1. 作业

1. 文件共享通信

fork完之后,子进程里面也是有相应的文件描述符,然后指向相同的文件.

```
//
// Created by bruce on 18-5-17.
//
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <fcntl.h>
int main(int argc,const char* argv[])
{
    int fd = open("temp",O_CREAT|O_RDWR,0664);
    if(fd == -1)
    {
        perror("open error");
        exit(1);;
    }
    pid_t pid = fork();
    if(pid == -1)
    {
        perror("fork error:");
        exit(1);
    }
    if(pid >0)
    {
        const char *p = "whtat hh";

        write(fd,p,strlen(p)+1);
```

```
        close(fd);
    }
    else if(pid ==0)
    {
        //子进程,先睡眠保证父进程可以完全写进去
        sleep(1);
        char buf[1024];
        lseek(fd,0,SEEK_SET);
        int len = read(fd,buf,sizeof(buf));
        printf("%s\n",buf);
        close(fd);

    }
    return 0;
}
```

2. 父进程打印三个子进程的退出状态


```
//
// Created by bruce on 18-5-17.
//
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    int num = 3;
    int i =0;
    pid_t pid;

    for(i =0;i<num;i++)
    {
        pid = fork();
        if(pid ==0)
            break;
    }
    if(i ==0)
    {
        execlp("ps","ps","aux",NULL);
        perror("execlp ps");
        exit(1);
    }
    else if(i ==1)
    {
        execl("./hello","hello",NULL);
        perror("execl error");
        exit(1);
    }
    else if(i ==2)
    {
```

```
            execl("./error","error",NULL);
            perror("execl error");
            exit(1);
        }
        else if(i == num)
        {
            int status;
            pid_t wpid;
            while((wpid = waitpid(-1,&status,WNOHANG))!= -1)//不会等待
            {
                if(wpid ==0)
                    continue; //如果是返回值是0,意味着当前没有可回收子进程
                printf("-------child died pid = %d\n" ,wpid);
                if(WIFEXITED(status))//判断是否是return退出的
                {
                    printf("return value %d\n",WEXITSTATUS(status));
                }
                else if(WIFSIGNALED(status))//判断是否是exit退出的.
                {
                    printf("died by signal: %d \n",WTERMSIG(status));
                }
            }
        }
    }
```

2. 回收进程

- waitpid

  

  

3. 进程间通信

还有一个文件,上面作业题就是.

管道

1. pipe函数

   

例子:

注意:这里/dev/tty就是一个终端的文件.

这些如果不用的文件描述符一定要关掉,如果不关的话,那么grep会一直等待数据的输入,这样就会有很多阻塞的进程.

```
//
// Created by bruce on 18-5-18.
//
#include <unistd.h>
#include <cstdio>
#include <cstdlib>

//
// Created by bruce on 18-5-18.
//
int main()
{
    int num = 2;
    int fd[2];
    int ret = pipe(fd);
    if(ret == -1){
        perror("pipe error");
        exit(1);
    }
    pid_t pid = fork();
    if(pid == -1)
    {
        perror("fork error");
        exit(1);
    }
    if(pid >0)
    {
        //父进程
        // 关闭读端
        close(fd[0]);
        //文件描述符重定向
        // stdout_fileno 管道的写端
        dup2(fd[1],STDOUT_FILENO);
        //执行ps_aux
        execlp("ps","ps","aux",NULL);
        perror("execlp");
        exit(1);
    }
    else if(pid ==0)
    {

        //子进程
```

```
        // 进程读,关闭写
        close(fd[1]);
        dup2(fd[0],STDIN_FILENO);
        execlp("grep","grep","--color=auto","zsh",NULL);
        perror("execlp");
        exit(1);
        // 写管道的操作


    }
//    printf("pipe[0] =  %d\n",fd[0]);
//    printf("pipe[1]= %d\n",fd[1]);
    close(fd[0]);
    close(fd[1]);
    return 0;
}
```

兄弟进程 之间的通信:

```
//
// Created by bruce on 18-5-18.
//
#include <unistd.h>
#include <cstdio>
#include <cstdlib>
#include <sys/wait.h>

//
// Created by bruce on 18-5-18.
//
int main()
{
    int fd[2];
    int ret = pipe(fd);
    if(ret == -1){
        perror("pipe error");
        exit(1);
    }
//    pid_t pid = fork();
    int i =0;
    pid_t pid;
    for(i =0;i<2;i++)
    {
        pid = fork();
        if(pid == 0)
            break;
    }
    if(pid == -1)
    {
        perror("fork error");
        exit(1);
    }
```

```
    //子进程1
    if(i==0)
    {
        //父进程
        // 关闭读端
        close(fd[0]);
        //文件描述符重定向
        // stdout_fileno 管道的写端
        dup2(fd[1],STDOUT_FILENO);
        //执行ps_aux
        execlp("ps","ps","aux",NULL);
        perror("execlp");
        exit(1);
    }
    else if(i==1)
    {
        //子进程2
        // 进程读,关闭写
        close(fd[1]);
        dup2(fd[0],STDIN_FILENO);
        execlp("grep","grep","--color=auto","zsh",NULL);
        perror("execlp");
        exit(1);
        // 写管道的操作
    }
    else if(i==2)
    {
        close(fd[0]);
        close(fd[1]);
        pid_t wpid;
        while((wpid = waitpid(-1,NULL,WNOHANG))!= -1)
        {
            //当返回值是-1的时候,就是出错了,这时候所有的子进程都被回收了
            if(wpid ==0)
                continue;
            printf("the died child is %d\n",wpid);
        }
    }
//    printf("pipe[0] =  %d\n",fd[0]);
//    printf("pipe[1]= %d\n",fd[1]);
//    close(fd[0]);
//    close(fd[1]);
    return 0;
}
```

管道读写行为

设置管道为非阻塞

有名管道

5. 进程间通信

   fifo文件--myfifo

   - 两个不相干的进程A(a.c) 和B(b.c)
   - a.c --> read
     - open("myfifo",O_RDONLY);

   

   默认有阻塞的属性,而不需要sleep来轮流来操作

内存映射区

```
//
// Created by bruce on 18-5-18.
//
#include<sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <cstdio>
#include <cstdlib>

int main()
{
    // 打开一个文件
    int fd = open("english.txt",O_RDWR);
    if(fd == -1)
    {
        perror("open error");

        exit(1);
```

```
    }
    int len = lseek(fd,0,SEEK_END);

    //创建内存映射区
    void *ptr = mmap(NULL,len,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    if(ptr == MAP_FAILED){
        perror("mmap error");
        exit(1);
    }
    printf("%s",(char*)ptr);

    //释放内存映射区
    munmap(ptr,len);
    close(fd);

    return 0;
}
```

思考题:

这里必须指定PROT_READ的权限,这个是必须的,我觉得就是因为是内存,所以起码有个只读的属性.

truncate(path,length)指定的length是多少就是多少.

mmap的优点就是效率高,但是不是阻塞的,需要考虑的就是读与写的冲突问题,有时需要用sleep来控制读和写的速度,这个是和普通Io的共享是一个道理的.而管道的话,那是默认阻塞的.

父子进程间共享内存映射区

```
//
// Created by bruce on 18-5-19.
//

#include<sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include<sys/wait.h>


int main()
```

```
{
    // 打开一个文件
    int fd = open("english.txt",O_RDWR);
    if(fd == -1)
    {
        perror("open error");
        exit(1);
    }
    int len = lseek(fd,0,SEEK_END);

    //创建内存映射区
    void *ptr = mmap(NULL,len,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    if(ptr == MAP_FAILED){
        perror("mmap error");
        exit(1);
    }
    pid_t  pid = fork();
    if(pid == -1)
    {
        perror("fork error");
        exit(1);
    }
    if(pid >0){
        strcpy((char*)ptr,"你是我儿子?");
        wait(NULL);
    }
    else if(pid == 0)
    {
        printf("%s\n",(char*)ptr);
    }
    //释放内存映射区
    munmap(ptr,len);
    close(fd);

    return 0;
}
```

匿名通信(父子进程)

```
//
// Created by bruce on 18-5-19.
//

#include<sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include<sys/wait.h>

int main()
{
```

```
    // 打开一个文件

    int len = 4096;
    //创建内存映射区
    void *ptr = mmap(NULL,len,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    if(ptr == MAP_FAILED){
        perror("mmap error");
        exit(1);
    }
    pid_t  pid = fork();
    if(pid == -1)
    {
        perror("fork error");
        exit(1);
    }
    if(pid >0){
        strcpy((char*)ptr,"你是我儿子?");
        wait(NULL);
    }
    else if(pid == 0)
    {
        printf("%s\n",(char*)ptr);
    }
    //释放内存映射区
    munmap(ptr,len);
    return 0;
}
```

没有血缘关系的进程通信