

Linux下消息队列性能比较(SVr4, POSIX, 信号量模拟消息队列)

[日期：2011-07-22]

来源：Linux社区 作者：zldrobit

[字体：大 中 小]

比较内容：**main**中创建一个接收线程，总共发送**5*1024*1024*1024 = 5G**的数据。发送**5*1024*1024**次，每次发送**1024**字节的数据。总共进行**5**次实验。

实验数据：

SVr4 消息队列：

第一次：26.020s
第二次：26.283s
第三次：25.872s
第四次：25.857s
第五次：26.597s
平均：26.126s

POSIX 消息队列：

第一次：25.273s
第二次：26.141s
第三次：25.214s
第四次：25.240s
第五次：25.504s
平均：25.474s

信号量模拟消息队列：

第一次：23.916s
第二次：23.936s
第三次：23.973s
第四次：24.137s
第五次：23.923s
平均：23.977s

我的结论：

既然**POSIX**消息队列要比**SVr4**消息队列要快，那么就用**POSIX**好了，除非要兼容旧的系统。**POSIX**比**SVr4**快了**2.5%**左右，可能是具体实现有差异吧。

对于用信号量模拟的消息队列，主要是想验证一下手动实现的是不是会比库提供的消息队列要快很多，结果只比**POSIX**的快**6.2%**左右。

最后，俺认为一般情况下要用**POSIX**的消息队列，旧系统用**SVr4**的消息队列。实在对性能需求很强烈的话就用信号量来模拟吧。

实验程序：

SVr4:

```
1. #include "stdio.h"
```

试试看神奇的Linux公社搜索

Linux搜索

Linux教程 Ubuntu安装教程 Linux命令教程

最新资讯

- Linux编程中接收主函数返回值以及错误码提示
- Ubuntu使用fbterm无法打开fb设备的解决及fcitx-5.0.2安装教程
- Ubuntu 14.04 终端模式下中文输入 听歌
- Ubuntu 14.04下Java开发环境的搭建
- Linux下文件轻松比对，自由开源的比较软件
- XDA 指责小米违反 GPL 许可证
- Ubuntu 14.04 下安装Samba 及SSH 服务端的方法
- Ubuntu 14.04 下FTP服务器的搭建
- OpenCV使用RANSAC的仿射变换估计 estimateAffineTransform
- OpenCV 动态调节canny参数 边缘检测

本周热门

- iOS开发入门教程
- Android核心分析
- Shell for&while 循环详细总结
- Java面试题及答案（基础题120道）
- 史上最牛的Linux内核学习方法论
- ./configure,make,make install的作用
- 2015届华为校园招聘机试题
- Android游戏开发入门：贪吃蛇 源代码分析
- PyQt4 精彩实例分析
- Android教程：ImageView 设置图片

```
2. #include "stdlib.h"
3. #include "pthread.h"
4. #include "sys/types.h"
5. #include "sys/ipc.h"
6. #include "sys/msg.h"
7. #define NTIMES 5*1024*1024
8. #define HANDLE_ERROR(x) /
9.     { if ((x) == -1) { /
10.         perror(#x); /
11.         exit(EXIT_FAILURE); } }
12. struct msgbuf{
13.     long int mtype;
14.     char mtext[1024];
15. }threadbuf, buf = {
16.     1,
17.     "",
18. };
19. int qid;
20. void* threadfunc(void* para)
21. {
22.     int i;
23.     for (i = 0; i < NTIMES; i++){
24.         msgrcv(qid, &threadbuf, 1024, 0, 0);
25.     }
26.     return NULL;
27. }
28. int main()
29. {
30.     key_t key;
31.     int i;
32.     pthread_t t1;
33.     struct msqid_ds msqds;
34.     HANDLE_ERROR(key = ftok(".", 'r'));
35.     HANDLE_ERROR(qid = msgget(key, IPC_CREAT | 00700));
36.     HANDLE_ERROR(msgctl(qid, IPC_STAT, &msqds));
37.
38.     //printf("msg_qnum = %u/n", msqds.msg_qnum);
39.     //printf("msg_qbytes = %u/n", msqds.msg_qbytes);
40.     pthread_create(&t1, NULL, threadfunc, NULL);
41.     for (i = 0; i < NTIMES; i++){
42.         HANDLE_ERROR(msgsnd(qid, &buf, 1024, 0));
43.     };
44.     pthread_join(t1, NULL);
45.     HANDLE_ERROR(msgctl(qid, IPC_RMID, NULL));
46.     return 0;
47. }
```

POSIX:

```
1. #include "stdio.h"
2. #include "stdlib.h"
3. #include "pthread.h"
4. #include "fcntl.h"
5. #include "sys/stat.h"
6. #include "mqueue.h"
7. #define NTIMES 5*1024*1024
8. mqd_t mymq;
9. struct mq_attr mymqattr;
10. struct mq_attr mqattr;
11. void* threadfunc(void* para)
12. {
13.     char buf[1024];
14.     int i;
15.     for (i = 0; i < NTIMES; i++){
16.         if (mq_receive(mymq, buf, 1024, NULL) == -1){
17.             perror("mq_recvive");
18.             exit(EXIT_FAILURE);
19.         }
```

```
20.     }
21.     return NULL;
22. }
23. int main()
24. {
25.     pthread_t t1;
26.     int i;
27.     char buf[1024];
28.     mymqattr.mq_flags = 0;
29.     mymqattr.mq_maxmsg = 10;
30.     mymqattr.mq_msgsize = 1024;
31.     mymqattr.mq_curmsgs = 0;
32.     mymq = mq_open("/mymq", O_RDWR | O_CREAT | O_EXCL, S_IRWXU, &mymqattr);
33.     if (mq_getattr(mymq, &mqattr) == -1){
34.         perror("mq_getattr");
35.         exit(EXIT_FAILURE);
36.     }
37.     //printf("mq_flags = %d/n", mqattr.mq_flags);
38.     //printf("mq_maxmsg = %d/n", mqattr.mq_maxmsg);
39.     //printf("mq_msgsize = %d/n", mqattr.mq_msgsize);
40.     //printf("mq_curmsgs = %d/n", mqattr.mq_curmsgs);
41.     if (mymq == -1){
42.         perror("mq_open");
43.         exit(EXIT_FAILURE);
44.     }
45.     pthread_create(&t1, NULL, threadfunc, NULL);
46.     for (i = 0; i < NTIMES; i++){
47.         if (mq_send(mymq, buf, 1024, 0) == -1){
48.             perror("mq_send");
49.             exit(EXIT_FAILURE);
50.         }
51.     }
52.     pthread_join(t1, NULL);
53.     if (mq_close(mymq) == -1){
54.         perror("mq_close");
55.         exit(EXIT_FAILURE);
56.     }
57.     if (mq_unlink("/mymq") == -1){
58.         perror("mq_unlink");
59.         exit(EXIT_FAILURE);
60.     }
61.     return 0;
62. }
```

信号量模拟消息队列：

```
1. #include "stdio.h"
2. #include "stdlib.h"
3. #include "string.h"
4. #include "pthread.h"
5. #include "semaphore.h"
6. #include "assert.h"
7. #define MAX_QUEUE_SIZE 10
8. #define NTIMES 5*1024*1024
9. sem_t sem_empty;
10. sem_t sem_occupy;
11. char queue[MAX_QUEUE_SIZE + 1][1024];
12. int front,rear;
13. void insertQueue(char *buf, unsigned size)
14. {
15.     sem_wait(&sem_empty);
16.     memcpy(queue[front], buf, size);
17.     front = (front + 1) % (MAX_QUEUE_SIZE + 1);
18.     sem_post(&sem_occupy);
19.     assert(front != rear);
20. }
21. void deleteQueue(char *buf, unsigned size)
22. {
```

```
23.     sem_wait(&sem_occupy);
24.     assert(front != rear);
25.     memcpy(buf, queue[rear], size);
26.     rear = (rear + 1) % (MAX_QUEUE_SIZE + 1);
27.     sem_post(&sem_empty);
28. }
29. void* threadfunc(void* para)
30. {
31.     char buf[1024];
32.     int i;
33.     for (i = 0; i < NTIMES; i++){
34.         deleteQueue(buf, 1024);
35.     }
36.     return NULL;
37. }
38. int main()
39. {
40.     pthread_t t1;
41.     int i;
42.     char buf[1024];
43.     front = 0;
44.     rear = 0;
45.     sem_init(&sem_occupy, 0, 0);
46.     sem_init(&sem_empty, 0, MAX_QUEUE_SIZE);
47.     pthread_create(&t1, NULL, threadfunc, NULL);
48.     for (i = 0; i < NTIMES; i++){
49.         insertQueue(buf, 1024);
50.     }
51.     pthread_join(t1, NULL);
52.     sem_destroy(&sem_occupy);
53.     sem_destroy(&sem_empty);
54.     return 0;
55. }
```



0

[顶一下](#)

关注Linux公社（LinuxIDC.com）官方微信与QQ群，随机发放邀请码

猜您喜欢

1 2 3

- | | |
|---------------------------|-------------------------|
| Linux下模拟一个简易的消息机制 | 使用Python模拟登录QQ邮箱获取QQ好友列 |
| Linux与Kubuntu和Windows性能比较 | Linux下的IPC - 信号量的使用 |
| Linux 信号量sigprocmask使用 | Linux互斥锁、条件变量和信号量 |
| Linux 多线程编程---- 信号量的使用 | Linux信号量编程实例 |
| Linux系统下的多线程编程-条件变量&信号量 | Linux消息队列 |

百度推荐

Linux编程中接收主函数返回值以及 (今 13:50)

Linux编程基础--什么是I/O (11/20/2014 12:16:23)


Linux编程---套接字 (06/21/2014 09:16:31)

学习Linux之前需要掌握编程能力么 (12/17/2014 08:19:12)


Linux编程---时间相关 (06/21/2014 09:24:37)

Linux编程---一些系统相关的说明 (06/17/2014 19:52:43)

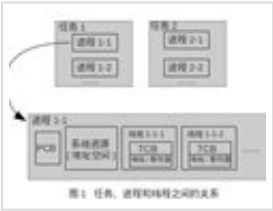
图片资讯



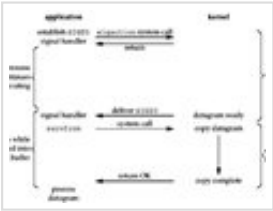
学习Linux之前需要掌




Linux编程女神计划招




Linux多任务编程




深入Linux网络编程 (



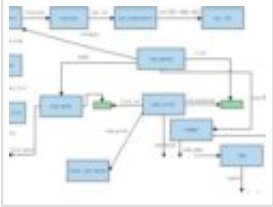
深入Linux网络编程 (



Linux程序设计-POSIX



硬实时Linux(RT-



Linux进程间通信之信

本文评论 查看全部评论 (0)

表情: 姓名:

☒ 匿名 字数 0

☒ 同意评论声明

请登录

- 评论声明
- 尊重网上道德，遵守中华人民共和国的各项有关法律法规
 - 承担一切因您的行为而直接或间接导致的民事或刑事责任
 - 本站管理人员有权保留或删除其管辖留言中的任意内容
 - 本站有权在网站内转载或引用您的评论
 - 参与本评论即表明您已经阅读并接受上述条款