陈同学的校招八股文系列 之 进程管理(下)

原创 chen079328 陈同学在搬砖 2021-12-05 10:01

这是陈同学的校招八股文第二期

命中率80%的操作系统八股文 之 进程管理(下)

下面是目录

Q1 什么是进程间的通信?

Q2 进程的通信有哪些措施?

整体来说，有下面这些措施

Q3 介绍一下信号机制

1.信号的介绍

2.常见的几种信号

3.信号的诞生

信号的诞生可以分为硬件来源和软件来源

硬件来源:比如我们按下了键盘或者其它硬件故障；

软件来源:一些非法运算等操作,人为的调用信号发送函数指定发送的信号,

4.信号的软件来源

信号常见的软件来源可以通过下面几个API来产生 `kill(),raise()、 sigqueue()、alarm()、setitimer()、abort()

- kill()
- raise()

```
raise ()
#include <signal.h>

int raise(int signo)

向进程本身发送信号，参数为即将发送的信号值。
调用成功返回 0；否则，返回 -1。
```

- sigqueue()

5.2　sigqueue ()

```
#include <sys/types.h>
#include <signal.h>
int sigqueue(pid_t pid, int sig, const union sigval val)
```

返回值:

调用成功返回 0；否则，返回 -1。

参数:

sigqueue的第一个参数是指定接收信号的进程ID，
第二个参数确定即将发送的信号，
第三个参数是一个联合数据结构union sigval，
指定了信号传递的参数，即通常所说的4字节值。

```
typedef union sigval {
               int  sival_int;
               void *sival_ptr;

}sigval_t;
```

使用如下

```
        union sigval mysigval;

        int i;

        int sig;

        pid_t pid;

        char data[10];

        memset(data,0,sizeof(data));

        for(i=0;i < 5;i++)

                data[i]='2';

        mysigval.sival_ptr=data;
```

区别:

sigqueue()是比较新的发送信号系统调用，
主要是针对实时信号提出的（当然也支持前32种），
支持信号带有参数，与函数sigaction()配合使用。

sigqueue()比kill()传递了更多的附加信息，
但sigqueue()只能向一个进程发送信号，
而不能发送信号给一个进程组。
如果signo=0，将会执行错误检查，
但实际上不发送任何信号，
0值信号可用于检查pid的有效性
以及当前进程是否有权限向目标进程发送信号。

在调用sigqueue时，sigval_t指定的信息
会拷贝到对应sig
注册的3参数信号处理函数的siginfo_t结构中，
这样信号处理函数就可以处理这些信息了。
由于sigqueue系统调用支持发送带参数信号，
所以比kill()系统调用的功能要灵活和强大得多。

- alarm()

```
 alarm ()
#include <unistd.h>

unsigned int alarm(unsigned int seconds)

系统调用alarm安排内核为
调用进程在指定的seconds秒后
给调用alarm的进程发出一个SIGALRM的信号。
如果指定的参数seconds为0,
则不再发送 SIGALRM信号。
后一次设定将取消前一次的设定。
该调用返回值为上次定时调用到
发送之间剩余的时间,
或者因为没有前一次定时调用而返回0。

注意，在使用时,
alarm只设定为发送一次信号,
如果要多次发送，就要多次使用alarm调用。
```

- setitimer()

```
5.4    setitimer ()

api
现在的系统中很多程序不再使用alarm调用，而是使用setitimer调用来设置定时器，用getitimer来得到定时器的状态，这两个调用的声明格式如下:

int getitimer(int which, struct itimerval *value);

int setitimer(int which, const struct itimerval *value, struct itimerval *ovalue);

在使用这两个调用的进程中加入以下头文件:

#include <sys/time.h>

 参数:

该系统调用给进程提供了三个定时器,
它们各自有其独有的计时域，当其中任何一个到达,
就发送一个相应的信号给进程，并使得计时器重新开始。
三个计时器由参数which指定，如下所示:


TIMER_REAL：按实际时间计时,
计时到达将给进程发送SIGALRM信号。


ITIMER_VIRTUAL：仅当进程执行时才进行计时。
计时到达将发送SIGVTALRM信号给进程。


ITIMER_PROF：当进程执行时和系统为该进程执行动作时都计时。
与ITIMER_VIR-TUAL是一对,
该定时器经常用来统计进程在用户态和内核态花费的时间。
计时到达将发送SIGPROF信号给进程。
```

定时器中的参数value用来指明定时器的时间，其结构如下：

```
struct itimerval {

        struct timeval it_interval; /* 下一次的取值 */

        struct timeval it_value; /* 本次的设定值 */

};
```

该结构中timeval结构定义如下：

```
struct timeval {

        long tv_sec; /* 秒 */

        long tv_usec; /* 微秒，1秒 = 1000000 微秒*/

};
```

返回值 ：
在setitimer 调用中，参数ovalue如果不为空，
则其中保留的是上次调用设定的值。
定时器将it_value递减到0时，产生一个信号，
并将it_value的值设定为it_interval的值，
然后重新开始计时，如此往复。当it_value设定为0时，
计时器停止，或者当它计时到期，
而it_interval 为0时停止。
调用成功时，返回0；错误时，
返回-1，并设置相应的错误代码errno：

EFAULT：参数value或ovalue是无效的指针。
EINVAL：参数which不是ITIMER_REAL、
ITIMER_VIRT或ITIMER_PROF中的一个。

- abort()

```
  abort()
#include <stdlib.h>

void abort(void);
```

向进程发送SIGABORT信号，默认情况下进程会异常退出，
当然可定义自己的信号处理函数。
即使SIGABORT被进程设置为阻塞信号，
调用abort()后，SIGABORT仍然能被进程接收。
该函数无返回值。

3.信号的注册

1.未决信号集

在进程表的task_struct表项中
有一个软中断信号域(未决信号集合),
该域中每一位对应一个信号。

内核给一个进程发送软中断信号的方法,
是在进程所在的进程表项的信号域
设置对应于该信号的位。

如果信号发送给一个正在睡眠的进程,
如果进程睡眠在可被中断的优先级上,
则唤醒进程;

否则仅设置进程表中信号域相应的位,
而不唤醒进程。

如果发送给一个处于可运行状态的进程,
则只置相应的域即可。

2.未决信号集相关数据结构

进程的task_struct结构中
有关于本进程中未决信号的数据成员：
struct sigpending pending;

```
struct sigpending{

        struct sigqueue *head, *tail;

        sigset_t signal;

};
```

第一、第二个成员
分别指向一个sigqueue类型的结构链
（称之为"未决信号信息链"）的首尾，
信息链中的每个sigqueue结构
刻画一个特定信号所携带的信息，
并指向下一个sigqueue结构：

```
struct sigqueue{

        struct sigqueue *next;

        siginfo_t info;

}
```

第三个成员是进程中所有未决信号集，
信号集数据结构

信号集被定义为一种数据类型：

```
typedef struct {
 unsigned long sig[_NSIG_WORDS];

} sigset_t
```

信号集用来描述信号的集合，每个信号占用一位。

信号在进程中注册指的就是信号值
加入到进程的未决信号集
sigset_t signal（每个信号占用一位）中，
并且信号所携带的信息
被保留到未决信号信息链的某个sigqueue结构中。
只要信号在进程的未决信号集中，
表明进程已经知道这些信号的存在，
但还没来得及处理，或者该信号被进程阻塞。

3.未决信号集相关操作函数

就是上面的信号发送函数

4.未决信号可靠非可靠信号注册区别:

当一个实时信号发送给一个进程时,
不管该信号是否已经在进程中注册,
都会被再注册一次,
因此,信号不会丢失,
因此,实时信号又叫做"可靠信号"。
这意味着同一个实时信号
可以在同一个进程的未决信号信息链中
占有多个sigqueue结构
(进程每收到一个实时信号,
都会为它分配一个结构来登记该信号信息,
并把该结构添加在未决信号链尾,
即所有诞生的实时信号都会在目标进程中注册)。

当一个非实时信号发送给一个进程时,
如果该信号已经在进程中注册
(通过sigset_t signal指示),
则该信号将被丢弃,造成信号丢失。
因此,非实时信号又叫做"不可靠信号"。
这意味着同一个非实时信号
在进程的未决信号信息链中,
至多占有一个sigqueue结构。

总之信号注册与否,
与发送信号的函数(如kill()或sigqueue()等)
以及信号安装函数(signal()及sigaction())无关,
只与信号值有关
(信号值小于SIGRTMIN的信号最多只注册一次,
信号值在SIGRTMIN及SIGRTMAX之间的信号,
只要被进程接收到就被注册)

5.信号集操作
信号集操作函数

Linux所支持的所有信号
可以全部或部分的出现在信号集中，
主要与信号阻塞相关函数配合使用。
下面是为信号集操作定义的相关函数：

```
#include <signal.h>
// 以下函数，成功，返回 0；失败，返回 -1.
int sigemptyset(sigset_t *set)                  // 清除信号集中的所以信号
int sigfillset(sigset_t *set)                   // 初始化set信号集中的信号，并把所有该程序的信号加入到信号集中
int sigaddset(sigset_t *set, int signo)         // 添加signo信号到信号集中
int sigdelset(sigset_t *set, int signo)         // 信号集中删除signo信号
int sigismember(const sigset_t *set, int signum); //判断signum位置是否为1
```

sigemptyset(sigset_t *set)初始化由set指定的信号集，信号集里面的所有信号被清空；

sigfillset(sigset_t *set)调用该函数后，set指向的信号集中将包含linux支持的64种信号；

sigaddset(sigset_t *set, int signum)在set指向的信号集中加入signum信号；

sigdelset(sigset_t *set, int signum)在set指向的信号集中删除signum信号；

sigismember(const sigset_t *set, int signum)判定信号signum是否在set指向的信号集中。

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
void print_sigset(sigset_t *set);
int main(void)
{
    sigset_t myset;
    sigemptyset(&myset);
    sigaddset(&myset,SIGINT);
    sigaddset(&myset,SIGQUIT);
    sigaddset(&myset,SIGUSR1);
    sigaddset(&myset,SIGRTMIN);
    print_sigset(&myset);

    return 0;

}
void print_sigset(sigset_t *set)
{
    int i;
    for(i = 1; i < NSIG; ++i){
        if(sigismember(set,i))
            printf("1");
        else
            printf("0");
    }
    putchar('\n');
}
```

4.信号的屏蔽

每个进程都有一个用来描述哪些信号递送到进程时将被阻塞的信号集，该信号集中的所有信号在递送到进程后都将被阻塞。下面是与信号阻塞相关的几个函数：

```c
int sigsuspend(const sigset_t *mask));
```

sigsuspend(const sigset_t *mask))用于在接收到某个信号之前，临时用mask替换进程的信号掩码，并暂停进程执行，直到收到信号为止。sigsu

```c
int sigpending(sigset_t *set));
```

sigpending(sigset_t *set))获得当前已递送到进程，却被阻塞的所有信号，在set指向的信号集中返回结果。

5.信号的处理时机

在其从内核空间返回到用户空间时会检测是否有信号等待处理。

如果存在未决信号等待处理且该信号没有被进程阻塞，

则在运行相应的信号处理函数前，进程会把信号在未决信号链中占有的结构卸掉。

对于非实时信号来说，由于在未决信号信息链中最多只占用一个sigqueue结构，因此该结构被释放后，应该把信号在进程未决信号集中删除（信号注销完毕）；

而对于实时信号来说，可能在未决信号信息链中占用多个sigqueue结构，因此应该针对占用sigqueue结构的数目区别对待：

如果只占用一个sigqueue结构（进程只收到该信号一次），则执行完相应的处理函数后应该把信号在进程的未决信号集中删除（信号注销完毕）。

否则待该信号的所有sigqueue处理完毕后再在进程的未决信号集中删除该信号。

当所有未被屏蔽的信号都处理完毕后，即可返回用户空间。

对于被屏蔽的信号，当取消屏蔽后，在返回到用户空间时会再次执行上述检查处理的一套流程。

6.信号递达以后的执行动作

收到信号的进程对各种信号有不同的处理方法。处理方法可以分为三类：

第一种方法是，对该信号的处理保留系统的默认值，这种缺省操作，对大部分的信号的缺省操作是使得进程终止。

第二种方法是，忽略某个信号，对该信号不做任何处理，就象未发生过一样。

第三种是对于需要处理的信号，进程可以指定指定处理函数，即信号的的安装 见下一点

7.信号的安装

如果进程要处理某一信号，那么就要在进程中安装该信号。

安装信号主要用来确定信号值及进程针对该信号值的动作之间的映射关系，即进程将要处理哪个信号；该信号被传递给进程时，将执行何种操作。

linux主要有两个函数实现信号的安装：signal()、sigaction()。其中signal()只有两个参数，不支持信号传递信息，主要是用于前32种非实时信号的安装；

而sigaction()是较新的函数（由两个系统调用实现：sys_signal以及sys_rt_sigaction），有三个参数，支持信号传递信息，主要用来与 sigqueue() 系统调用配合使用，

当然，sigaction()同样支持非实时信号的安装。sigaction()优于signal()主要体现在支持信号带有参数。

- sigaction
- signal

```
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
void sigroutine(int dunno)
{ /* 信号处理例程，其中dunno将会得到信号的值 */
        switch (dunno) {
        case 1:
        printf("Get a signal -- SIGHUP ");
        break;
        case 2:
        printf("Get a signal -- SIGINT ");
        break;
        case 3:
        printf("Get a signal -- SIGQUIT ");
        break;
        }
        return;
}

int main() {
        printf("process id is %d ",getpid());
        signal(SIGHUP, sigroutine); //* 下面设置三个信号的处理方法
        signal(SIGINT, sigroutine);
        signal(SIGQUIT, sigroutine);
        for (;;) ;
}
```

其中信号SIGINT由按下Ctrl-C发出，信号SIGQUIT由按下Ctrl-发出。该程序执行的结果如下：

```
localhost:~$ ./sig_test
```

```
process id is 463

Get a signal -SIGINT //按下Ctrl-C得到的结果

Get a signal -SIGQUIT //按下Ctrl-得到的结果

//按下Ctrl-z将进程置于后台

 [1]+ Stopped ./sig_test

localhost:~$ bg

 [1]+ ./sig_test &

localhost:~$ kill -HUP 463 //向进程发送SIGHUP信号

localhost:~$ Get a signal — SIGHUP

kill -9 463 //向进程发送SIGKILL信号，终止进程

localhost:~$
```

Q4 介绍一下管道机制

1.有名管道

2.无名管道

Q5 介绍一下消息队列机制

1.概念

2.特点

3.使用

Q6 介绍一下共享内存机制

1.介绍

2.特点

3.原理

4.指令

5.使用

Q7 介绍一下socket套接字机制

一般用于两个不同IP主机之间的通信，在后面 网络编程 那一部分会讲

Q8 介绍一下信号量机制

1.介绍

2.实现机制

3.PV操作

4.应用：生产者消费者问题

5.应用：读者写者问题

6.应用：哲学家进餐问题

我是陈同学

一名放弃BAT offer的深圳老师

后台回复【故事】

了解我的故事