

=====

第七课 进程通信

=====

一、基本概念

1. 何为进程间通信

~~~~~

进程间通信 (Interprocess Communication, IPC) 是指两个, 或多个进程之间进行数据交换的过程。

2. 进程间通信分类

~~~~~

- 1) 简单进程间通信: 命令行参数、环境变量、信号、文件。
- 2) 传统进程间通信: 管道(fifo/pipe)。
- 3) XSI进程间通信: 共享内存、消息队列、信号量。
- 4) 网络进程间通信: 套接字。

二、传统进程间通信——管道

- 1. 管道是Unix系统最古老的进程间通信方式。
- 2. 历史上的管道通常是指半双工管道, 只允许数据单向流动。现代系统大都提供全双工管道, 数据可以沿着管道双向流动。
- 3. 有名管道(fifo): 基于有名文件(管道文件)的管道通信。

1) 命令形式

```
# mkfifo fifo
# echo hello > fifo    # cat fifo
```

2) 编程模型

步骤	进程A	函数	进程B	步骤
1	创建管道	mkfifo	-----	
2	打开管道	open	打开管道	1
3	读写管道	read/write	读写管道	2
4	关闭管道	close	关闭管道	3
5	删除管道	unlink	-----	

范例: wfifo.c、rfifo.c

图示: fifo.bmp

4. 无名管道(pipe): 适用于父子进程之间的通信。

```
#include <unistd.h>
```

```
int pipe (int pipefd[2]);
```

- 1) 成功返回0, 失败返回-1。
- 2) 通过输出参数pipefd返回两个文件描述符, 其中pipefd[0]用于读, pipefd[1]用于写。
- 3) 一般用法
 - A. 调用该函数在内核中创建管道文件, 并通过其输出参数, 获得分别用于读和写的两个文件描述符;
 - B. 调用fork函数, 创建子进程;
 - C. 写数据的进程关闭读端(pipefd[0]), 读数据的进程关闭写端(pipefd[1]);
 - D. 传输数据;
 - E. 父子进程分别关闭自己的文件描述符。

图示: pipe.bmp

范例: pipe.c

三、XSI进程间通信

1. IPC标识

内核为每个进程间通信维护一个结构体形式的IPC对象。该对象可通过一个非负整数的IPC标识来引用。

与文件描述符不同, IPC标识在使用时会持续加1, 当达到最大值时, 向0回转。

2. IPC键值

IPC标识是IPC对象的内部名称。若多个进程需要在同一个IPC对象上会合, 则必须通过键值作为其外部名称来引用该对象。

- 1) 无论何时, 只要创建IPC对象, 就必须指定一个键值。
- 2) 键值的数据类型在sys/types.h头文件中被定义为key_t, 其原始类型就是长整型。

3. 客户机进程与服务器进程在IPC对象上的三种会合方式

- 1) 服务器进程以IPC_PRIVATE为键值创建一个新的IPC对象，并将该IPC对象的标识存放在某处(如文件中)，以方便客户机进程读取。
- 2) 在一个公共头文件中，定义一个客户机进程和服务器进程都认可的键值，服务器进程用此键值创建IPC对象，客户机进程用此键值获取该IPC对象。
- 3) 客户机进程和服务器进程，事先约定好一个路径名和一个项目ID(0-255)，二者通过ftok函数，将该路径名和项目ID转换为一致的键值。

```
#include <sys/types.h>
#include <sys/ipc.h>
```

```
key_t ftok (const char* pathname, int proj_id);
```

pathname - 一个真实存在的文件或目录的路径名。

proj_id - 项目ID，仅低8位有效，其值域为[0, 255]。

成功返回键值，失败返回-1。

注意：起作用的是pathname参数所表示的路径，而非pathname字符串本身。
因此假设当前目录是/home/soft01/uc/day07，则
ftok (".", 100);
和
ftok ("/home/soft01/uc/day07", 100);
的返回值完全相同。

4. IPC对象的创建

- 1) 若以IPC_PRIVATE为键值创建IPC对象，则永远创建成功。
- 2) 若所指定的键值在系统范围内未与任何IPC对象相结合，且创建标志包含IPC_CREAT位，则创建成功。
- 3) 若所指定的键值在系统范围内已与某个IPC对象相结合，且创建标志包含IPC_CREAT和IPC_EXCL位，则创建失败。

5. IPC对象的销毁/控制

IPC_STAT - 获取IPC对象属性
IPC_SET - 设置IPC对象属性
IPC_RMID - 删除IPC对象

四、共享内存

1. 基本特点

- ~~~~~
- 1) 两个或者更多进程，
共享同一块由系统内核负责维护的内存区域，
其地址空间通常被映射到堆和栈之间。

图示：shm.bmp

- 2) 无需复制信息，最快的一种IPC机制。
- 3) 需要考虑同步访问的问题。
- 4) 内核为每个共享内存，
维护一个shm_id_ds结构体形式的共享内存对象。

2. 常用函数

~~~~~

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

### 1) 创建/获取共享内存

```
int shmget (key_t key, size_t size, int shmflg);
```

- A. 该函数以key参数为键值创建共享内存，  
或获取已有的共享内存。
- B. size参数为共享内存的字节数，  
建议取内存页字节数(4096)的整数倍。  
若希望创建共享内存，则必需指定size参数。  
若只为获取已有的共享内存，则size参数可取0。
- C. shmflg取值：
  - 0            - 获取，不存在即失败。
  - IPC\_CREAT - 创建，不存在即创建，  
              已存在即获取，除非...
  - IPC\_EXCL    - 排斥，已存在即失败。
- D. 成功返回共享内存标识，失败返回-1。

### 2) 加载共享内存

```
void* shmat (int shm_id, const void* shmaddr,  
            int shmflg);
```

- A. 将shm\_id参数所标识的共享内存，  
映射到调用进程的地址空间。
- B. 可通过shmaddr参数人为指定映射地址，  
也可将该参数置NULL，由系统自动选择。
- C. shmflg取值：

0 - 以读写方式使用共享内存。

SHM\_RDONLY - 以只读方式使用共享内存。

SHM\_RND - 只在shmaddr参数非NULL时起作用。  
表示对该参数向下取内存页的整数倍，  
作为映射地址。

D. 成功返回映射地址，失败返回-1。

E. 内核将该共享内存的加载计数加1。

### 3) 卸载共享内存

```
int shmdt (const void* shmaddr);
```

A. 从调用进程的地址空间中，  
取消由shmaddr参数所指向的，共享内存映射区域。

B. 成功返回0，失败返回-1。

C. 内核将该共享内存的加载计数减1。

### 4) 销毁/控制共享内存

```
int shmctl (int shmid, int cmd, struct shmid_ds* buf);
```

```
struct shmid_ds {
    struct ipc_perm shm_perm;    // 所有者及其权限
    size_t          shm_segsz;   // 大小(以字节为单位)
    time_t          shm_atime;   // 最后加载时间
    time_t          shm_dtime;   // 最后卸载时间
    time_t          shm_ctime;   // 最后改变时间
    pid_t           shm_cpid;    // 创建进程PID
    pid_t           shm_lpid;    // 最后加载/卸载进程PID
    shmatt_t        shm_nattch;  // 当前加载计数
    ...
};
```

```
struct ipc_perm {
    key_t    __key; // 键值
    uid_t    uid;   // 有效属主ID
    gid_t    gid;   // 有效属组ID
    uid_t    cuid;  // 有效创建者ID
    gid_t    cgid;  // 有效创建组ID
    unsigned short mode; // 权限字
    unsigned short __seq; // 序列号
};
```

A. cmd取值:

IPC\_STAT - 获取共享内存的属性，通过buf参数输出。

IPC\_SET - 设置共享内存的属性，通过buf参数输入，  
仅以下三个属性可设置:

```
shm_id_ds::shm_perm.uid
shm_id_ds::shm_perm.gid
shm_id_ds::shm_perm.mode
```

IPC\_RMID - 标记删除共享内存。  
并非真正删除共享内存，只是做一个删除标记，  
禁止其被继续加载，但已有加载依然保留。  
只有当该共享内存的加载计数为0时，  
才真正被删除。

B. 成功返回0，失败返回-1。

3. 编程模型

| 步骤 | 进程A    | 函数     | 进程B    | 步骤 |
|----|--------|--------|--------|----|
| 1  | 创建共享内存 | shmget | 获取共享内存 | 1  |
| 2  | 加载共享内存 | shmat  | 加载共享内存 | 2  |
| 3  | 使用共享内存 | ...    | 使用共享内存 | 3  |
| 4  | 卸载共享内存 | shmdt  | 卸载共享内存 | 4  |
| 5  | 销毁共享内存 | shmctl | -----  |    |

范例：wshm.c、rshm.c

五、消息队列

1. 基本特点

- 1) 消息队列是一个由系统内核负责存储和管理，  
并通过消息队列标识引用的数据链表。
- 2) 可以通过msgget函数创建一个新的消息队列，  
或获取一个已有的消息队列。  
通过msgsnd函数向消息队列的后端追加消息，  
通过msgrcv函数从消息队列的前端提取消息。
- 3) 消息队列中的每个消息单元除包含消息数据外，  
还包含消息类型和数据长度。
- 4) 内核为每个消息队列，  
维护一个mq\_id\_ds结构体形式的消息队列对象。

2. 常用函数

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

1) 创建/获取消息队列

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

A. 该函数以key参数为键值创建消息队列，或获取已有的消息队列。

B. msgflg取值：

0 - 获取，不存在即失败。

IPC\_CREAT - 创建，不存在即创建，已存在即获取，除非...

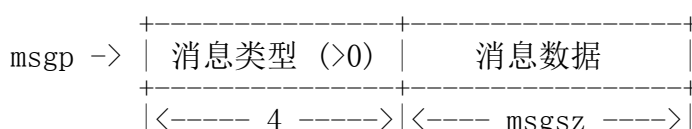
IPC\_EXCL - 排斥，已存在即失败。

C. 成功返回消息队列标识，失败返回-1。

## 2) 向消息队列发送消息

```
int msgsnd (int msqid, const void* msgp,
            size_t msgsz, int msgflg);
```

A. msgp参数指向一个包含消息类型和消息数据的内存块。  
该内存块的前4个字节必须是一个大于0的整数，代表消息类型，其后紧跟消息数据。  
消息数据的字节长度用msgsz参数表示。



注意：msgsz参数并不包含消息类型的字节数(4)。

B. 若内核中的消息队列缓冲区有足够的空闲空间，则此函数会将消息拷入该缓冲区并立即返回0，表示发送成功，否则此函数会阻塞，直到内核中的消息队列缓冲区有足够的空闲空间为止(比如有消息被接收)。

C. 若msgflg参数包含IPC\_NOWAIT位，则当内核中的消息队列缓冲区没有足够的空闲空间时，此函数不会阻塞，而是返回-1，errno为EAGAIN。

D. 成功返回0，失败返回-1。

## 3) 从消息队列接收消息

```
ssize_t msgrcv (int msqid, void* msgp, size_t msgsz,
               long msgtyp, int msgflg);
```

A. msgp参数指向一个包含消息类型(4字节)，和消息数据的内存块，其中消息数据缓冲区的字节大小用msgsz参数表示。

B. 若所接收到的消息数据字节数大于msgsz参数，即消息太长，且msgflg参数包含MSG\_NOERROR位，则该消息被截取msgsz字节返回，剩余部分被丢弃。

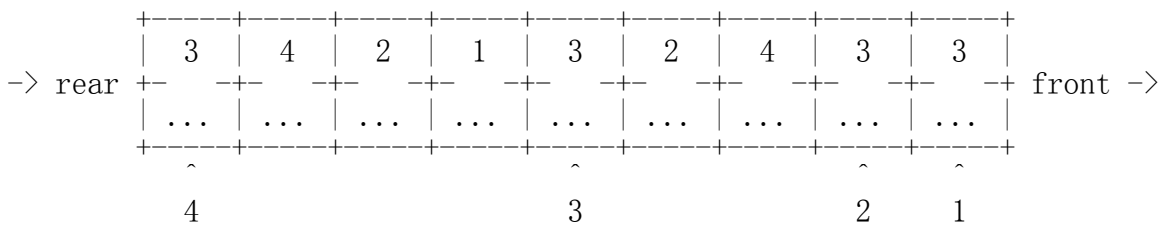
C. 若msgflg参数不包含MSG\_NOERROR位，消息又太长，  
则不对该消息做任何处理，直接返回-1，errno为E2BIG。

D. msgtyp参数表示期望接收哪类消息：

=0 - 返回消息队列中的第一条消息。

>0 - 若msgflg参数不包含MSG\_EXCEPT位，  
则返回消息队列中第一个类型为msgtyp的消息；  
若msgflg参数包含MSG\_EXCEPT位，  
则返回消息队列中第一个类型不为msgtyp的消息。

<0 - 返回消息队列中类型小于等于msgtyp的绝对值的消息。  
若有多个，则取类型最小者。



msgrcv (... , ... , ... , 3, ... );

E. 若消息队列中有可接收消息，  
则此函数会将该消息移出消息队列并立即返回0，  
表示接收成功，否则此函数会阻塞，  
直到消息队列中有可接收消息为止。

F. 若msgflg参数包含IPC\_NOWAIT位，  
则当消息队列中没有可接收消息时，此函数不会阻塞，  
而是返回-1，errno为ENOMSG。

G. 成功返回所接收到的消息数据的字节数，失败返回-1。

#### 4) 销毁/控制消息队列

int msgctl (int msqid, int cmd, struct msqid\_ds\* buf);

```
struct msqid_ds {
    struct ipc_perm msg_perm;    // 权限信息
    time_t          msg_stime;   // 随后发送时间
    time_t          msg_rtime;   // 最后接收时间
    time_t          msg_ctime;   // 最后改变时间
    unsigned long   __msg_cbytes; // 消息队列中的字节数
    msgqnum_t       msg_qnum;    // 消息队列中的消息数
    msglen_t        msg_qbytes;  // 消息队列能容纳的最大字节数
    pid_t           msg_lspid;    // 最后发送进程PID
    pid_t           msg_lrpid;    // 最后接收进程PID
};
```

```
struct ipc_perm {
    key_t   __key; // 键值
    uid_t   uid;   // 有效属主ID
    gid_t   gid;   // 有效属组ID
};
```



```
uid_t      cuid; // 有效创建者ID
gid_t      cgid; // 有效创建组ID
unsigned short mode; // 权限字
unsigned short __seq; // 序列号
};
```

A. cmd取值:

IPC\_STAT - 获取消息队列的属性, 通过buf参数输出。

IPC\_SET - 设置消息队列的属性, 通过buf参数输入,  
仅以下四个属性可设置:

```
msqid_ds::msg_perm.uid
msqid_ds::msg_perm.gid
msqid_ds::msg_perm.mode
msqid_ds::msg_qbytes
```

IPC\_RMID - 立即删除消息队列。  
此时所有阻塞在该消息队列的,  
msgsnd和msgrcv函数调用,  
都会立即返回失败, errno为EIDRM。

B. 成功返回0, 失败返回-1。

### 3. 编程模型

| 步骤 | 进程A    | 函数            | 进程B    | 步骤 |
|----|--------|---------------|--------|----|
| 1  | 创建消息队列 | msgget        | 获取消息队列 | 1  |
| 2  | 发送接收消息 | msgsnd/msgrcv | 发送接收消息 | 2  |
| 3  | 销毁消息队列 | msgctl        | ----   |    |

范例: wmsq.c、rmsq.c

练习: 基于消息队列的本地银行。

代码: bank/

## 六、信号量

### 1. 基本特点

- 1) 计数器, 用于限制多个进程对有限共享资源的访问。
- 2) 多个进程获取有限共享资源的操作模式

A. 测试控制该资源的信号量;

B. 若信号量大于0, 则进程可以使用该资源,  
为了表示此进程已获得该资源, 需将信号量减1;

- C. 若信号量等于0，则进程休眠等待该资源，直到信号量大于0，进程被唤醒，执行步骤A；
- D. 当某进程不再使用该资源时，信号量增1，正在休眠等待该资源的其它进程将被唤醒。

## 2. 常用函数

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

### 1) 创建/获取信号量

```
int semget (key_t key, int nsems, int semflg);
```

- A. 该函数以key参数为键值创建一个信号量集合 (nsems参数表示集合中的信号量数)，或获取已有的信号量集合 (nsems取0)。

B. semflg取值：

0            - 获取，不存在即失败。

IPC\_CREAT - 创建，不存在即创建，已存在即获取，除非...

IPC\_EXCL - 排斥，已存在即失败。

C. 成功返回信号量集合标识，失败返回-1。

### 2) 操作信号量

```
int semop (int semid, struct sembuf* sops, unsigned nsops);
```

```
struct sembuf {
    unsigned short sem_num; // 信号量下标
    short          sem_op;  // 操作数
    short          sem_flg; // 操作标记
};
```

- A. 该函数对semid参数所标识的信号量集合中，由sops参数所指向的包含nsops个元素的，结构体数组中的每个元素，依次执行如下操作：

- a) 若sem\_op大于0，则将其加到第sem\_num个信号量的计数值上，以表示对资源的释放；
- b) 若sem\_op小于0，则从第sem\_num个信号量的计数值中减去其绝对值，以表示对资源的获取；
- c) 若第sem\_num个信号量的计数值不够减(信号量不能为负)，则此函数会阻塞，直到该信号量够减为止，

以表示对资源的等待；

- d) 若sem\_flg包含IPC\_NOWAIT位，  
则当第sem\_num个信号量的计数值不够减时，  
此函数不会阻塞，而是返回-1，errno为EAGAIN，  
以便在等待资源的同时还可做其它处理；
- e) 若sem\_op等于0，  
则直到第sem\_num个信号量的计数值为0时才返回，  
除非sem\_flg包含IPC\_NOWAIT位。

B. 成功返回0，失败返回-1。

### 3) 销毁/控制信号量

```
int semctl (int semid, int semnum, int cmd);
int semctl (int semid, int semnum, int cmd,
            union semun arg);

union semun {
    int          val;      // Value for SETVAL
    struct semid_ds* buf;   // Buffer for IPC_STAT, IPC_SET
    unsigned short* array; // Array for GETALL, SETALL
    struct seminfo* __buf;  // Buffer for IPC_INFO
};

struct semid_ds {
    struct ipc_perm sem_perm; // Ownership and permissions
    time_t          sem_otime; // Last semop time
    time_t          sem_ctime; // Last change time
    unsigned short  sem_nsems; // No. of semaphores in set
};

struct ipc_perm {
    key_t    __key; // 键值
    uid_t    uid;   // 有效属主ID
    gid_t    gid;   // 有效属组ID
    uid_t    cuid;  // 有效创建者ID
    gid_t    cgid;  // 有效创建组ID
    unsigned short mode; // 权限字
    unsigned short __seq; // 序列号
};
```

A. cmd取值：

IPC\_STAT - 获取信号量集合的属性，通过arg.buf输出。

IPC\_SET - 设置信号量集合的属性，通过arg.buf输入，  
仅以下四个属性可设置：

```
semid_ds::sem_perm.uid
semid_ds::sem_perm.gid
semid_ds::sem_perm.mode
```

IPC\_RMID - 立即删除信号量集合。  
此时所有阻塞在对该信号量集合的，

semop函数调用，都会立即返回失败，  
errno为EIDRM。

- GETALL - 获取信号量集合中每个信号量的计数值，  
通过arg.array输出。
- SETALL - 设置信号量集合中每个信号量的计数值，  
通过arg.array输入。
- GETVAL - 获取信号量集合中，  
第semnum个信号量的计数值，  
通过返回值输出。
- SETVAL - 设置信号量集合中，  
第semnum个信号量的计数值，  
通过arg.val输入。

注意：只有针对信号量集合中具体某个信号量的操作，  
才会使用semnum参数。针对整个信号量集合的操作，  
会忽略semnum参数。

B. 成功返回值因cmd而异，失败返回-1。

### 3. 编程模型

| 步骤 | 进程A   | 函数     | 进程B   | 步骤 |
|----|-------|--------|-------|----|
| 1  | 创建信号量 | semget | 获取信号量 | 1  |
| 2  | 初始信号量 | semctl | ----- |    |
| 3  | 加减信号量 | semop  | 加减信号量 | 2  |
| 4  | 销毁信号量 | semctl | ----- |    |

范例：csem.c、gsem.c

## 七、IPC命令

### 1. 显示

ipcs -m - 显示共享内存(m: memory)  
ipcs -q - 显示消息队列(q: queue)  
ipcs -s - 显示信号量(s: semaphore)  
ipcs -a - 显示所有IPC对象(a: all)

### 2. 删除

ipcrm -m ID - 删除共享内存  
ipcrm -q ID - 删除消息队列  
ipcrm -s ID - 删除信号量