close(p[READ]);
    }
    return 0;
}
```