进程间通信之消息队列 msgget()、msgsend()、msgrcv()、msgctl()

其他 2020-04-10 14:01:17 阅读次数: 0

消息队列

1、消息队列提供了一个从一个进程向另外一个进程发送一块数据的方法
2、每个数据块都被认为是有一个类型，接收者进程接收的数据块可以有不同的类型值

3、消息队列与管道不同的是，消息队列是基于消息的，而管道是基于字节流的，且消息队列的读取不一定是先入先出。
4、消息队列也有管道一样的不足，就是每个消息的最大长度是有上限的（MSGMAX），每个消息队列的总的字节数是有上限的（MSGMNB），系统上消息队列的总数也有一个上限（MSGMNI），这三个参数都可以查看：

```
[root@/nfsroot]#
[root@/nfsroot]#cat /proc/sys/kernel/msgmax
8192
[root@/nfsroot]#
[root@/nfsroot]#cat /proc/sys/kernel/msgmnb
16384
[root@/nfsroot]#
[root@/nfsroot]#cat /proc/sys/kernel/msgmni
243
[root@/nfsroot]#
[root@/nfsroot]#
```

一、什么是消息队列

消息队列提供了一种从一个进程向另一个进程发送一个数据块的方法。 每个数据块都被认为含有一个类型，接收进程可以独立地接收含有不同类型的数据结构。我们可以通过发送消息来避免命名管道的同步和阻塞问题。但是消息队列与命名管道一样，每个数据块都有一个最大长度的限制。

Linux用宏MSGMAX和MSGMNB来限制一条消息的最大长度和一个队列的最大长度。

二、在Linux中使用消息队列

Linux提供了一系列消息队列的函数接口来让我们方便地使用它来实现进程间的通信。它的用法与其他两个System V PIC机制，即信号量和共享内存相似。

1、msgget()函数

该函数用来创建和访问一个消息队列。它的原型为：

```
int msgget(key_t, key, int msgflg);
```

与其他的IPC机制一样，程序必须提供一个键来命名某个特定的消息队列。msgflg是一个权限标志，表示消息队列的访问权限，它与文件的访问权限一样。msgflg可以与IPC_CREAT做或操作，表示当key所命名的消息队列不存在时创建一个消息队列，如果key所命名的消息队列存在时，IPC_CREAT标志会被忽略，而只返回一个标识符。

它返回一个以key命名的消息队列的标识符（非零整数），失败时返回-1.

2、msgsnd()函数

该函数用来把消息添加到消息队列中。它的原型为：

今日推荐

Fedora 32 因 Bug 将推迟发布

云办公系统 skyeye v3.1.3 发布，学校模块更新以及修复 bug

Apache Kafka 2.5.0 发布

Apache Solr 8.5.1 发布，Java 全文搜索服务器

Apache Lucene 8.5.1 发布，Java 全文搜索引擎

Wine 将提供更好的 USB 支持

Spring Cloud 2020.0.0-M1 发布

Chrome 83 Beta 发布：与 Edge 合作改进的表单控件

SeaweedFS 1.74 发布，分布式文件系统

TiDB 3.1.0 发布，分布式 NewSQL 数据库

Node.js 13.13.0 发布

LibreOffice 6.4.3 发布，开源办公套件

周排行

文件服务器配置

C#图解教程 第七章 类和继承

centos7中安装mongodb3.6

有限自动机

js的this用法

jQuery中的动画

python class(1)

Mysql->索引的维护[20180504]

网页截图的两种方式

TreeSet知识点总结

每日归档 更多

2020-04-17(6484)

2020-04-16(7024)

```
int msgsend(int msgid, const void *msg_ptr, size_t msg_sz, int msgflg);
```

msgid是由msgget函数返回的消息队列标识符。

msg_ptr是一个指向准备发送消息的指针，但是消息的数据结构却有一定的要求，指针msg_ptr所指向的消息结构一定要是以一个长整型成员变量开始的结构体，接收函数将用这个成员来确定消息的类型。所以消息结构要定义成这样：

```
struct my_message {
    long int message_type;
    /* The data you wish to transfer */
};
```

msg_sz 是msg_ptr指向的消息的长度，注意是消息的长度，而不是整个结构体的长度，也就是说msg_sz是不包括长整型消息类型成员变量的长度。

msgflg 用于控制当前消息队列满或队列消息到达系统范围的限制时将要发生的事情。

如果调用成功，消息数据的一分副本将被放到消息队列中，并返回0，失败时返回-1.

3、msgrcv()函数

该函数用来从一个消息队列获取消息，它的原型为

```
int msgrcv(int msgid, void *msg_ptr, size_t msg_st, long int msgtype, int msgflg);
```

msgid, msg_ptr, msg_st 的作用也函数msgsnd()函数的一样。

msgtype 可以实现一种简单的接收优先级。**如果msgtype为0，就获取队列中的第一个消息。**如果它的值大于零，将获取具有相同消息类型的第一个信息。如果它小于零，就获取类型等于或小于msgtype的绝对值的第一个消息。

msgflg 用于控制当队列中没有相应类型的消息可以接收时将发生的事情。

调用成功时，该函数返回放到接收缓存区中的字节数，消息被复制到由msg_ptr指向的用户分配的缓存区中，然后删除消息队列中的对应消息。失败时返回-1。

4、msgctl()函数

该函数用来控制消息队列，它与共享内存的shmctl函数相似，它的原型为：

```
int msgctl(int msgid, int command, struct msgid_ds *buf);
```

command是将要采取的动作，它可以取3个值，

- IPC_STAT：把msgid_ds结构中的数据设置为消息队列的当前关联值，即用消息队列的当前关联值覆盖msgid_ds的值。
- IPC_SET：如果进程有足够的权限，就把消息列队的当前关联值设置为msgid_ds结构中给出的值
- IPC_RMID：删除消息队列

buf是指向msgid_ds结构的指针，它指向消息队列模式和访问权限的结构。msgid_ds结构至少包括以下成员：

2020-04-15(7656)
2020-04-14(6987)
2020-04-13(7124)
2020-04-12(6585)
2020-04-11(7454)
2020-04-10(7603)
2020-04-09(6808)
2020-04-08(7024)

```
struct msgid_ds
{
    uid_t shm_perm.uid;
    uid_t shm_perm.gid;
    mode_t shm_perm.mode;
};
```

成功时返回0，失败时返回-1.

三、使用消息队列进行进程间通信

马不停蹄，介绍完消息队列的定义和可使用的接口之后，我们来看看它是怎么让进程进行通信的。由于可以让不相关的进程进行行通信，所以我们在这里将会编写两个程序，msgreceive()和msgsned()来表示接收和发送信息。根据正常的情况，我们允许两个程序都可以创建消息，但只有接收者在接收完最后一个消息之后，它才把它删除。

接收信息的程序源文件为msgreceive.c的源代码为：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/msg.h>
#include <errno.h>

struct msg_st
{
    long int msg_type;
    char text[BUFSIZ];
};

int main(int argc, char **argv)
{
    int msgid = -1;
    struct msg_st data;
    long int msgtype = 0;   // 注意1

    // 建立消息队列
    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
    if (msgid == -1)
    {
        fprintf(stderr, "msgget failed width error: %d\n", errno);
        exit(EXIT_FAILURE);
    }

    // 从队列中获取消息，直到遇到end消息为止
    while (1)
    {
        if (msgrcv(msgid, (void *)&data, BUFSIZ, msgtype, 0) == -1)
        {
            fprintf(stderr, "msgrcv failed width erro: %d", errno);
        }

        printf("You wrote: %s\n", data.text);

        // 遇到end结束
        if (strncmp(data.text, "end", 3) == 0)
        {
            break;
        }
    }

    // 删除消息队列
    if (msgctl(msgid, IPC_RMID, 0) == -1)
    {
        fprintf(stderr, "msgctl(IPC_RMID) failed\n");
    }

    exit(EXIT_SUCCESS);
}
```

发送信息的程序的源文件msgsend.c的源代码为：

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/msg.h>
#include <errno.h>

#define MAX_TEXT 512

struct msg_st
{
    long int msg_type;
    char text[MAX_TEXT];
};

int main(int argc, char **argv)
{
    struct msg_st data;
    char buffer[BUFSIZ];
    int msgid = -1;

    // 建立消息队列
    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
    if (msgid == -1)
    {
        fprintf(stderr, "msgget failed error: %d\n", errno);
        exit(EXIT_FAILURE);
    }

    // 向消息队里中写消息，直到写入end
    while (1)
    {
        printf("Enter some text: \n");
        fgets(buffer, BUFSIZ, stdin);
        data.msg_type = 1; // 注意2
        strcpy(data.text, buffer);

        // 向队列里发送数据
        if (msgsnd(msgid, (void *)&data, MAX_TEXT, 0) == -1)
        {
            fprintf(stderr, "msgsnd failed\n");
            exit(EXIT_FAILURE);
        }

        // 输入end结束输入
        if (strncmp(buffer, "end", 3) == 0)
        {
            break;
        }

        sleep(1);
    }

    exit(EXIT_SUCCESS);
}
```

运行结果如下：

```
[ljianhui@localhost CppCode]$ gcc -o msgreceive.exe msgreceive.c -lm
[ljianhui@localhost CppCode]$ gcc -o msgsend.exe msgsend.c -lm
[ljianhui@localhost CppCode]$ ./msgreceive.exe &
[1] 5539
[ljianhui@localhost CppCode]$ ./msgsend.exe
Enter some text: Linux
You wrote: Linux

Enter some text: Programming
You wrote: Programming

Enter some text: end
You wrote: end

[1]+  Done                    ./msgreceive.exe
[ljianhui@localhost CppCode]$
```


四、例子分析——消息类型

这里主要说明一下消息类型是怎么一回事，注意msgreceive.c文件main()函数中定义的变量msgtype（注释为注意1），它作为msgrcv()函数的接收信息类型参数的值，其值为0，表示获取队列中第一个可用的消息。再来看看msgsend.c文件中while循环中的语句data.msg_type = 1（注释为注意2），它用来设置发送的信息的信息类型，即其发送的信息的类型为1。所以程序msgreceive()能够接收到程序msgsend()发送的信息。

如果把注意1，即msgreceive.c文件main()函数中的语句由long int msgtype = 0;改变为long int msgtype = 2;会发生什么情况，msgreceive()将不能接收到程序msgsend()发送的信息。因为在调用msgrcv()函数时，如果msgtype（第四个参数）大于零，则将只获取具有相同消息类型的第一个消息，修改后获取的消息类型为2，而msgsend()发送的消息类型为1，所以不能被msgreceive()程序接收。重新编译msgreceive.c文件并再次执行，其结果如下：

```
[ljianhui@localhost CppCode]$ gcc -o ./msgreceive.exe ./msgreceive.c -lm
[ljianhui@localhost CppCode]$ ./msgreceive.exe &
[1] 5767
[ljianhui@localhost CppCode]$ ./msgsend.exe
Enter some text: Linux
Enter some text: Programming
Enter some text: end
[ljianhui@localhost CppCode]$ jobs
[1]+  Running                 ./msgreceive.exe &
[ljianhui@localhost CppCode]$
```


我们可以看到，msgreceive并没有接收到信息和输出，而且当msgsend输入end结束后，msgreceive也没有结束，通过jobs命令我们可以看到它还在后台运行着。

五、消息队列与命名管道的比较

消息队列跟命名管道有不少的相同之处，通过与命名管道一样，消息队列进行通信的进程可以是不相关的进程，同时它们都是通过发送和接收的方式来传递数据的。在命名管道中，发送数据用write()，接收数据用read()，则在消息队列中，发送数据用msgsnd()，接收数据用msgrcv()。而且它们对每个数据都有一个最大长度的限制。

与命名管道相比，消息队列的优势在于：

> 1、消息队列也可以独立于发送和接收进程而存在，从而消除了在同步命名管道的打开和关闭时可能产生的困难。
>
> 2、同时通过发送消息还可以避免命名管道的同步和阻塞问题，不需要由进程自己来提供同步方法。
>
> 3、接收程序可以通过消息类型有选择地接收数据，而不是像命名管道中那样，只能默认地接收。