# IO

常见的阻塞：wait、pause、sleep等函数；read或者write某些文件时候

如何实现非阻塞式IO访问：O\_NONBLOCK（打开文件之前设置）和fcntl（fcontrol，打开文件之后设置）

程序中读取键盘

#include<stdio.h>

#include<unistd.h>

#include<string.h>

int main(){

//读取键盘

//键盘就是标准输入设备。stdin 0 标准输入

char buf[100];

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

printf("before read.\n");

read(0,buf,2); //阻塞住了，知道输入到控制台行缓冲后回车后跳出阻塞，tips控制台行缓冲可以输入很多字符，但是回车后只会取前2个字符

printf("读出的内容是：[%s].\n",buf);

return 0;

}

#include<stdio.h>

#include<unistd.h>

# #include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

include<string.h>

int main(){

//读鼠标 /dev/input cat /dev/input/mouse0 晃动鼠标，出现东西说明是该文件，没反应则不是，ctrl+c终止该程序，换cat /dev/input/mouse1 (信号唤醒了鼠标的阻塞)

int fd=-1;

char buf[200]

fd=open("/dev/input/mouse1",O\_RDONLY);

if(fd<0){

perror("open:");

return -1;

}

menset(buf,0,sizeof(buf));

printf("before read.\n");

read(fd,buf,5);

printf("读出的内容是：[%s].\n",buf);

return 0;

}

阻塞式IO的困境：

下面程序只能按照先动鼠标再按键盘，不能用户随心所欲。

#include<stdio.h>

#include<unistd.h>

# #include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

include<string.h>

int main(){

int fd=-1;

char buf[200]

fd=open("/dev/input/mouse1",O\_RDONLY);

if(fd<0){

perror("open:");

return -1;

}

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

printf("before鼠标 read.\n");

read(fd,buf,5);

printf("鼠标读出的内容是：[%s].\n",buf);

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

printf("before键盘 read.\n");

read(0,buf,2);

printf("键盘读出的内容是：[%s].\n",buf);

return 0;

}

并发式IO的解决方案

非阻塞式IO

1、轮询

#include<stdio.h>

#include<unistd.h>

# include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include<string.h>

int main(){

int fd=-1;

char buf[200];

int flag=-1;

//把0号文件描述符变成非阻塞式的

flag=fcntl(0,F\_GETFL);//先获取原来的flag

flag |=O\_NONBLOCK;

fcntl(0,F\_SETFL,flag);

//这三步之后，0就变成非阻塞式的了

fd=open("/dev/input/mouse1",O\_RDONLY |O\_NONBLOCK); //文件变成非阻塞式的只需要在打开方式之后加上O\_NONBLOCK标志就可以了

if(fd<0){

perror("open:");

return -1;

}

int ret=0;

while(1){

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

ret=read(0,buf,2);

if(ret>0){

printf("键盘读出的内容是：[%s].\n",buf);

}

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

ret=read(fd,buf,5);

if(ret>0){

printf("鼠标读出的内容是：[%s].\n",buf);

}

}

return 0;

}

多路复用IO

IO多路复用原理

为何多路复用：轮询浪费CPU，但是不轮询又不能随心所欲的动键盘和鼠标

select和poll：

外部阻塞式，内部非阻塞式自动轮询多路阻塞式IO

select原理：键盘和鼠标仍然使用阻塞式IO，但是select在二者外部加了一个监听器，键盘和鼠标一旦有一个发生了IO，就会被监听到。唤醒select。

select 函数

#include<stdio.h>

#include<unistd.h>

# #include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

include<string.h>

#include <sys/time.h>

int main(){

int fd=-1;ret=-1;

char buf[200]

fd\_set myset;

struct timeval tm;

fd=open("/dev/input/mouse1",O\_RDONLY);

if(fd<0){

perror("open:");

return -1;

}

//当前有两个fd，一个是fd，一个是0

//处理myset

FD\_ZERO(&myset);

FD\_SET(fd,&myset);

FD\_SET(0,&myset);

tm.tv\_set=10;

tm.tv\_usec=0;

ret=select(fd+1,&mysetn,NULL,NULL,&tm);

if(ret<0){

perror("select:");

return -1;

}

else if(ret==0){

printf("超时了\n");

}

else(){

//这里等待IO的到来，哪个来处理哪个

if(FD\_ISSET(0,&myset)){

//这里处理键盘

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

printf("before键盘 read.\n");

read(0,buf,2);

printf("键盘读出的内容是：[%s].\n",buf);

}

if(FD\_ISSET(fd,&myset)){

//这里处理鼠标

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

printf("before鼠标 read.\n");

read(fd,buf,5);

printf("鼠标读出的内容是：[%s].\n",buf);

}

}

return 0;

}

poll 函数

#include<stdio.h>

#include<unistd.h>

# #include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

include<string.h>

#include <poll.h>

int main(){

int fd=-1;ret=-1;

char buf[200]

struct pollfd myfds[2]={0,0};

fd=open("/dev/input/mouse1",O\_RDONLY);

if(fd<0){

perror("open:");

return -1;

}

//当前有两个fd，一个是fd，一个是0

myfds[0].fd=0;

myfds[0].events=POLLIN;

myfds[1].fd=fd;

myfds[1].events=POLLIN;

ret=select(myfds,fd+1,10000);

if(ret<0){

perror("poll:");

return -1;

}

else if(ret==0){

printf("超时了\n");

}

else(){

if(myfds[0].events==myfds[0].revents){

//这里处理键盘

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

printf("before键盘 read.\n");

read(0,buf,2);

printf("键盘读出的内容是：[%s].\n",buf);

}

if(myfds[1].events==myfd[1].revents){

//这里处理鼠标

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

printf("before鼠标 read.\n");

read(fd,buf,5);

printf("鼠标读出的内容是：[%s].\n",buf);

}

}

return 0;

}

异步IO

几乎可以认为异步IO就是操作系统用软件实现的一套中断响应系统

异步IO的工作方法是：我们当前进程注册一个异步IO事件（使用signal注册一个信号的SIGIO处理函数），然后当前进程可以正常处理自己的事情。当异步事件发生之后，当前进程会收到一个SIGIO信号去处理这个异步事件。

涉及的函数

fcntl（F\_GETFL、F\_SETFLG、O\_ASYNC、F\_SETOWN）

signal或者sigaciton（SIGNO）

#include<stdio.h>

#include<unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include<string.h>

#include<signal.h>

int mousefd=-1;

//绑定到SIGNO信号，在函数内部处理异步事件

void func(int sig){

char buf[200]={0};

if(sig!==SIGIO){

return;

}

read(mousefd,buf,50);

printf("鼠标读出的内容是：[%s].\n",buf);

}

int main(){

char buf[200]

//注册异步通知

mousefd=open("/dev/input/mouse1",O\_RDONLY);

if(mousefd<0){

perror("open:");

return -1;

}

//把鼠标的文件描述符设置为可以接受异步IO

fcntl(mousefd,F\_GETFL);

int flag=-1;

flag |=O\_ASYNC;

fcntl(mousefd,F\_SETFL,flag);

//把异步IO事件的接收进程设置为当前进程

fcntl(mousefd,F\_SETOWN,getpid());

//注册当前进程的SIGIO信号捕获函数

signal(SIGIO,func);

while(1){

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

read(0,buf,2);

printf("键盘读出的内容是：[%s].\n",buf);

}

return 0;

}

存储映射IO

mmap函数

LCD显示和IPC之间共享内存

存储映射IO的特点

共享而不是复制，减少内存操作

处理大文件时效率高，