进程间通信

无名管道

#include <unistd.h>

int pipe(int file[2]);

file[0] 读文件描述符

file[1] 写文件描述符

返回值 成功返回 0

失败返回 -1

有名管道

#include<sys/types.h>

#incldue<sys/stat.h>

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

返回值 成功返回 0

失败返回 -1

错误原因存于errno中

信号

#include<signal.h>

void (\*signal(int signum,void(\* handler)(int)))(int);

typedef void (\*sig\_t)(int);

sig\_t signal(int signum,sig\_t handler);

第一个参数signum指明要处理信号的类型，SIGKILL和SIGSTOP不可取。

第二个参数handler描述了与信号关联的动作，可取三种值：

(1)一个无返回值的函数地址，例：void func(int sig);

(2)SIG\_IGN 表示忽略该信号。

(3)SIG\_DFL 表示恢复系统对该信号的默认处理。

返回值 返回先前的信号处理函数指针，错误返回SIG\_ERR(-1)

#include<sys/types.h>

#include<signal.h>

int kill(pid\_t pid,int sig);

第一个参数pid 是进程号

pid>0 将信号传给进程识别码为pid的进程

pid=0 将信号传给当前进程相同进程组的所有进程

pid=-1 将信号广播传送给系统内所有的进程

pid<-1 将信号传给进程组识别码为pid绝对值的所有进程

第二个参数sig是进程号

返回值 成功返回0

失败返回-1

#include<signal.h>

#include<sys/types.h>

int raise(int sig);

sig参数为进程号

返回值 成功返回0

失败返回-1

#include<unistd.h>

unsigned int alarm(unsigned int seconds);

seconds参数为指定秒数。

返回值 若调用此参数前已经设定了时间闹钟，则返回上一次闹钟时间的剩余时间。否则返回0

失败返回-1

#include<unistd.h>

int pause(void);

返回值 只返回-1

#include<unistd.h>

void sleep(DWORD dwMilliseconds);

void usleep(int micro\_seconds);

ipc

#include<sys/types.h>

#include<sys/ipc.h>

key\_t ftok(const char \*fname,int id);

fname参数就是你指定的文件名

id参数是子序号虽然是int类型，但只是用8bits(1-255)

共享内存

#include<sys/ipc.h>

#include<sys/shm.h>

int shmget(key\_key,size\_t size,int shmflg);

key 0(IPC\_PRIVATE)会建立新共享内存对象

大于0的32位整数，视shmflg来定，通常来源于ftok的返回值

size 0为0时只获取共享内存

大于0的整数为共享内存的大小，以字节为单位

shmflg 0取共享内存标识符，若不存在则函数会报错。

IPC\_CREAT若存在同key值的共享内存则打开。

若不存在则创建。

IPC\_CREAT | IPC\_EXCL若存在同key值的共享内存则打开。

若不存在则报错。

注：Shmflg | 权限

返回值 成功返回共享内存的标识符

出错返回-1

#include<sys/types.h>

#include<sys/shm.h>

void \*shmat(int shmid,const void \*shmaddr,int shmflg);

shmid 共享内存标识符

shmaddr 指定共享内存地址

NULL则自动分配

shmflg SHM\_RDONLY为只读模式

其他为读写模式。

返回值 成功为附加好的共享内存地址

失败为-1

#include<sys/types.h>

#include<sys/shm.h>

int shmdt(const void \*shmaddr);

shmaddr 连接的共享内存的起始地址

返回值 成功为0

失败为-1

#include<sys/types.h>

#include<sys/shm.h>

int shmctl(int shmid,int cmd,struct shmid\_ds \*buf);

shmid 共享内存标识符

cmd IPC\_STAT 得到共享内存的状态

IPC\_SET 改变共享内存的状态

IPC\_RMID 删除这片共享内存

buf 共享内存管理结构体

返回值 成功为0

失败为-1

消息队列

#include<sys/ipc.h>

#include<sys/msg.h>

int msgget(key\_t key,int masgflg);

key 消息队列关联的键

msgflg 消息队列的建立标志和存取权限

返回值 成功返回消息队列标识值

失败返回-1

#include<sys/msg.h>

int msgctl(int msgqid,int cmd,struct msqid\_ds \*buf);

msgqid 消息列队的标识值

cmd IPC\_STAT 读取消息列队的数据结构，存入buf

IPC\_SET 设置消息列队饿的数据结构，取自buf

IPC\_RMID 从系统内核中移走消息列队

buf 消息列队的msqid\_ds结构

返回值 成功返回0

失败返回-1

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

int msgsnd(int msqid,const void \*msgp,size\_t msgsz,int msgflg);

msqid 消息队列的标识符

msgp 指向消息缓冲区的指针

struct msgbuf{long mtype;char mtext[n]}

msgsz 消息的大小

msgflg 0 表示忽略；

IPC\_NOWAIT 如果消息队列为空，则返回一个ENOMSG，并将控制权交回调用函数的进程。如果不指定这个参数，那么进程将被阻塞直到函数可以从队列中得到符合条件的消息为止。如果一个client 正在等待消息的时候队列被删除，EIDRM 就会被返回。如果进程在阻塞等待过程中收到了系统的中断信号，EINTR 就会被返回。

MSG\_NOERROR 如果函数取得的消息长度大于msgsz，将只返回msgsz 长度的信息，剩下的部分被丢弃了。如果不指定这个参数，E2BIG 将被返回，而消息则留在队列中不被取出。当消息从队列内取出后，相应的消息就从队列中删除了。

返回值 成功返回0

失败返回-1

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

ssize\_t msgrcv(int msqid,void \*msgp,size\_t msgsz,long msgtyp,int msgflg);

msqid 消息队列的标识符

msgp 指向消息缓冲区的指针

struct msgbuf{long mtype;char mtext[n]}

msgsz 消息的大小

msgtyp 消息类型 等于0 则返回列队的最早的一个消息

大于0 则返回类型为mtype的第一个消息

小于0 则返回类型小于等于mtype绝对值的最小的一个消息

msgflg 0 表示忽略；

IPC\_NOWAIT 如果消息队列为空，则返回一个ENOMSG，并将控制权交回调用函数的进程。如果不指定这个参数，那么进程将被阻塞直到函数可以从队列中得到符合条件的消息为止。如果一个client 正在等待消息的时候队列被删除，EIDRM 就会被返回。如果进程在阻塞等待过程中收到了系统的中断信号，EINTR 就会被返回。

MSG\_NOERROR 如果函数取得的消息长度大于msgsz，将只返回msgsz 长度的信息，剩下的部分被丢弃了。如果不指定这个参数，E2BIG 将被返回，而消息则留在队列中不被取出。当消息从队列内取出后，相应的消息就从队列中删除了。

返回值 成功则返回mtext数组的实际字节数。

失败返回-1

信号量

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/sem.h>

int semget(key\_t key,int nsems,int semflg);

key 所创建或打开信号量的键值

nsems 创建信号量集中的信号量的个数

semflg 打开模式和权限

返回值 成功返回信号量集的ipc标识符

失败返回-1

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/sem.h>

int semop(int semid,struct sembuf \*sops,unsigned nsops);

semid 信号集的识别码。

sops 指向存储信号操作结构的数组指针。

struct sembuf

{

unsigned short sem\_num;

short sem\_op;

short sem\_flg;

}

sem\_num 操作信号在信号集中的编号，第一个信号的编号是0

sem\_op 大于0 该值会加到现有的信号内含值中。

通常用于释放所控制的使用权。

小于0 其绝对值又大于信号现值，操作将阻塞，直到信号值大于等于sem\_op的绝对值

通常用于获取资源的使用权。

等于0 则挫折将暂时阻塞，直到信号的值变为0.

sem\_flg

nsops 信号操作结构的数量，恒大于等于1

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/sem.h>

int semctl(int semid,int semnum,int cmd,…….);

semid 信号集的标识符，即信号表的索引。

semnum 信号集的索引，用来存取信号集内的某个型号。

cmd 需要执行的命令，有效值有。

IPC\_STAT

IPC\_SET

IPC\_RMID 删除信号集，唤醒所有被阻塞的进程。

IPC\_INFO

。。。。 对于不同的命令，可能用到也可能用不到，是一个联合体semum

union semun

{

int val;

struct semid\_ds \*buf;

unsigned short \*array;

struct seminfo \*\_\_buf;

}

semid\_ds定义在<sys/sem.h>

struct semid\_ds

{

struct ipc\_perm sem\_perm;

time\_t sem\_otime;

time\_t sem\_ctime;

unsigned short sem\_nsems;

}

ipc\_perm定义在<sys/ipc.h>

struct ipc\_perm

{

key\_t key;

uid\_t uid;

gid\_t gid;

uid\_t cuid

gid\_t cgid;

unsigned short mode;

unsigned short seq;

}