嵌入式系统工程师

信号

大纲

- 概述
- 信号的基本操作

大纲

- 概述
- 信号的基本操作

信号(signal)概述

- 信号是软件中断，它是在软件层次上对中断机制的一种模拟。

  信号可以导致一个正在运行的进程被另一个正在运行的异步进程中断，转而处理某一个突发事件。

- 信号是一种异步通信方式。

  进程不必等待信号的到达，进程也不知道信号什么时候到达。

- 信号可以直接进行用户空间进程和内核空间进程的交互，内核进程可以利用它来通知用户空间进程发生了哪些系统事件。

信号(signal)概述

- 每个信号的名字都以字符SIG开头。
- 每个信号和一个数字编码相对应，在头文件signum.h中，这些信号都被定义为正整数。
- 参考路径:

  /usr/include/i386-linux-gnu/bits/signum.h
- 在Linux下，要想查看这些信号和编码的对应关系，可使用命令: kill -l

信号(signal)概述

```
[root@localhost ~]# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
[root@localhost ~]#
```

信号(signal)概述

- 以下条件可以产生一个信号
  - 1、当用户按某些终端键时，将产生信号。

    例如:

    终端上按“Ctrl+c”组合键通常产生中断信号SIGINT、终端上按"Ctrl+\"键通常产生中断信号SIGQUIT、终端上按"Ctrl+z"键通常产生中断信号SIGSTOP。
  - 2、硬件异常将产生信号。

    除数为0，无效的内存访问等。这些情况通常由硬件检测到，并通知内核,然后内核产生适当的信号发送给相应的进程。

信号(signal)概述

- 3、软件异常将产生信号。

  当检测到某种软件条件已发生，并将其通知有关进程时，产生信号。

- 4、调用kill函数将发送信号。

  注意：接收信号进程和发送信号进程的所有者必须相同，或发送信号进程的所有者必须是超级用户。

- 5、运行kill命令将发送信号。

  此程序实际上是使用kill函数来发送信号。也常用此命令终止一个失控的后台进程。

信号(signal)概述

- 一个进程收到一个信号的时候，可以用如下方法进行处理：
  - 忽略此信号

    大多数信号都可以使用这种方式处理，但SIGKILL和SIGSTOP决不能被忽略，原因是它们向超级用户提供一种使进程终止的可靠方法。
  - 执行系统默认动作

    对大多数信号来说，系统默认动作是用来终止该进程。
  - 自定义信号处理函数

    用用户定义的信号处理函数处理该信号。

大纲

- 概述
- 信号的基本操作

信号的基本操作

- #include <sys/types.h>
  #include <signal.h>
  int kill(pid_t pid, int signum);
  功能:
  给指定进程发送信号。
  参数:
  pid: 详见下页
  signum: 信号的编号
  返回值:
  成功返回 0，失败返回 -1。

信号的基本操作

- pid的取值有4种情况:

  pid>0: 将信号传送给进程ID为pid的进程。

  pid=0: 将信号传送给当前进程所在进程组中的所有进程。

  pid=-1: 将信号传送给系统内所有的进程。

  pid<-1: 将信号传给指定进程组的所有进程。这个进程组号等于pid的绝对值。

例: 01_kill.c

信号的基本操作

注意：

使用kill函数发送信号，接收信号进程和发送信号进程的所有者必须相同，或者发送信号进程的所有者是超级用户。

信号的基本操作

- #include <unistd.h>

  unsigned int alarm(unsigned int seconds);

  功能:

  在seconds秒后，向调用进程发送一个SIGALRM信号，SIGALRM信号的默认动作是终止调用alarm函数的进程。

  返回值:

  若以前没有设置过定时器，或设置的定时器已超时，返回0；否则返回定时器剩余的秒数，并重新设定定时器。

例: 02_alarm.c

一个进程只有一个闹钟，多次设置的话，采用最后设置的。（重新设置）

信号的基本操作

- #include <signal.h>

  int raise(int signum);

  功能:

  给调用进程本身送一个信号。

  参数:

  signum: 信号的编号。

  返回值:

  成功返回 0，失败返回 -1。

例: 03_raise.c

信号的基本操作

- #include <unistd.h>

  int pause(void); 无参 <一个pause只能等待一个信号>

  功能:

  将调用进程挂起直至捕捉到信号为止。这个函数通常用于判断信号是否已到。

  返回值:

  直到捕获到信号，pause函数才返回-1，且errno被设置成EINTR。

例: 04_pause.c

信号的基本操作

- #include <stdlib.h>

  void abort(void);

  功能:

  向进程发送一个SIGABRT信号，默认情况下进程会退出。

  注意:

  即使SIGABRT信号被加入阻塞集，一旦进程调用了abort函数，进程也还是会被终止，且在终止前会刷新缓冲区，关文件描述符。

信号的基本操作

- 处理信号的函数主要有如下三种:
  - signal函数
  - 信号集函数组
  - sigaction函数

信号的基本操作

- #include <signal.h>

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum,
                    sighandler_t handler);
```

功能:

注册信号处理函数(除了SIGKILL信号、SIGSTOP信号)，即确定收到信号后处理函数的入口地址。

参数:

signum: 信号编号

信号的基本操作

- handler的取值:

  忽略该信号: SIG_IGN

  执行系统默认动作: SIG_DFL

  自定义信号处理函数: 处理函数名

- 返回值:

  成功: 返回函数地址，该地址为此信号上一次注册的信号处理函数的地址。

  失败: 返回SIG_ERR

例: 05_signal_1.c 05_signal_2.c

信号的基本操作

- 可重入函数

  可重入函数是指函数可以由多个任务并发使用，而不必担心数据错误。

  编写可重入函数:

  - 不使用（返回）静态的数据、全局变量（除非用信号量互斥）。
  - 不调用动态内存分配、释放的函数。
  - 不调用任何不可重入的函数（如标准I/O函数）。

  注：即使信号处理函数使用的都是可重入函数（常见的可重入函数），也要注意进入处理函数时，首先要保存errno的值，结束时，再恢复原值。因为，信号处理过程中，errno值随时可能被改变。

信号的基本操作

- 信号集
  - 信号集概述
    - 一个用户进程常常需要对多个信号做出处理。为了方便对多个信号进行处理，在Linux系统中引入了信号集。
    - 信号集是用来表示多个信号的数据类型。

信号的基本操作

- 信号集相关的操作主要有如下几个函数:
  - sigemptyset
  - sigfillset
  - sigismember
  - sigaddset
  - sigdelset

信号的基本操作

- #include <signal.h>

  int sigemptyset(sigset_t *set);

  功能:

  初始化由set指向的信号集，清除其中所有的信号即初始化一个空信号集。

  参数:

  set: 信号集标识的地址，以后操作此信号集，对set进行操作就可以了。

  返回值:

  成功返回 0，失败返回 -1。

信号的基本操作

- #include <signal.h>

  int sigfillset(sigset_t *set);

  功能:

  初始化信号集合set，将信号集合设置为所有信号的集合。

  参数:

  信号集标识的地址，以后操作此信号集，对set进行操作就可以了。

  返回值:

  成功返回 0，失败返回 -1。

信号的基本操作

- #include <signal.h>

  int sigismember(const sigset_t *set, intsignum);

  功能:

  查询signum标识的信号是否在信号集合set之中。

  参数:

  set: 信号集标识符号的地址。

  signum: 信号的编号。

  返回值:

  在信号集中返回 1，不在信号集中返回 0

  错误，返回 -1

信号的基本操作

- #include <signal.h>

  int sigaddset(sigset_t *set, int signum);

  功能:

  将信号signo加入到信号集合set之中。

  参数:

  set: 信号集标识的地址。

  signum: 信号的编号。

  返回值:

  成功返回 0，失败返回 -1。

信号的基本操作

- #include <signal.h>

  int sigdelset(sigset_t *set, int signum);

  功能:

  将signum所标识的信号从信号集合set中删除。

  参数:

  set: 信号集标识的地址。

  signum: 信号的编号。

  返回值:

  成功: 返回 0

  失败: 返回 -1

例: 06_signal_set.c

信号的基本操作

- 信号阻塞集(屏蔽集、掩码)

  每个进程都有一个阻塞集，它用来描述哪些信号递送到该进程的时候被阻塞(在信号发生时记住它，直到进程准备好时再将信号通知进程)。

  所谓阻塞并不是禁止传送信号，而是暂时信号的传送。若将被阻塞的信号从信号阻塞集中删除，且对应的信号在被阻塞时发生了，进程将会收到相应的信号。

信号的基本操作

- #include <signal.h>

  int sigprocmask(int how,

  const sigset_t *set, sigset_t *oldset);

  功能:

  检查或修改信号阻塞集，根据how指定的方法对进程的阻塞集合进行修改，新的信号阻塞集由set指定，而原先的信号阻塞集合由oldset保存。

  参数:

  how: 信号阻塞集合的修改方法。

  set: 要操作的信号集地址。

  oldset: 保存原先信号集地址。(NULL，不保存)

信号的基本操作

- how:

  SIG_BLOCK：向信号阻塞集合中添加set信号集

  SIG_UNBLOCK：从信号阻塞集合中删除set集合

  SIG_SETMASK：将信号阻塞集合设为set集合

  注：若set为NULL，则不改变信号阻塞集合，函数只把当前信号阻塞集合保存到oldset中。

- 返回值:

  成功：返回 0

  失败：返回 -1

例：06_sigprocmask.c

信号的基本操作

- signal函数只能提供简单的信号安装操作。使用signal函数处理信号比较简单，只要把要处理的信号和处理函数列出即可。
- signal函数主要用于前面32种不可靠(在Linux系统中不可靠的信号是指信号可能会丢失)、非实时信号的处理，并且不支持信号传递信息。
- Linux提供了功能更强大的sigaction函数，此函数可以用来检查和更改信号处理操作，可以支持可靠、实时信号的处理，并且支持信号传递信息。

信号的基本操作

- 可靠、不可靠信号

  从UNIX系统继承过来的信号（SIGHUP～SIGSYS）都是不可靠信号，不支持排队（多次发送相同的信号，进程可能只能收到一次，可能会丢失）。SIGRTMIN至SIGRTMAX的信号支持排队（发多少次，就可以收到多少次，不会丢失），故称为可靠信号。

- 实时、非实时信号

  可靠信号就是实时信号，非可靠信号就是非实时信号。

信号的基本操作

- 实时信号（可靠信号）的优势
  - 多个实时信号可以同时发生并传送给相应的进程。
  - 实时信号可以传参(整数或任意类型的地址)。
  - 实时信号可以保证信号传递的顺序(传递顺序和信号产生的顺序相同)。

信号的基本操作

- #include <signal.h>

  int sigqueue(pid_t pid, int sig,

  const union sigval value);

  功能:

  给指定进程发送信号。

  参数:

  pid: 进程号。

  sig: 信号的编号。

  value: 通过信号传递的参数。

信号的基本操作

- 信号传递的参数:

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

返回值:

成功返回 0，失败返回 -1

信号的基本操作

- #include <signal.h>

```
int  sigaction(int signum,
                    const struct sigaction *act,
                       struct sigaction *oldact);
```

功能:

检查或修改指定信号的设置(或同时执行这两种操作。

参数:

signum: 信号编号。

act: 新的信号设置指针 。

oldact: 旧的信号设置指针。

信号的基本操作

- sigaction函数
  - 如果act指针非空，则要改变指定信号的设置，如果oldact指针非空，则系统将此前指定信号的设置存入oldact。
  - 返回值:

    成功：0

    失败：-1

信号的基本操作

信号设置结构体:

```
struct sigaction
{
   /*旧的信号处理函数指针*/
   void (*sa_handler)(int signum) ;
   /*新的信号处理函数指针*/
   void (*sa_sigaction)(
         int signum, siginfo_t *info, void *context);
   sigset_t sa_mask;/*信号阻塞集*/
   int sa_flags;/*信号处理的方式*/
};
```

信号的基本操作

- sa_handler、sa_sigaction:

  信号处理函数指针:

  - 忽略该信号: SIG_IGN
  - 执行系统默认动作: SIG_DFL
  - 自定义信号处理函数: 处理函数名

  注: 应根据情况给sa_sigaction、sa_handler两者之一赋值。

  sa_mask:

  信号处理函数执行期间要阻塞的信号集(但不阻塞SIGKILL、SIGSTOP)。

信号的基本操作

- sa_flags：用来决定信号的行为。
  - 不指定SA_SIGINFO:

    信号的处理函数应由sa_handler设置。可完成与signal函数一样的功能。
  - 指定SA_SIGINFO:

    信号的处理函数应由sa_sigaction设置。它的功能更强大，除了获取基本的信号值外，还可以获取此信号发送者及接收者的部分信息。

例：07_sigaction_1.c　07_sigaction_2.c

信号的基本操作

- 信号处理函数:

```
void (*sa_sigaction)(int signum,
                     siginfo_t *info, void *context);
```

参数:

- signum: 信号的编号。
- info: 记录信号发送进程信息的结构体。
- context: 可以赋给指向ucontext_t类型的一个对象的指针，以引用在传递信号时被中断的接收进程或线程的上下文。

信号的基本操作

进程信息结构体路径:

/usr/include/i386-linux-gnu/bits/siginfo.h

发送进程信息的结构体: siginfo_t

接收进程或线程的上下文结构体: ucontext_t

例: 07_sigaction_3_rcv.c

07_sigaction_3_snd.c