**Lab3实验内容**

**进程控制和进程通信**

**实验目的：**

（1）掌握Linux操作系统的进程创建和终止；

（2）利用Linux操作系统提供的“软中断”机制实现进程间的同步。

（3）了解几种典型的进程通信机制（例如：管道通信，消息队列，共享内存、套接字等）以及使用信号量进行进程通信。

**实验介绍：**

**进程控制**

1、 **fork()创建一个新进程。**系统调用格式：pid= int fork(); 其中返回 int 取值意义如下：

0 ：创建子进程，从子进程返回的 id 值;

大于0：从父进程返回的子进程 id 值;

-1 ：创建失败;

fork表示分叉的意思，执行完进程后，将进程“分叉”，即产生了另一个进程。在已经存在的进程中将创建一个新进程。

2、**lockf(files,function,size):**

用作锁定文件的某些段或者整个文件，本函数适用的头文件为：

#include<unistd.h>

参数定义：

int lockf(files,function,size)

int files,function;

long size;

其中：files 是文件描述符：function 是锁定和解锁；1 表示锁定，0 表示解锁。size 是锁定和解锁的字节数，若用 0，表示从文件的当前位置到文件尾。

3．**进程标识符管理：**

int getpid():取得当前进程的标识符；

int getppid():取得当前进程的父进程ID；

**“软中断”通信**

**（1）发送信号kill(pid,sig)**:本进程将指定信号sig发送给指定进程pid，其中参数为pid进程号，pid与sig均为整数值。

**（2）接受信号signal(sig,func**),本进程接受到其他进程发送给他的信号后，完成指定的功能func。func一般是函数。

1、**signal(sig,function):**

允许调用进程控制软中断信号的处理。

头文件为：

#include<signal.h>

参数定义:

signal(sig,function);

int sig;

void(\*func)();

其中：sig 的值是：

SIGHVP 挂起

**SIGINT** 键盘按^c 键或 break 键

SIGQUIT 键盘按 quit 键

SIGILL 非法指令

SIGIOT IOT 指令

SIGEMT EMT 指令

SIGFPE 浮点运算溢出

SIGKILL 要求终止进程

SIGBUS 总线错

SIGSEGV 段违例

SIGSYS 系统调用参数错

SIGPIPE 向无读者管道上写

SIGALRM 闹钟

SIGTERM 软件终结

SIGUSRI 用户定义信号

SIGUSR2 第二个用户定义信号

SIGCLD 子进程死

SIGPWR 电源故障

function 的解释如下：

SIG\_DEL：缺省操作。

对除 SIGPWR 和 SIGCLD 外所有信号的缺省操作是进程终结对信号 SIGQUIT,SIGILL,SIGTRA,SIGIOT,SIGEMT,SIGFPE,SIGBUS,SIGSEGV 和SIGSYS 它产生一内存映像文件。**SIG\_IGN：忽视该信号的出现**。Function:在该进程中的一个函数地址，在核心返回用户态时，它以软中断信号的序号作为参数调用该函数，对除了信号 SIGILL,SIGTRAP 和 SIGTWR 以外的信号，核心自动地重新设置软中断信号处理程序的值为 SIG\_DEL，一个进程不能捕获 SIGKILL 信号。

**管道通信：**

LINUX中最古老的一种进程间通信机制。中间介质就是文件，将写进程与读进程连接在一起，实现两个进程之间的通信。

**消息队列**

这是消息的链表，用于运行在同一台机器上的进程间的通信，用户可以向消息队列中添加消息和读消息。

1、**msgget(key,flag):**

获得一个消息的描述符，该描述符指定一个消息队列以便用于其他系统调用。

该函数使用头文件如下：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/msg.h>

参数定义：

int msgget(key,flag)

key\_t key;

int flag;

语法格式：msgqid=msgget(key,flag)

其中：msgid 是该系统调用返回的描述符，失败则返回-1；flag 本身由操作允许权和控制命令值相“或”得到。

如：IP\_CREAT|0400 是否该队列应被创建；IP\_EXCL |0400 是否该队列的创建应是互斥的；等。

2、**msgsnd(id,msgp,size,flag):**发送一消息。

该函数是用头文件如下：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/msg.h>

参数定义

int msgnd(id,msgp,size,flag)

int id,size,flag;

struct msgbuf \* msgp;

其中：id 是返回消息队列的描述符；msgp 是指向用户存储区的一个构造体指针，size 指示由 msgp 指向的数据结构中字符数组的长度；即消息的长度。这个数组的最大值由 MSG-MAX 系统可调用参数来确定。flag 规定当核心用尽内部缓冲空间时应执行的动作；若在标志 flag 中末设置 IPC\_NOWAIT 位，则当该消息队列中字节数超过一最大值时，或系统范围的消息数超过某一最大值时，调用 msgsnd 进程睡眠。若是设置 IPC\_NOWAIT，则在此情况下，msgsnd 立即返回。

3、**msgrcv(id,msgp,size,type,flag):**接受一消息。

该函数调用使用头文件如下：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/msg.h>

参数定义

int msgrcv(id,msgp,size,type,flag)

int id,size,type,flag;

struct msgbuf \* msgq;

struct sgbuf{long mtpe;chat mtext[];};

语法格式：

count=msgrcv(id,msgp,size,type,flag)

其中：id 是用来存放欲接收消息的拥护数据结构的地址；size 是 msgp 中数据数组的大小； type 是用户要读的消息类型：

type 为 0:接收该队列的第一个消息；

type 为正：接收类型 type 的第一个消息；

type 为负：接收小于或等于 type 绝对值的最低类型的第一个消息。

flag 规定倘若该队列无消息，核心应当做什么事，如果此时设置了IPC\_NOWAIT 标志，则立即返回，若在 flag 中设置了 MSG\_NOERROR，且所接收的消息大小大于 size，核心截断所接受的消息。count 是返回消息正文的字节数。

4**、msgctl(id,cmd,buf):**

查询一个消息描述符的状态，设置它的状态及删除一个消息描述符。

调用该函数使用头文件如下：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/msg.h>

参数定义

int msgctl(id,cmd,buf)

int id,cmd;

struct msgbuf \* msgq;

struct msqid\_ds \* buf;

其中：函数调用成功时返回 0，调用不成功时返回-1。id 用来识别该消息的描述符；cmd 规定命令的类型。

IPC\_START 将与 id 相关联的消息队列首标读入 buf。

IPC\_SET 为这个消息序列设置有效的用户和小组标识及操作允许权和字节的数量。

IPC\_RMID 删除 id 的消息队列。

buf 是含有控制参数或查询结果的用户数据结构的地址。

附：msgid\_ds 结构定义如下：

struct msgid\_ds

{struct ipc\_perm msg\_perm; /\*许可权结构\*/

shot padl[7]; /\*由系统使用\*/

ushort onsg\_qnum; /\*队列上消息数\*/

ushort msg\_qbytes; /\*队列上最大字节数\*/

ushort msg\_lspid; /\*最后发送消息的 PID\*/

ushort msg\_lrpid; /\*最后接收消息的 PID\*/

time\_t msg\_\_stime; /\*最后发送消息的时间\*/

time\_t msg\_rtime; /\*最后接收消息的时间\*/

me\_t msg\_ctime; /\*最后更改时间\*/

}；

struct ipc\_perm

{ushort uid; /\*当前用户 id\*/

ushort gid; /\*当前进程组 id\*/

ushort cuid; /\*创建用户 id\*/

ushort cgid /\*创建进程组 id\*/

ushort mode; /\*存取许可权\*/

{shot patl;long pad2} /\*由系统使用\*/

}；

**共享内存**

实现两个正在运行的进程之间的地址和传递数据的一种简单但非常有效的方式。将同一块内存区域映射到共享它的不同进程的地址空间。

1、**shmget(key,size,flag):**

获得一个共享存储区。

该函数使用头文件如下：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/shm.h>

语法格式：

shmid=shmaget(key,size,flag)

参数定义:

int shmaget(key,size,flag)

key\_t key;

int size,flag;

其中：size 是存储区的字节数，key 和 flag 与系统调用 msgget 中的参数含义相同。

附：

操作允许权 八进制数

用户可读 00400

用户可读 00200

小组可读 00040

小组可读 00020

其他可读 00004

其他可读 00002

控制命令 值

IPC\_CREAT 0001000

IPC\_EXCL 0002000

如:shmid=shmget(key,size,(IPC\_CREAT|0400））；

创建一个关键字为 key,长度为 size 的共享存储区。

2、**shmat(id,addr,flag):**

从逻辑上将一个共享存储区附接到进程的虚拟地址空间上。

该函数调用使用头文件如下：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/msg.h>

参数定义:

char \* shmat(id,addr,flag)

int id,flag;

char \* addr;

语法格式：virtaddr=shmat(id,addr,flag)

其中：id 是共享存储区的标识符，addr 是用户要使用共享存储区附接的虚地址，若 addr 是 0，系统是否对应用户规定的地址做舍入操作。如果 flag 中设置了shm\_rnd 即表示操作系统在必要时舍去这个地址。如果设置了 shm\_rdonly，即表示只允许读操作。viraddr 是附接的虚地址。

3、**shmdt(addr):**

把一个共享存储区从指定进程的虚地址空间分开。

调用该函数使用头文件：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/mhm.h>

参数定义:

int shmdt(addr)

char \* addr

其中，当调用成功时，返回 0 值，调用不成功，返回-1，addr 是系统调用 shmat所返回的地址。

4**、shmctl(id,cmd,buf):**

对与共享存储区关联的各种参数进行操作，从而对共享存储区进行控制。

调用该函数使用头文件：

#include<sy/types.h>

#include<sy/ipc.h>

#include<sy/shm.h>

参数定义:

int shmctl(id,cmd,buf)

int id,cmd;

struct shmid\_ds \* buf;

其中：调用成功返回 0，否则返回-1。id 为被共享存储区的标识符。cmd 规定操作的类型。规定如下：

IPC\_STAT:返回包含在指定的 shmid 相关数据结构中的状态信息，并且把它放置在用户存储区中的\*but 指针所指的数据结构中。执行此命令的进程必须有读取允许权。

IPC\_SET：对于指定的 shmid，为它设置有效用户和小组标识和操作存取权。

IPC\_RMID：删除指定的 shmid 以及与它相关的共享存储区的数据结构。

SHM\_LOCK：在内存中锁定指定的共享存储区，必须是超级用户才可以进

行此项操作。Buf 是一个用户级数据结构地址。

附：

**shmid\_ds**

{struct ipc\_perm shm\_perm; /\*允许权结构\*/

int shm\_segsz; /\*段大小\*/

int padl; /\*由系统使用；\*/

ushort shm\_lpid; /\*最后操作的进程 id；\*/

ushort shm\_cpid; /\*创建者的进程 id；\*/

ushort shm\_nattch; /\*当前附界数；\*/

short pad2; /\*由系统使用；\*/

time\_t shm\_atime; /\*最后附接时间\*/

time\_t shm\_dtime; /\*最后段接时间\*/

time\_t shm\_ctime; /\*最后修改时间\*/

}

Mmap

munmap

1. **Linux线程的基本使用：**

**进程是资源管理的基本单元，而线程是系统调度的基本单元**。一个进程中可以并发多个线程，每条线程并行执行不同的任务。

单核处理器上，也能支持多线程编程模型。

linux操作系统使用符合POSIX线程作为系统标准线程，该POSIX线程标准定义了一整套操作线程的API。

**编程时使用头文件pthread.h,在使用gcc编译时在编译后需要加连接库选项“-lpthread”。**

基本的函数有以下五个：

**创建线程**：init pthread\_create(pthread\_t \* tid,const pthread\_attr\_t \*attr,void \*(\*func)(void\*),void \* arg);

**等待tid线程终止**：init pthread\_join(pthread\_t tid, void \*\*status);

**取得自己的线程ID**：pthread\_t pthread\_self(void);

**将指定的线程脱离，脱离的线程类似于守护线程：**int pthread\_detach(pthread\_t tid);

**终止线程的执行：**void pthread\_exit(void \* status);

1. **Linux线程之互斥锁：**

主要有pthread\_mutex\_init，pthread\_mutex\_destory，pthread\_mutex\_lock，pthread\_mutex\_unlock这几个函数以完成锁的初始化，锁的销毁，上锁和释放锁操作。

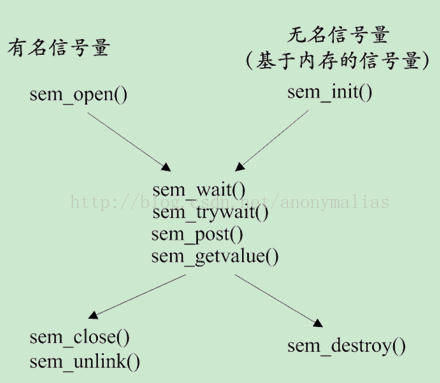
**锁的静态创建** ：pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

**锁的动态创建**： int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \* attr)

**锁的释放：**调用pthread\_mutex\_destory之后，可以释放锁占用的资源。

1. **POSIX信号量：**

**POSIX信号量有两种：有名信号量和无名信号量**，无名信号量也被称作基于内存的信号量。有名信号量通过IPC名字进行进程间的同步，而无名信号量如果不是放在进程间的共享内存区中，是不能用来进行进程间同步的，只能用来进行线程同步。

****

POSIX信号量有三种操作：

（1）**创建一个信号量**。创建的过程还要求初始化信号量的值。

根据信号量取值（代表可用资源的数目）的不同，POSIX信号量还可以分为：

**二值信号量**：信号量的值只有0和1，这和互斥量很类型，若资源被锁住，信号量的值为0，若资源可用，则信号量的值为1；

**计数信号量**：信号量的值在0到一个大于1的限制值（POSIX指出系统的最大限制值至少要为32767）。该计数表示可用的资源的个数。（本实验中应该使用计数信号量）

（2）**等待一个信号量（wait）**。该操作会检查信号量的值，如果其值小于或等于0，那就阻塞，直到该值变成大于0，然后等待进程将信号量的值减1，进程获得共享资源的访问权限。这整个操作必须是一个原子操作。该操作还经常被称为P操作（荷兰语Proberen，意为：尝试）。

（3）**挂出一个信号量（post）**。该操作将信号量的值加1，如果有进程阻塞着等待该信号量，那么其中一个进程将被唤醒。该操作也必须是一个原子操作。该操作还经常被称为V操作（荷兰语Verhogen，意为：增加）

**实验内容**：

（1）编制一段程序，使用系统调用 fork()创建两个子进程，再用系统调用 signal()让父进程捕捉键盘上来的中断信号（即按【Del】键），当捕捉到中断信号后，父进程用系统调用kill()向两个子进程发出信号，子进程捕捉到信号后，分别输出下列信息后终止：

child process1（pid=XX ,ppid=XX） is killed by parent!

child process2 (pid=XX ,ppid=XX) is killed by parent!

父进程等待两个子进程终止后，输出以下信息后终止：

Parent(pid=XX) process is killed!

（2）在上面的程序中增加语句signal(SIGNAL,SIG-IGN)和signal(SIGQUIT,SIG-IGN)，观察执行结果，并分析原因。

（3）使用多线程和信号量解决生产者/消费者问题：有一个长度为N的缓冲池被生产者和消费者共同使用。只要缓冲池未满，生产者就可以将消息送入缓冲池；只要缓冲池不空，消费者便可以从缓冲池中取走一个消息。生产者向缓冲池放入消息的同时，消费者不能操作缓冲池，反之亦然。

参考文档：

<https://ask.csdn.net/questions/757601?sort=id>

<https://www.jianshu.com/p/2876de4314e0>

<https://blog.csdn.net/zmxiangde_88/article/details/7998458>

<https://blog.csdn.net/taoyanqi8932/article/details/56288950>

<https://blog.csdn.net/anonymalias/article/details/9219945>

<https://www.cnblogs.com/LUO77/p/5816326.html>