Stephen，Yang

永久域名 http://shihaiyang.javaeye.com

tomotoboy

浏览: 11042 次

性别:

来自: 成都

我现在离线

详细资料 留言簿

文件I/O | 进程间通信——系统调用setjmp()与longjmp(...

2009-08-13

进程间通信——管道

1.匿名管道

匿名管道创建的四种方法

- 使用pipe()函数
- 使用dup()函数
- 使用dup2()函数
- 使用popen()/pclose()函数

dup()函数

有时候我们需要将子进程当中的管道的句柄定向到标准 I/O（stdin/stdout）上去。这样，在子进程中使用 exec()函数调用外部程序时，这个外部程序就会将管道作为它的输入/输出。这个过程可以用系统函数 dup()来实现。

下面是它的原型:

C代码

```
1.  int dup( int oldfd);
```

虽然原句柄和新句柄是可以互换使用的，但为了避免混淆，我们通常会将原句柄关闭（close） 。同时要注意，在 dup()函数中我们无法指定重定向的新句柄，系统将自动使用未被使用的最小的文件句柄（记住，句柄是一个整型量）作为重定向的新句柄。

dup2()函数

在 Linux 系统中还有一个系统调用函数 dup2()。单从函数名上我们也可以判断出它和 dup()函数的渊源。dup2 将用 oldfd 文件描述符来代替 newfd 文件描述符,同时关闭 newfd 文件描述符.也就是说, 所有向 newfd 操作都转到 oldfd 上面.

下面是它的原型:

C代码

```
1.  int dup2( int oldfd, int newfd );
```

注意：旧句柄将被 dup2()自动关闭。显然，原来的 close 以及 dup 这一套调用现在全部由 dup2()来完成。这样不仅简便了程序，更重要的是，它保证了操作的独立性和完整性，不会被外来的信号所中断。在原来的dup()调用中，我们必须先调用close()函数。假设此时恰好

- 全部博客 (84)
- Java (14)
- Mina (1)
- Spring (0)
- Hibernate (1)
- Linux/Unix (17)
- Oracle (5)
- Ibatis (1)
- Java Multi-thread (6)
- mysql (3)
- TCP/IP (1)
- Shell (22)
- Mobile Communication (1)
- Text Mining (0)
- SQL (6)
- Design Pattern (1)
- JSP/Servlet (2)
- EXT (1)
- Data Warehousing (1)
- ROR (1)

其他分类

- 我的收藏 (0)
- 我的论坛主题贴 (89)
- 我的所有论坛贴 (7)
- 我的精华良好贴 (0)

最近加入圈子

存档

一个信号使接下来的 dup()调用不能立即执行，这就会引发错误（进程没有了 stdin）。使用 dup2()就不会有这样的危险。

2.有名管道的I/O使用

有名管道和管道的操作是相同的，只是要注意，在引用已经存在的有名管道时，首先要用系统中的文件函数来打开它，才能接下来进行其他的操作。例如，我们可以用操作文件流的fopen()和fclose()来打开一个有名管道。下面是一个server 方的例子：

C代码

```c
/* fifoserver.c */
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <linux/stat.h>
#define FIFO_FILE  " sampleFIFO"
int main(void)
{
        FILE *fp;
         char readbuf[80];
        /* Create the FIFO if it does not exist * /
        umask(0);
        /*在文件系统中创建有名管道*/
         mknod(FIFO_FILE, S_IFIFO|0666, 0);
         while(1)
         {
                 /*打开有名管道*/
                 fp = fopen(FIFO_FILE, "r");
                 /*从有名管道中读取数据*/
                 fgets(readbuf, 80, fp);
                 printf("Received string: %s\n", readbuf);
                 /*关闭有名管道*/
                 fclose(fp);
         }
          return(0);
}
```

因为有名管道自动支持进程阻塞，所以我们可以让这个 server 在后台运行：

Shell代码

```
#fifoserver &
```

2009-11 (1)
- 2009-10 (7)
- 2009-08 (48)
- 更多存档...

最新评论

- tr用法

  thanks

  -- by tomotoboy

- tr用法

  tr "[0*4]" "*" < h ...

  -- by ibelieve

评论排行榜

- tr用法
- 后台执行命令——守护进程创建
- Berkeley套接字的一些基本知识
- TreeMap和HashMap
- 进程间通信——管道

然后运行下面的 client 程序：

C代码

```c
#include <stdio.h>
#include <stdlib.h>
#define FIFO_FILE       "sampleFIFO"
int main(int argc, char *argv[])
{
        FILE *fp;
         if ( argc != 2 ) {
                 printf("USAGE: fifoclient [string]\n");
                 exit(1);
        }
         /*打开有名管道*/
         if((fp = fopen(FIFO_FILE, "w")) == NULL) {
                 perror("fopen");
                 exit(1);
        }
         /*向有名管道中写入数据*/
         fputs(argv[1], fp);
         /*关闭有名管道*/


         fclose(fp);
         return(0);
}
```

由于有名管道的自动阻塞特性，当上面的 server 打开一个有名管道准备读入时，server进程就会被阻塞以等待其他进程（在这里是我们的 client 进程）在有名管道中写入数据。反之亦然。不过，如果需要，我们也可以在打开一个有名管道时使用 O_NONBLOCK标志来关闭它的自动阻塞特性。

未提到的关于有名管道的一些注意

首先，有名管道必须同时有读/写两个进程端。如果一个进程试图向一个没有读入端进程的有名管道写入数据，一个 SIGPIPE信号就会产生。这在涉及多个进程的有名管道通信中是很有用的。

其次，关于管道操作的独立性。一个“独立”的操作意味着，这个操作不会因为任何原因而被中断。比如，在 OSIX 标准中，头文件/usr/include/posix1_lim.h中定义了在一次独立的管道读/写操作中最大传输的数据量(buffer size)：

C代码

```c
#define _POSIX_PIPE_BUF         512
```

也即是说，在一次独立的管道读/写操作中最多只能传送 512 个字节的数据，当数据量超过这个上限时操作就只能被分成多次独立的读/写操作。在 Linux 系统中，头文件“ linux/limits.h” 中定义了类似的限制：

C代码

```c
#define PIPE_BUF                4096
```

可以看出，和 POSIX 标准比，上限被大大增加了。这在涉及多进程的有名管道操作中是非常重要的。如果在某个进程的一次写操作中传输的数据量超过了独立读/写操作的数据量上限，这个操作就有可能被别的进程的写操作打断。也就是说，别的进程把数据插入了该进程写入管道的数据序列中从而造成混乱。这是在有名管道应用中需要特别注意的。

参考资料：

《linux网络编程》李卓恒等译

文件I/O | 进程间通信——系统调用setjmp()与longjmp(...

您还没有登录，请登录后发表评论(快捷键 Alt+S / Ctrl+Enter) 提交