1. IPC
2. 概述

Inter Process Communication，进程间通信，通过内核提供的缓冲区进行数据交换的机制。

1. IPC通信的方式
2. pipe （匿名）管道（最简单）
3. fifo 有名管道
4. mmap 文件映射共享IO（速度最快）
5. 本地socket （最稳定）
6. 信号（携带信息量最小）
7. 共享内存
8. 消息队列
9. pipe管道
10. 概述

父子进程之间通过管道进行通信，管道分为两端，一端进行输入，一端进行读取。管道是半双工通信，同一时刻只能有一个方向的通信。

1. 函数原型

#include<unisted.h>

int pipe(int pipefd[2])

pipefd: 文件描述符

1. 父子进程实现ps-grep命令

ps命令输出到标准输出中

grep命令从标准输入中匹配

#include<stdio.h>

#include<unistd.h>

int main(){

int fd[2];

pipe(fd);

pid\_t pid = fork();

if(pid == 0){

//子进程

//标准输出重定向到管道写端

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

execlp(“ps”, “ps”, “aux”, NULL);

}

else if(pid > 0){

//父进程

//标准输入重定向到管道读端

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

execlp(“grep”, “grep”, “bash”, NULL);

}

return 0;

}

1. 管道的读写行为
2. 读管道
3. 写端全部关闭

当指向管道写端的所有文件描述符全部关闭时，则管道中无法进入新的内容，当read读到0，相当于读到文件末尾。

1. 写端没有全部关闭

有数据写入时，则read读操作能够读到数据

没有数据写入时，则read读操作阻塞，等待数据的写入。

1. 写管道
2. 读端全部关闭

当指向管道读端的所有文件描述符全部关闭时，则管道写入的内容无法读出，当有进程向管道的写端write时，会产生一个信号SIGPIPE，程序会异常终止。

1. 读端未全部关闭

当指向管道读端的文件描述符没有全部关闭时，若管道已经被塞满了，则write写操作会阻塞；若管道没有被塞满，则write写操作能够正常写入。

1. 管道的优劣
2. 优点

简单，相比信号，套接字实现进程间通信。

1. 缺点

只能单向通信，如想要双向通信需要建立两个管道

只能用于父子或者兄弟进程间的通信。

1. 管道缓冲区大小

使用 ulimit -a 命令来查看当前系统中创建管道文件所对应的内核缓冲区的大小：

也可以使用 fpathconf 函数，成功时返回管道的大小，失败时返回-1，设置errno:

long fpathconf(int fd, int name);

1. FIFO管道
2. 概述

FIFO为命名管道，不同于pipe管道，FIFO可以在不相关的进程之间交换数据。FIFO是Linux基础文件类型中的一种，但是FIFO文件在磁盘上是没有数据块的，仅仅用来表示内核中的一条通道，各个进程可以打开这个文件进行read / write操作，实际上是在读写内核通道，这样实现了进程间的通信。常见的文件 I / O 操作都能用于FIFO.该文件可用open打开。

1. FIFO创建
2. mkfifo 命令

mkfifo 管道名

1. mkfifo函数

int mkfifo(const char \* pathname, mode\_t mode);

成功返回0， 失败返回-1

1. FIFO实现管道通信写端

#include<stdio.h>

#include<unistd.h>

#Include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

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

{

if(argc != 2){

return -1;

}

int fd = open(argv[1], O\_WRONLY);

char buf[256];

int num = 1;

while(1){

memset(buf, 0x00, sizeof(buf));

sprint(buf, “ming”, num++);

write(fd, buf, strlen(buf));

sleep(1);

}

close(fd);

return 0;

}

1. FIFO实现管道通信的读端

#include<stdio.h>

#Include<unistd.h>

#include<sys.type.h>

#include<sys.stat.h>

#include<fcntl.h>

#include<string.h>

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

{

if(argc != 2)

return -1;

int fd = open(argv[1], O\_RDONLY);

char buf[256];

int ret;

while(1){

ret = read(fd, buf, sizeof(buf));

if(ret > 0)

printf(“read:%s\n”, buf);

}

close(fd);

return 0;

}