绝迹·危言的专栏

个人资料

QQ276592716

访问： 213908次
积分： 3271
等级： BLOG 5
排名： 第5984名

原创： 95篇 转载： 164篇
译文： 0篇 评论： 54条

Linux多线程中使用信号-1

标签： 多线程 linux signal thread ubuntu kill

2012-03-06 16:07 10292人阅读 评论(8) 收藏 举报

分类： linux~（29）

在Linux的多线程中使用信号机制，与在进程中使用信号机制有着根本的区别，可以说是完全不同。在进程环境中，对信号的处理是，先注册信号处理函数，当信号异步发生时，调用处理函数来处理信号。它**完全是异步**的（我们完全不知到信号会在进程的那个执行点到来！）。然而信号处理函数的实现，有着许多的限制；比如有一些函数不能在信号处理函数中调用；再比如一些函数read、recv等调用时会被异步的信号给中断(interrupt)，因此我们必须对在这些函数在调用时因为信号而中断的情况进行处理（判断函数返回时 enno 是否等于 EINTR）。

但是在多线程中处理信号的原则却完全不同，它的基本原则是：**将对信号的异步处理，转换成同步处理**，也就是说用一个线程专门的来“同步等待”信号的到来，而其它的线程可以完全不被该信号中断/打断(interrupt)。这样就在相当程度上简化了在多线程环境中对信号的处理。而且可以保证其它的线程不受信号的影响。这样我们对信号就可以完全预测，因为它不再是异步的，而是同步的（我们完全知道信号会在哪个线程中的哪个执行点到来而被处理！）。而同步的编程模式总是比异步的编程模式简单。其实多线程相比于多进程的其中一个优点就是：多线程可以将进程中异步的东西转换成同步的来处理。

1. sigwait函数：

```
1. sigwait - wait for a signal
2.
3. #include <signal.h>
4. int sigwait(const sigset_t *set, int *sig);
5.
   Description
6. The sigwait() function suspends execution of the calling thread until the delivery of one
7. of the signals specified in the signal set set. The function accepts the signal (removes
8. it from the pending list of signals), and returns the signal number in sig.
9.
10. The operation of sigwait() is the same as sigwaitinfo(2), except that:
11.
12. * sigwait() only returns the signal number, rather than a siginfo_t structure describing
13.   the signal.
14. * The return values of the two functions are different.
15.
16. Return Value
17.
18. On success, sigwait() returns 0. On error, it returns a positive error number.
```

从上面的man sigwait的描述中，我们知道：sigwait是同步的等待信号的到来，而不是像进程中那样是异步的等待信号的到来。sigwait函数使用一个信号集作为他的参数，并且在集合中的任一个信号发生时返回该信号值，解除阻塞，然后可以针对该信号进行一些相应的处理。

2. 记住：

在多线程代码中，总是使用sigwait或者sigwaitinfo或者sigtimedwait等函数来处理信号。

而不是signal或者sigaction等函数。因为在一个线程中调用signal或者sigaction等函数会改变所以线程中的信号处理函数。而不是仅仅改变调用signal/sigaction的那个线程的信号处理函数。

3. pthread_sigmask函数：

每个线程均有自己的信号屏蔽集（信号掩码），可以使用pthread_sigmask函数来屏蔽某个线程对某些信号的响应处理，仅留下需要处理该信号的线程来处理指定的信号。实现方式是：利用线程信号屏蔽集的继承关系（在主进程中对sigmask进行设置后，主进程创建出来的线程将继承主进程的掩码）

1. pthread_sigmask - examine and change mask of blocked signals
2.
3. #include <signal.h>
4. int pthread_sigmask(int *how*, const sigset_t **set*, sigset_t **oldset*);
5.
6. Compile and link with -pthread.
7.
8. DESCRIPTION
9. The pthread_sigmask() function is just like sigprocmask(2), with the difference that **its use**
10. **in multithreaded programs** is explicitly specified by POSIX.1-2001.
11. Other differences are noted in this page.
12. For a description of the arguments and operation of this function, see sigprocmask(2).
13.
14. RETURN VALUE
15. On success, pthread_sigmask() returns 0; on error, it returns an error number.
16. NOTES
17. A new thread **inherits** a copy of its creator's signal mask.
18.
19. (from man sigprocmask:)
20.
21. The behavior of the call is dependent on the value of how, as follows.
22. SIG_BLOCK
23. The set of blocked signals is the **union** of the current set and the *set* argument.
24. SIG_UNBLOCK
25. The signals in set are removed from the current set of blocked signals. It is permissible
26. to attempt to **unblock** a signal which is not blocked.
27. SIG_SETMASK
28. The set of blocked signals is **set** to the argument *set*.
29.
30. If *oldset* is non-NULL, the previous value of the signal mask is stored in *oldset*.
31.
32. If *set* is NULL, then the signal mask is unchanged (i.e., *how* is ignored), but the current
33. value of the signal mask is nevertheless returned in *oldset* (if it is not NULL).

4. pthread_kill函数：

在多线程程序中，一个线程可以使用pthread_kill对同一个进程中指定的线程（包括自己）发送信号。注意在多线程中一般不使用kill函数发送信号，因为kill是对进程发送信号，结果是：正在运行的线程会处理该信号，如果该线程没有注册信号处理函数，那么会导致整个进程退出。

1. #include <signal.h>
2. int pthread_kill(pthread_t *thread*, int *sig*);
3.
4. Compile and link with -pthread.
5.
6. DESCRIPTION
7. The pthread_kill() function sends the signal *sig* to thread, another thread in the same
8. process as the caller. The signal is asynchronously directed to thread.

```
9.
10. If sig is 0, then no signal is sent, but error checking is still performed; this can be
11. used to check for the existence of a thread ID.
12.
13. RETURN VALUE
14. On success, pthread_kill() returns 0; on error, it returns an error number, and no signal
15. is sent.
16.
17. ERRORS
18. ESRCH No thread with the ID thread could be found.
19. EINVAL An invalid signal was specified.
```

5. 记住：调用sigwait同步等待的信号必须在调用线程中被屏蔽，并且通常应该在所有的线程中被屏蔽（这样可以保证信号绝不会被送到除了调用sigwait的任何其它线程），这是通过利用信号掩码的继承关系来达到的。

(The semantics of sigwait require that all threads (including the thread calling sigwait) have the signal masked, for reliable operation. Otherwise, a signal that arrives not blocked in sigwait **might be delivered to another thread**.)

6. 代码示例：(from man pthread_sigmask)

```
1. #include <pthread.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <unistd.h>
5. #include <signal.h>
6. #include <errno.h>
7.
8. /* Simple error handling functions */
9.
10. #define handle_error_en(en, msg) \
11.         do { errno = en; perror(msg); exit(EXIT_FAILURE); } while (0)
12.
13. static void *           sig_thread(void *arg)
14. {
15.     sigset_t *set = (sigset_t *) arg;
16.     int s, sig;
17.
18.     for (;;) {
19.         s = sigwait(set, &sig);
20.         if (s != 0)
21.             handle_error_en(s, "sigwait");
22.         printf("Signal handling thread got signal %d\n", sig);
23.     }
24. }
25.
26. int           main(int argc, char *argv[])
27. {
28.     pthread_t thread;
29.     sigset_t set;
30.     int s;
31.
32.     /*
33.        Block SIGINT; other threads created by main() will inherit
34.        a copy of the signal mask.
35.     */
36.     sigemptyset(&set);
```

```
37.     sigaddset(&set, SIGQUIT);
38.     sigaddset(&set, SIGUSR1);
39.     s = pthread_sigmask(SIG_BLOCK, &set, NULL);
40.     if (s != 0)
41.         handle_error_en(s, "pthread_sigmask");
42.     s = pthread_create(&thread, NULL, &sig_thread, (void *) &set);
43.     if (s != 0)
44.         handle_error_en(s, "pthread_create");
45.     /*
46.       Main thread carries on to create other threads and/or do
47.       other work
48.      */
49.     pause(); /* Dummy pause so we can test program */
50.     return 0;
51. }
```

编译运行情况：

```
1. digdeep@ubuntu:~/pthread/learnthread$ gcc -Wall -pthread -o pthread_sigmask pthread_sigmask.c
2. digdeep@ubuntu:~/pthread/learnthread$ ./pthread_sigmask &
3. [1] 4170
4. digdeep@ubuntu:~/pthread/learnthread$ kill -QUIT %1
5. digdeep@ubuntu:~/pthread/learnthread$ Signal handling thread got signal 3
6.
7. digdeep@ubuntu:~/pthread/learnthread$ kill -USR1 %1
8. digdeep@ubuntu:~/pthread/learnthread$ Signal handling thread got signal 10
9.
10. digdeep@ubuntu:~/pthread/learnthread$ kill -TERM %1
11. digdeep@ubuntu:~/pthread/learnthread$
12. [1]+ Terminated ./pthread_sigmask
13. digdeep@ubuntu:~/pthread/learnthread$
```

这个例子演示了：通过在主线程中阻塞一些信号，其它的线程会继承信号掩码，然后专门用一个线程使用sigwait函数来同步的处理信号，使其它的线程不受到信号的影响。

上一篇 Linux内核信号处理机制介绍
下一篇 Linux多线程中使用信号-2

我的同类文章

linux~（29）

- splice系列系统调用
- 一个写优先的读写锁实现
- 互斥锁，条件变量，信号量的一个区别（unix网络...
- unix网络编程-十五，十六章-小结
- unix网络编程-十二，十三章-小结
- unix网络编程-第十一章-小结
- unix网络编程-第十章-小结
- unix网络编程-第九章-小结
- unix网络编程-第八章-小结
- unix网络编程-第七章-小结

更多