

第4章 进程通信与进程同步

《计算机操作系统实验指导》

王红玲褚晓敏

内容

- Linux进程通信机制介绍
- 实验4.1 进程通信实验
- Linux进程同步介绍
- 实验4.2 进程同步实验

Linux进程通信（IPC）机制介绍

Linux IPC简介 (1)

- **管道（Pipe）及有名管道（named pipe）**：最古老的进程间通信方式，通过管道文件，将读进程和写进程连接在一起，实现两个进程之间的通信
 - 无名管道：有亲缘关系的父子进程或兄弟进程间的通信
 - 有名管道：允许无亲缘关系的任意两个进程间进行通信
- **消息队列（报文队列）**：消息队列是消息的链接表。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。
 - 消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点
 - System V消息队列、Posix消息队列
- **共享内存**：使得多个进程可以访问同一块内存空间，是最快的IPC形式
 - 往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥
 - POSIX共享内存、System V共享内存

Linux IPC简介 (2)

- **信号量 (semaphore)**：主要作为进程间以及同一进程不同线程之间的同步手段
 - POSIX信号量和System V信号量
- **套接字 (Socket)**：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。Linux支持多种类型的套接字。
- **信号 (Signal)**：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生（软中断）
 - 除用于进程间通信外，进程还可以发送信号给进程本身
 - 支持Unix早期信号语义函数`sigal()`和符合Posix.1标准的信号函数`sigaction()`

共享内存

- 允许两个不相关的进程访问同一个逻辑内存。不同进程之间共享的内存通常安排为同一段物理内存。
- 进程可以将同一段共享内存连接到它们自己的地址空间中，所有进程都可以访问共享内存中的地址。
- **优点：**
 - ①使用方便，函数的接口简单；
 - ②数据的共享是直接访问内存，加快了程序的效率；
 - ③不要求通信的进程有一定的父子关系
- **缺点：**

共享内存没有提供同步的机制，需要借助其他的手段（信号量）来进行进程间的同步

System V共享内存操作函数

- `#include <sys/shm.h>`
- 创建共享内存
 - **`int shmget(key_t key, size_t size, int shmflg);`**
- 把共享内存连接到当前进程的地址空间
 - **`void *shmat(int shm_id, const void *shm_addr, int shmflg);`**
- 将共享内存从当前进程中分离
 - **`int shmdt(const void *shmaddr);`**
- 控制共享内存
 - **`int shmctl(int shm_id, int command, struct shmid_ds *buf);`**

System V消息队列（1）

- 消息队列是由一条由消息连接而成的链表，是消息的链式队列，它保存在内核中，通过消息队列的引用标识符来访问。
- 信息被放置在一个预定义的消息结构中，进程生成的消息指明了该消息的类型，并把它放入一个由系统负责维护的消息队列中去。
- 访问消息队列的进程可以根据消息的类型，有选择地从队列中遵照FIFO原则读取特定类型的消息。

System V消息队列 (2)

```
#include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>
#include <sys/ipc.h>
key_t ftok(const char *pathname, int proj_id); //根据关键字生成标识符
int msgget(ket_t key, int msgflg); //打开或创建消息队列
int msgsnd(int msqid, struct msgbuf * msgp, size_t msgsz, int msgflg); //发送消息
int msgrcv(int msqid, struct msgbuf * msgq, size_t msgsz, long msgtype, int msgflg); //
接收消息
int msgctl ( int msgqid, int cmd, struct msqid_ds *buf ); //消息队列的属性控制
```

实验4.1 两个进程相互通信

实验4.1 实验目的和实验内容

- 实验目的

- (1) 理解进程间通信的概念和方法。
- (2) 掌握常用的Linux 进程间通信的方法。

- 实验内容

- (1) 编写C程序，使用Linux中的IPC机制，完成“石头、剪子、布”的游戏。
- (2) 修改上述程序，使之能够在网络上运行该游戏。

实验4.1 实验指导（1）

- 可以创建三个进程：一个进程为裁判进程，另外两个进程为选手进程。可将“石头、剪子、布”这三招定义为三个整型值，胜负关系为：石头>剪子>布>石头。
- 选手进程按照某种策略（例如，随机产生）出招，交给裁判进程判断大小。裁判进程将对手的出招和胜负结果通知选手。比赛可以采取多盘（如100盘）定胜负，由裁判宣布最后结果。每次出招由裁判限定时间，超时判负。
- 每盘结果可以存放在文件或其他数据结构中。比赛结束，可以打印每盘的胜负情况和总的结果。

实验4.1 实验指导（2）

- 具体的实验步骤包括：
 - 设计表示“石头、剪子、布”的数据结构，以及它们之间的大小规则；
 - 设计比赛结果的存放方式；
 - 选择IPC的方法；
 - 根据你所选择的IPC方法，创建对应的IPC资源；
 - 完成选手进程；
 - 完成裁判进程。

实验4.1 实验指导 (3)

- 使用随机数函数rand()模拟出拳信息
 - 使用时间作为种子
 - srand()
 - 时间需要岔开，否则产生的随机数会相同
 - srand((unsigned)time(0) * 3000);
 - srand((unsigned)time(NULL)* i);
- 管理消息队列命令
 - ipcs -q //查看消息队列
 - ipcrm -q //删除消息队列

实验4.1 实验指导4

- 针对实验内容（1）
 - 可以采用多种IPC机制，如管道、消息队列、共享内存
 - 在编写程序时，有两种模式
 - 编写1个C程序，然后在该程序中创建2个子进程，分别模拟裁判和2个选手，运行时只需1个shell
 - 编写3个C程序，分别模拟裁判和2个选手，运行时可打开3个shell同时运行
- 针对实验内容（2），可以使用socket()编程，编写三个C程序分别代表裁判和两个选手。

Linux进程同步

Linux进程/线程同步简介

- 互斥锁：保证资源独占
- 自旋锁：与互斥量类似，但是等待自旋锁时，进程不会释放CPU，而是一直占用CPU。
- 条件变量：等待和通知，一般与互斥锁合用
- 读写锁：与互斥锁类似，不过读写锁允许更高的并行性
- 记录锁（文件锁）：在读写锁的基础上进一步细分被锁对象的粒度
- 信号量：条件变量的升级版

Linux信号量机制

- POSIX信号量

- 有名信号量:

- 基于内存的信号量
 - 常用于多线程间的同步, 也可用于相关进程间的同步
 - 用于进行进程同步时, 需要放在进程间的共享内存区中

- 无名信号量:

- 通过IPC名字进行进程间的同步
 - 特点是把信号量值保存在文件中
 - 既可用于线程, 也可用于相关进程, 甚至是不相关的进程

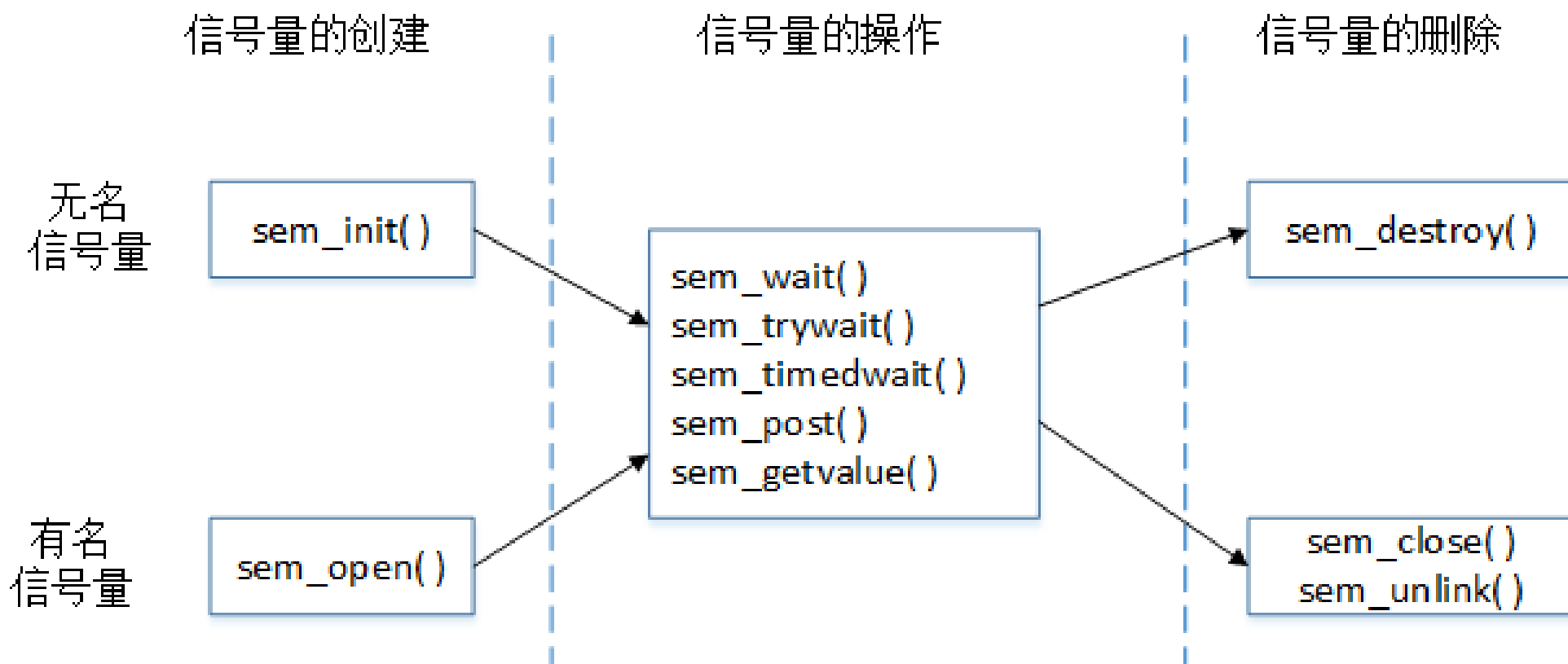
- System V信号量

- 使用相对复杂

- 在内核中维护

POSIX信号量操作函数

- `#include<semaphore.h>`



System V信号量操作

```
#include <sys/sem.h>
```

```
int semget(key_t key, int num_sems, int sem_flags);
```

```
int semctl(int sem_id, int sem_num, int cmd, union semun arg);
```

```
int semop(int sem_id, struct sembuf *sops, size_t nsops);
```

- 使用步骤:

- ① 使用semget()函数创建或获取信号量。不同进程通过使用同一个信号量键值来获得同一个信号量。
- ② 使用semctl()函数的SETVAL操作初始化信号量。
- ③ 使用semop()函数进行信号量的PV操作，这是实现进程同步或互斥的核心工作。
- ④ 如果不需要信号量，则从系统中删除它，此时使用shmctl()函数的IPC_RMID操作。

实验4.2 进程同步实验

实验4.2 实验目的

- ① 加强对进程同步和互斥的理解，学会使用信号量解决资源共享问题。
- ② 熟悉Linux 进程同步原语。
- ③ 掌握信号量wait/signal 原语的使用方法，理解信号量的定义、赋初值及wait/signal操作

实验4.2 实验内容

编写程序，使用Linux操作系统中的信号量机制模拟实现生产者-消费者问题。设有一个生产者和一个消费者，缓冲区可以存放产品，生产者不断生成产品放入缓冲区，消费者不断从缓冲区中取出产品，消费产品。

实验4.2 实验指导 (1)

- 示例代码设计：
 - 使用两个**线程**来模拟生产者和消费者
 - 使用pthread库提供的线程操作，需要包含头文件pthread.h
 - 使用POSIX的无名信号量机制，需要包含头文件semaphore.h

实验4.2 实验指导 (2)

- 向缓冲区写产品

```
buffer=(char *)malloc(MAX); //给缓冲区分配内存空间  
fgets(buffer,MAX,stdin);    //输入产品至缓冲区
```

- 从缓冲区读产品

```
printf("read product from buffer:%s",buffer); //从缓冲区中取出产品  
memset(buffer,0,MAX);                        //清空缓冲区
```

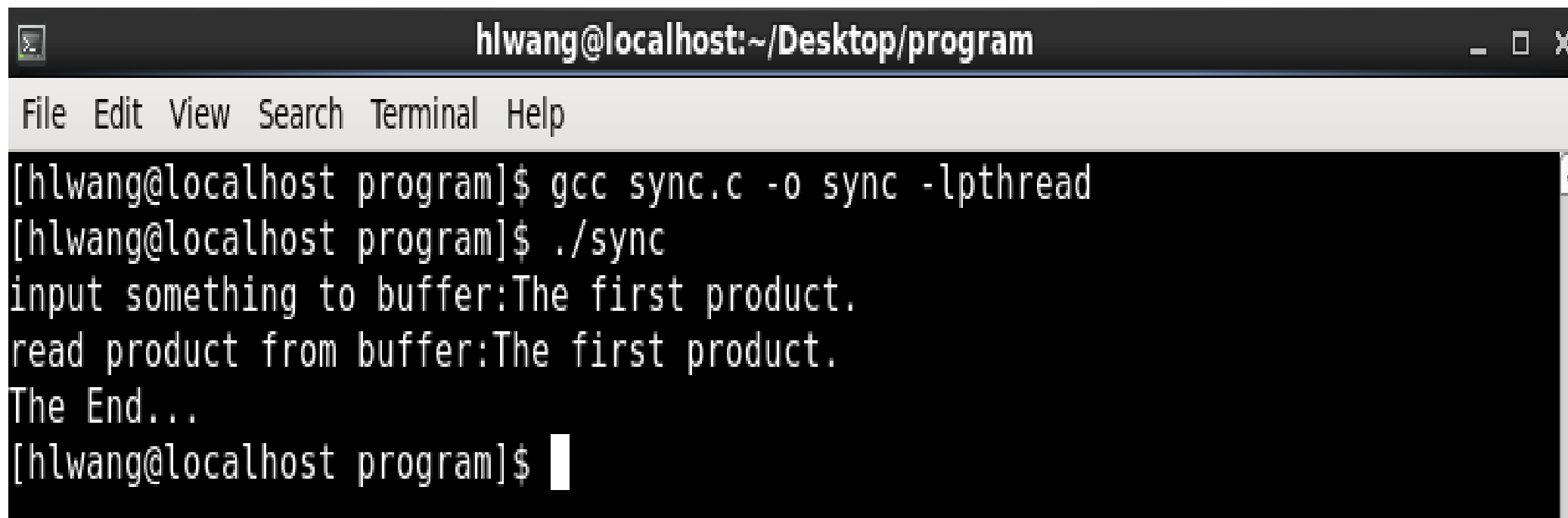
- 线程创建、等待线程结束

```
ret=pthread_create(&id_producer,NULL,producer,NULL); //创建生产者线程  
ret=pthread_create(&id_consumer,NULL,consumer,NULL); //创建消费者线程  
pthread_join(id_producer,NULL); //等待生产者线程结束  
pthread_join(id_consumer,NULL); //等待消费者线程结束
```

实验4.2 实验结果

- 注意：pthread库并非Linux操作系统的默认库，编译时需加上-lpthread选项，以调用该链接库

`gcc *.c -lpthread`



```
hlwang@localhost:~/Desktop/program
File Edit View Search Terminal Help
[hlwang@localhost program]$ gcc sync.c -o sync -lpthread
[hlwang@localhost program]$ ./sync
input something to buffer:The first product.
read product from buffer:The first product.
The End...
[hlwang@localhost program]$
```