中南大學
CENTRAL SOUTH UNIVERSITY

Linux 应用程序
实验报告

学生姓名	王雅蓉
学　　号	8206200602
专业班级	计科 2003
指导教师	沈海澜
学　　院	计算机学院
完成时间	2023 年 05 月 20 日

计算机学院
2023 年 05 月

目录

一 实验概述

（一）实验目的

本实验为 Linux 应用程序实验。实验主要目的如下，

(1) 掌握 Linux 平台下应用程序开发的基本过程；

(2) 掌握 Linux 进程控制相关系统调用函数的使用；

(3) 掌握 Linux 进程通信相关的系统调用函数的使用；

(4) 掌握 Linux 线程、信号、信号量相关系统调用；

（二）实验内容

编写 Linux 平台下的应用程序，完成如下各题任务，调试通过并给出运行结果。

1、有 P1，P2，P3 三个进程，P1 和 P2 负责从键盘接收字符串，均发送给 P3，P3 接收到字符串，根据发送方分别显示”P3 received *** from P1（或 P2）" ；分别用管道通信，消息队列和共享存储三种通信方式实现。

2、研究哲学家就餐问题解决方案，选定其中一种编码实现，哲学家就餐用进程或线程实现均可。

（三）实验准备及要求

1、准备知识

(1) Linux 平台下 C/C++应用程序开发的基本工具：

编译、链接：GCC 源文件或 GCC -o 可执行文件 源文件

调试工具：GDB

(2) Linux 进程控制相关系统调用：fork();wait();waitpid();kill();exit();system();exec()系列函数；

(3) Linux 进程通信的基本模式及系统调用：

无名管道通信：pipe();read();write();close()

有名管道通信：mknod();open;read();write();close()

消息队列通信：msgget();msgsnd();msgrcv();msgctl();

共享存储：shmget();shmat();shmdt()

信号：kill();alarm();signal()

信号量：semget(),semop()

(4) Linux 线程相关系统调用：pthread_creat();pthread_exit()；pthread_join();

2、实验要求

(1) 掌握 Linux 环境下应用程序的编辑、运行、调试方法

(2) 完成实验内容要求实现的任务，并记录实验过程

(3) 撰写实验报告，包括实验目的、实验内容、实验要求、实验代码及运行截图、实验心得等。

二 实验设计

（一）管道通信实现

管道通信的实现依靠两个模块，分别是 pipe_client 模块和 pipe_server 模块。其中 pipe_client 模块主要实现功能有：定义了消息结构体、监听输入生成消息、打开管道写端、将消息写入管道、检测当前输入信息是否为退出信息，如果是则退出。其中消息结构体如下。其中 pid 为发送消息的客户端进程 id，content 数组为发送消息信息数组。

```
struct message
{
int pid;
char content[4096];
};
```

其中 pipe_server 模块主要实现功能有：创建管道、打开管道读端、输出管道中的消息信息、检测该客户是否发送了退出信息。其中实时输出新消息的 while 结构如下，

```
while (read(public_fifo, &msg, sizeof(msg)))
{
if (strcmp(msg.content, "quit\n") == 0)
 {
     printf("Process %d exited.\n", msg.pid);
} else
printf("P%d received from p%d: %s",this_pid,msg.pid,msg.content);
```

（二）消息队列通信实现

消息队列通信的实现主要依靠两个模块，分别是消息队列客户端以及消息队

列服务端。消息队列客户端主要实现的功能有：定义消息正文结构、获取消息队列、获取输入、发送消息。其中消息正文格式如下，

```
struct my_msg
{
long mytype;//消息类型
int pid;
char text[256];
};
```

消息队列服务器端主要实现的功能有：定义消息正文结构、获取消息队列、接收消息、输出消息、判断客户端是否为退出消息。其中的输出消息正文如下，

```
while(1) {
 if(msgrcv(msgid,&msgbuf,sizeof(msgbuf),1,0)==-1)
{   printf("receive error");//接收失败    exit(1);   }
printf("p%d send the message %s",msgbuf.pid,msgbuf.text);
if(strcmp(msgbuf.text,"quit\n")==0)
printf("p%d has exit",msgbuf.pid);
}
```

（三）共享存储通信实现

共享存储通信实现主要有客户端模块、服务器模块实现。其中客户端模块实现的功能有：创建共享存储、共享存储空间映射、将输入的消息保存到共享存储空间内、取消映射。其中消息正文结构如下，

```
struct message
{
  int pid;
  char text[4096];
};
```

共享存储的创建函数如下，客户端与服务器端共用一个共享存储空间，故二者在创立时使用的共享存储标识符 KEY 相同。

```
bool get_memory(int &shmid)
{      if ((shmid = shmget(MEM_KEY, 27, IPC_CREAT | 0666)) < 0)
       {
         std::cout << "Failed to get shared memory.\n";
         return false;
       }
 return true;
}
```

（四）哲学家就餐问题

在哲学家就餐问题中，五位哲学家按顺序就坐在圆桌上，每位哲学家中间有一只筷子，只有当哲学家同时取到左右两边的筷子时才能进行就餐。当相邻的哲学家都想拿起筷子就会出现资源不足的问题。哲学家循环处于吃饭-冥想的状态中。只有当一位哲学家左边及右边的哲学家都在冥想时，该哲学家才能获取两只筷子并进行吃饭。该问题主要需要克服循环等待问题，就是避免所有哲学家都有一只筷子，死锁等待另一只筷子的现象。

本程序设计中采用的是第一种解法，并采用 5 个线程模拟五个哲学家。程序模拟了每个哲学家一共需要进行五次“吃”，故一共需要进行 5 次资源的获取。主要借用了信号量。为每一个筷子设置 id，从 0-4，并为其设置信号量为 1。对该信号量进行 P\V 操作，分别表示拿起和放下。同时设置了就餐人数信号量 sem_id 为 4，表示当前最多一起就餐人数为 4。信号量设置如下，

```
for (int i = 0; i < 5; i++)
{
set_semvalue(stick_sem, 1, i);
}
set_semvalue(sem_id, 4, 0);
```

1、解法一：限制就餐人数

将五个筷子视为五个共享资源，筷子是并发资源，具有排他性，因此为每个筷子设置一个信号量。限制最多允许四位哲学家同时就餐，就可以避免循环依赖的条件，因为依照抽屉原理，总是会有一位哲学家可以拿到两根筷子，所以程序可以运行下去。因此当前就餐人数可以设置 4 个信号量。

3、解法二：奇偶资源

给每一位哲学家编号，从 1 到 5, 如果我们规定奇数号的哲学家首先拿左手边的筷子，再拿右手边的筷子，偶数号的哲学家先拿右手边的筷子，再拿左手边的筷子， 释放筷子的时候按照相反的顺序，这样也可以避免出现循环依赖的情况。

4、解法三：资源分级

另一个简单的解法是为资源（这里是筷子）分配一个偏序或者分级的关系，并约定所有资源都按照这种顺序获取，按相反顺序释放，而且保证不会有两个无关资源同时被同一项工作所需要。在哲学家就餐问题中，筷子按照某种规则编号

为 1 至 5，每一个工作单元（哲学家）总是先拿起左右两边编号较低的筷子，再拿编号较高的。用完筷子后，他总是先放下编号较高的筷子，再放下编号较低的。在这种情况下，当四位哲学家同时拿起他们手边编号较低的筷子时，只有编号最高的筷子留在桌上，从而第五位哲学家就不能使用任何一只筷子了。

三 代码及运行结果

（一）管道通信

pipe_client.cpp

```
1.  struct message {
2.  int pid;
3.  char content[4096];
4.  };
5.
6.
7.  int main()
8.  {
9.
10. int public_fifo;
11. struct message msg;
12.
13.  public_fifo=open(PUBLIC,O_WRONLY);
14.  if (public_fifo == -1)
15.  {
16.  std::cout << "error\n";
17.  exit(1);
18.  }
19.
20. //写
21. while(1)
22. {
23.  memset(msg.content,0,4096);
24.  fgets(msg.content,sizeof(msg.content),stdin);
25.  msg.pid=getpid();
26.  write(public_fifo,&msg,sizeof(msg));
27.  if(strcmp(msg.content,"quit\n")==0)
28.      {
29.     break;
30.      }
31. }
```

```
32. close(public_fifo);//管道关联数-1
33. exit(0);
34. }
```

Pipe_server.cpp

```
1.  struct message {
2.  int pid;
3.  char content[4096];
4.  };
5.  int main()
6.  {
7.   int n, public_fifo;
8.     struct message msg;
9.
10.    mknod(PUBLIC, S_IFIFO | 0666, 0);
11.
12.    public_fifo = open(PUBLIC, O_RDONLY);
13.
14. if(public_fifo==-1)
15. {
16.  std::cout << "error\n";
17.        exit(1);
18. }
19.    int this_pid=getpid();
20.
21. while (read(public_fifo, &msg, sizeof(msg)))
22.    {
23.       if (strcmp(msg.content, "quit\n") == 0) {
24.          printf("Process %d exited.\n", msg.pid);
25.       } else
26.          printf("P%d received from p%d: %s",this_pid,msg.pid,msg.content);
27.
28.    }
29.    exit(0);}
```

运行结果：

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./client
hello i'm p1
quit
```

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./client
hello i'm p2
quit
```

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./server
P2824 received from p2823: hello i'm p1
P2824 received from p2825: hello i'm p2
Process 2825 exited.
Process 2823 exited.
```

（二）消息队列通信

代码：queue_client.cpp

```
#include <iostream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
struct my_msg
{
long mytype;//消息类型
int pid;
char text[256];//新哦阿培训哦走进恶魔和为恶魔
};

int main()
{
 int msgid;
 struct my_msg msgbuf;
 msgid=msgget((key_t)1234,IPC_CREAT|0666);
 if(msgid<0)
 exit(1);
 while(1)
 {
  printf("Enter some text");
  fgets(msgbuf.text,256,stdin);//消息正文
  msgbuf.mytype=1;
  msgbuf.pid=getpid();
 if(msgsnd(msgid,&msgbuf,256,0)==-1)
 {
 printf("send error");
 exit(1);
 }
 if(strcmp(msgbuf.text,"quit\n")==0)
    break;
```

```
34.
35.  }
36.  exit(0);
37. }
```

Queue_server.cpp

```
1.  #include <iostream>
2.  #include <unistd.h>
3.  #include <sys/types.h>
4.  #include <sys/ipc.h>
5.  #include <sys/msg.h>
6.  #include <string.h>
7.
8.  struct my_msg
9.  {
10. long mytype;//消息类型
11. int pid;
12. char text[256];//
13. };
14. int main()
15. {
16.  int msgid;
17.  struct my_msg msgbuf;
18.  msgid=msgget((key_t)1234,IPC_CREAT|0666);
19.  if(msgid<0)
20.  exit(1);
21.  while(1)
22.  {
23.  if(msgrcv(msgid,&msgbuf,sizeof(msgbuf),1,0)==-1)
24.  {
25.  printf("receive error");//接收失败
26.  exit(1);
27.  }
28.  printf("p%d send the message %s",msgbuf.pid,msgbuf.text);
29.  if(strcmp(msgbuf.text,"quit\n")==0)
30.     printf("p%d has exit",msgbuf.pid);
31.
32.  }
33.  if (msgctl(msgid, IPC_RMID, 0) == -1) {
34.        std::cout << "error";
35.        exit(1);
36.    }
37.  exit(0);
38. }
```

运行结果：

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./qclient
Enter some texthello i'm p1
Enter some textquit
```

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./qclient
Enter some texthello i'm p2
Enter some textquit
```

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./qserver
p3041 send the message hello i'm p1
p3042 send the message hello i'm p2
p3041 send the message quit
p3041 has exitp3042 send the message quit
```

（三）共享存储通信

代码：shm.sh

```
1.  #include <iostream>
2.  #include <unistd.h>
3.  #include <sys/types.h>
4.  #include <sys/stat.h>
5.  #include <fcntl.h>
6.  #include <cstring>
7.  #include <sys/shm.h>
8.  #define MEM_KEY 456789
9.
10.
11. struct message
12. {
13.     int pid;
14.     char text[4096];
15. };
16.
17. bool get_memory(int &shmid)
18. {
19.     if ((shmid = shmget(MEM_KEY, 27, IPC_CREAT | 0666)) < 0)
20.     {
21.         std::cout << "Failed to get shared memory.\n";
22.         return false;
23.     }
24.     return true;
25. }
```

Space_client.cpp

```cpp
#include <iostream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <cstring>
#include <sys/shm.h>
#define MEM_KEY 456789
struct message
{
    int pid;
    char text[4096];
};
bool get_memory(int &shmid)
{
    if ((shmid = shmget(MEM_KEY, 27, IPC_CREAT | 0666)) < 0)
    {
        std::cout << "Failed to get shared memory.\n";
        return false;
    }
    return true;
}
```

Space_server.cpp

```cpp
#include "shm.h"

int main()
{
    int shmid;
    int pid;
    struct message *shm = new message(), *s = NULL;

    while (1)
    {
        if (get_memory(shmid))//共享存储创建成功
        {
        /*
            if ((shm=(struct message *)shmat(shmid, NULL, 0)) < 0)
            {
                std::cout << "Failed to match shared memory.\n";
            }
      */
            //映射成功
          shm=(struct message *)shmat(shmid, NULL, 0);
```

```
21.         s=shm;
22.         while (s->pid !=0)
23.          {
24.
25.        printf("Message received from p%d: %s", s->pid, s->text);
26.             s->pid = 0;
27.             strcpy(s->text, "");
28.             s++;
29.          }
30.          if (strcmp(s->text, "quit\n") == 0)
31.             {
32.                printf("Process %d exited.\n", s->pid);
33.                break;
34.             }
35.          shmdt(shm);
36.       }
37.    }
38.
39.    printf("\n%d - finished\n", getpid());
40.
41.    exit(EXIT_SUCCESS);
42. }
```

运行结果:

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./s_client
hello i'm p1
quit

3347 - finished
```

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./s_client
hello i'm p2
quit

3338 - finished
```

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./s_server
Message received from p3347: hello i'm p1
Message received from p3338: hello i'm p2
Message received from p3338: quit
Message received from p3347: quit
```

（四）哲学家就餐问题

代码：semphore.cpp

```cpp
#include "sem_phore.h"
#include <cstring>
#include <pthread.h>
#include <mutex>
#define MAX_COUNT 5

// ./method_1 0 & ./method_1 1 & ./method_1 2 & ./method_1 3 & ./method_1 4
// 至多允许 4 个哲学家同时取左边的筷子

bool running = true;
int stick_sem = 0, sem_id = 0;
pthread_t tid[5] = {0}, finish_t = 0;

void *thread_func(void *arg);

int main(int argc, char *argv[])
{
    get_sem(stick_sem, 987456, 5);
    get_sem(sem_id, 999999, 1);
    for (int i = 0; i < 5; i++)
    {
        set_semvalue(stick_sem, 1, i);
    }
    set_semvalue(sem_id, 4, 0);

    int thread_cnt = 0;
    while (thread_cnt < 5)
    {
    int ret = pthread_create(&tid[thread_cnt], NULL, thread_func, NULL);
        if (ret != 0)
            printf("pthread_create error\n");
        thread_cnt++;
    }

    while(running);
    for (int i=0; i<5; i++) {
        if (tid[i] != finish_t)
            pthread_cancel(tid[i]);
    }
    del_semvalue(stick_sem, 5);
```

```
41.     del_semvalue(sem_id, 1);
42.     return 0;
43. }
44.
45. void *thread_func(void *arg)
46. {
47.     int id = 0, count = 0;
48.     for (int i=0; i<5; i++) {
49.         if (tid[i] == pthread_self()) {
50.             id = i;
51.             break;
52.         }
53.     }
54.     printf("Philosopher %d has entered.\n", id);
55.     while(tid[0] == 0 || tid[1] == 0 || tid[2] == 0 || tid[3] == 0 || tid[4] == 0);
56.     printf("%s - Philosopher %d is thinking.\n", show_time(), id);
57.     while (count < MAX_COUNT)
58.     {
59.         pthread_testcancel();
60.         if (!semaphore_p(sem_id, 0))
61.             exit(1);
62.         if (!semaphore_p(stick_sem, id))
63.             exit(1);
64.         if (!semaphore_p(stick_sem, (id + 1) % 5))
65.             exit(1);
66.         printf("%s - Philosopher %d is eating.\n", show_time(), id);
67.         count++;
68.         sleep(3);
69.         if (!semaphore_v(stick_sem, id))
70.             exit(1);
71.         if (!semaphore_v(stick_sem, (id + 1) % 5))
72.             exit(1);
73.         if (!semaphore_v(sem_id, 0))
74.             exit(1);
75.         printf("%s - Philosopher %d is thinking.\n", show_time(), id);
76.     }
77.     finish_t = pthread_self();
78.     running = false;
79.     return NULL;
80. }
```

```
felicia@felicia-virtual-machine:~/Linux_research/research2$ ./semphore
Philosopher 0 has entered.
Philosopher 1 has entered.
Philosopher 2 has entered.
Philosopher 3 has entered.
Philosopher 4 has entered.
18:08:36 - Philosopher 0 is thinking.
18:08:36 - Philosopher 2 is thinking.
18:08:36 - Philosopher 2 is eating.
18:08:36 - Philosopher 3 is thinking.
18:08:36 - Philosopher 4 is thinking.
18:08:36 - Philosopher 1 is thinking.
18:08:36 - Philosopher 0 is eating.
18:08:39 - Philosopher 0 is thinking.
18:08:39 - Philosopher 2 is thinking.
18:08:39 - Philosopher 4 is eating.
18:08:39 - Philosopher 1 is eating.
```

四 总结与收获

经过本次 Linux 应用程序实验，我解决了课堂上许多有待解决的问题，更加深刻的了解了管道通信、消息队列通信、共享存储通信三种通信方式，也解决了关于通信细节的相关问题。

实验中很多需要调试的地方，也是我自身经常犯错的地方。在进行哲学家就餐问题时，我也重点学习了死锁等知识，重温了操作系统中的知识。

在进行管道创建时，我也学习了 linux 系统中 tmp 目录的重要作用，并将其作为创建管道的目录，这促使我对 linux 系统的了解也更近一步。而对于哲学家就餐问题中的线程有关知识，我也比以前运用的更加熟练。

在未来的 Linux 系统与应用的学习中，我会更加深入学习课堂上没有深入讲解的内容，更注重自身知识的深度与厚度积累，积极探索，认真学习，在今后的实验中精益求精的要求自己。