

1.System V IPC进程间通信

- 1.1IPC进程间通信基础知识
 - 1.1.1IPC进程间通信的种类
 - 1.1.2IPC进程间通信相关命令
 - 1.1.3IPC进程间通信键值获取及组成
- 1.2消息队列
 - 1.2.1消息队列通信原理
 - 1.2.2消息队列的API（msgget|msgsnd|msgrcv|msgctl）
 - 1.2.3消息队列通信实例
 - 01snd.c
 - 02rcv.c
 - 1.2.4msgctl函数详解
- 1.3共享内存
 - 1.3.1共享内存的工作原理
 - 1.3.2共享内存的API（shmget|shmat|shmdt|shmctl）
 - 1.3.3共享内存的实例
 - 01write.c
 - 02read.c
 - 1.3.4shmctl函数详解
- 1.4信号灯集
 - 1.4.1信号灯集工作原理
 - 1.4.2信号灯集的API（semget|semctl|semop）
 - 1.4.3信号灯集的实例
 - 01write.c
 - 02read.c

2.总结

1.System V IPC进程间通信

1.1IPC进程间通信基础知识

1.1.1IPC进程间通信的种类

- 1. 消息队列
- 2. 共享内存
- 3. 信号灯集（信号量）

1.1.2IPC进程间通信相关命令

(1) 查看IPC进程间通信ipcs

```
1      
```

ipcs -q //查看消息队列

ipcs -m //查看共享内存

ipcs -s //查看信号灯集

(2) 删除IPC进程间通信ipcrm

ipcrm -q msqid //删除消息队列

ipcrm -m shmid //删除共享内存

ipcrm -s semid //删除信号灯集

1.1.3IPC进程间通信键值获取及组成

在使用IPC进程间通信的时候，首先需要获取一个key，
只有当两个进程拿到相同的键的之后才能找到同一个IPC。

```
1  #include <sys/types.h>
2  #include <sys/ipc.h>
3
4  key_t ftok(const char *pathname, int proj_id);
5  功能： 获取一个键值（键不是唯一的）
6  参数：
7      @pathname: 路径及名字
8      @proj_id: 只有低8bit有效
9  返回值： 成功返回键值，失败返回-1置位错误码
```

```
1  #include <head.h>
2
3  int main(int argc, const char* argv[])
4  {
5      key_t key;
6      struct stat st;
7
```

```
8     if ((key = ftok("/home/linux", 't')) == -1)
9         PRINT_ERR("ftok error");
10
11     printf("key = %x\n", key);
12
13     if (stat("/home/linux", &st))
14         PRINT_ERR("stat error");
15
16     printf("inode = %#lx,devno = %#lx,proj_id = %x\n",st.st_ino,st.st_dev,'t');
17     return 0;
18 }
```

•linux@ubuntu:~/work/day8\$./a.out

key = 0x74012316

inode = 0x1c2316,devno = 0x801,proj_id = 0x74

◦linux@ubuntu:~/work/day8\$

key(32bit) = proj_id(8) + devno(8) +inode(16)

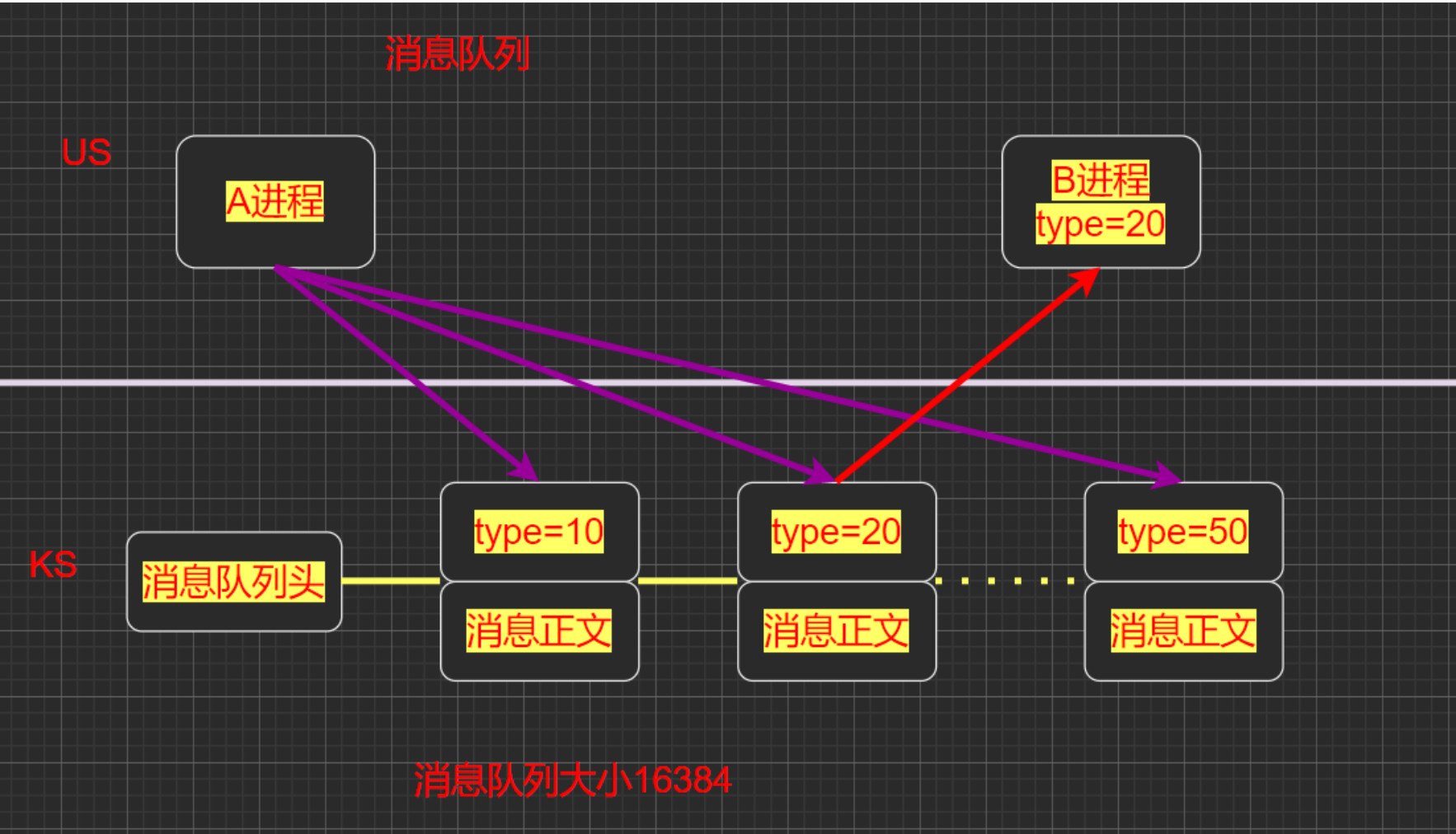
1.2消息队列

1.2.1消息队列通信原理

如果想要使用消息队列实现进程间通信，就必须在内核空间创建出来消息队列，消息队列默认大小是16384（16K），当创建好消息队列之后A进程可以向消息队列中发消息，消息的格式是类型+正文。当消息队列满的时候A进程如果还想往消息队列中发消息A进程休眠。B进程可以通过消息的类型从消息队列中取消息，取出的消息从队列中移除。如果B进程想要获取的消息类型在队列中不存在B进程休眠等。

A进程向消息队列中发消息的时候可以采用：阻塞，非阻塞方式

B进程从消息队列中收消息的时候可以采用：阻塞，非阻塞方式



1.2.2消息队列的API（msgget | msgsnd | msgrcv | msgctl）

```
1 #include <sys/ipc.h>
2 #include <sys/msg.h>
3 int msgget(key_t key, int msgflg);
4 功能：创建消息队列
5 参数：
6     @key: 键值
```

```
7      key:通过ftok获取
8      IPC_PRIVATE: 只能用于亲缘间进程的通信
9      @msgflag: 消息队列的标志位
10     IPC_CREAT|0666 或 IPC_CREAT|IPC_EXCL|0666
11 返回值: 成功返回消息队列号, 失败返回-1置位错误码
12
13 int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
14 功能: 向消息队列中发消息
15 参数:
16     @msqid:消息队列号
17     @msgp:消息的首地址
18     struct msgbuf {
19         long mtype;      //消息的类型, 必须大于0
20         char mtext[255]; //消息的正文
21     };
22     @msgsz:消息正文的大小
23     @msgflg:消息的标志
24         0: 阻塞发送
25         IPC_NOWAIT: 非阻塞发送
26 返回值: 成功返回0, 失败返回-1置位错误码
27
28 ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,int msgflg);
29 功能: 从消息队列获取消息
30 参数:
31     @msqid: 消息队列号
32     @msgp: 消息的首地址
33     @msgsz: 消息正文的大小
34     @msgtyp: 消息的类型
35         如果=0, 接收消息队列中的第一个消息
36         如果>0 ,接收msgtyp指定的消息类型
37         如果<0, 那么将读取队列中第一个最小类型小于或等于msgtyp绝对值的消息。
38         2-3-100-500-30-2000
39         -100===>100
40         2-3-100-30
41     @msgflg: 消息的标志
42         0: 阻塞接收
43         IPC_NOWAIT: 非阻塞接收
44 返回值: 失败返回-1置位错误码。成功返回接收接收的字节个数
45
```

1.2.3消息队列通信实例

01snd.c

```
1  #include <head.h>
2  typedef struct {
3      long mtype;
4      char name[19];
5      char sex;
6      int age;
7  } msg_t;
8
9  #define MSGSIZE  (sizeof(msg_t)-sizeof(long))
10
11 int main(int argc, const char* argv[])
12 {
13     key_t key;
14     int msgqid;
15     // 1.获取键值
16     if ((key = ftok("/home/linux/", 'p')) == -1)
17         PRINT_ERR("ftok error");
18
19     // 2.创建消息队列
20     if ((msgqid = msgget(key, IPC_CREAT | 0666)) == -1)
21         PRINT_ERR("msgget error");
22
23     // 3.向消息队列中发消息
24     int ret;
25     msg_t msg;
26     while (1) {
27         retry:
28         printf("input (type name sex age) > ");
29         ret = scanf("%ld %s %c %d", &msg.mtype, msg.name, &msg.sex, &msg.age);
30         if (ret != 4) {
31             printf("input error,try again\n");
32             while (getchar() != '\n');
33             goto retry;
34         }
35
36         if(msgsnd(msgqid, &msg,MSGSIZE, 0))
37             PRINT_ERR("msgsnd error");
38     }
```

```
39         if(msg.mtype == 1000) break;
40     }
41
42     return 0;
43 }
```

02rcv.c

```
1  #include <head.h>
2  typedef struct {
3      long mtype;
4      char name[19];
5      char sex;
6      int age;
7  } msg_t;
8
9  #define MSGSIZE  (sizeof(msg_t)-sizeof(long))
10
11 int main(int argc, const char* argv[])
12 {
13     key_t key;
14     int msgqid;
15     // 1.获取键值
16     if ((key = ftok("/home/linux/", 'p')) == -1)
17         PRINT_ERR("ftok error");
18
19     // 2.创建消息队列
20     if((msgqid = msgget(key,IPC_CREAT|0666))==-1)
21         PRINT_ERR("msgget error");
22
23     // 3.接收消息
24     long type;
25     msg_t msg;
26     while(1){
27         printf("input (type) > ");
28         scanf("%ld",&type);
29
30         if(msgrcv(msgqid,&msg,MSGSIZE,type,0)==-1)
31             PRINT_ERR("msgrcv error");
32
33         printf("type=%ld,name=%s,sex=%c,age=%d\n",msg.mtype,msg.name,msg.sex,msg.age);
34         if(msg.mtype == 1000) break;
35     }
36     if(msgctl(msgqid,IPC_RMID,NULL)) //删除消息队列
37         PRINT_ERR("msgctl error");
38     return 0;
39 }
```

1.2.4msgctl函数详解

```
1  int msgctl(int msqid, int cmd, struct msqid_ds *buf);
2  功能：消息队列的控制
3  参数：
4      @msqid: 消息队列号
5      @cmd: 命令码
6          IPC_STAT: 获取消息队列的属性(ipcs -q msqid)
7          IPC_SET: 设置消息队列的属性(如设置消息队列大小)
8          IPC_RMID:立即删除消息队列，唤醒所有等待的读取器和写入（ipcrm -q msqid）
9          器进程(返回一个错误并将errno设置为EIDRM)。调用进程
10         必须具有适当的特权，或者它的有效用户ID必须是消息队
11         列的创建者或所有者的ID。在这种情况下，msgctl()的/*第
12         三个参数将被忽略*/。
13     @buf: msqid_ds消息队列属性结构体
14 返回值：成功返回0，失败返回-1，置位错误码
15
16 eg1:使用msgctl删除消息队列
17     msgctl(msqid,IPC_RMID,NULL);
18 eg2:获取消息队列属性
19     struct msqid_ds msqds;
20     msgctl(msqid, IPC_STAT, &msqds); //获取到的属性在msqds结构体中存放
21
22     //以下是对msqid_ds结构体的详解
23     struct msqid_ds
24     {
25         struct ipc_perm msg_perm;    //权限结构体
26         __time_t msg_stime;          //最后一次发送消息的时间
27         __time_t msg_rtime;          //最后一次接收消息的时间
28         __syscall_ulong_t __msg_cbytes;//当前消息队列中字节数
29         msgqnum_t msg_qnum;          //当前消息队列中消息的个数
30         msglen_t msg_qbytes;          //消息队列中能够容纳的字节数（16384）
31         __pid_t msg_lspid;            //最后一次发送消息的进程号
32         __pid_t msg_lrpid;            //最后一次接收消息的进程号
33     };
```

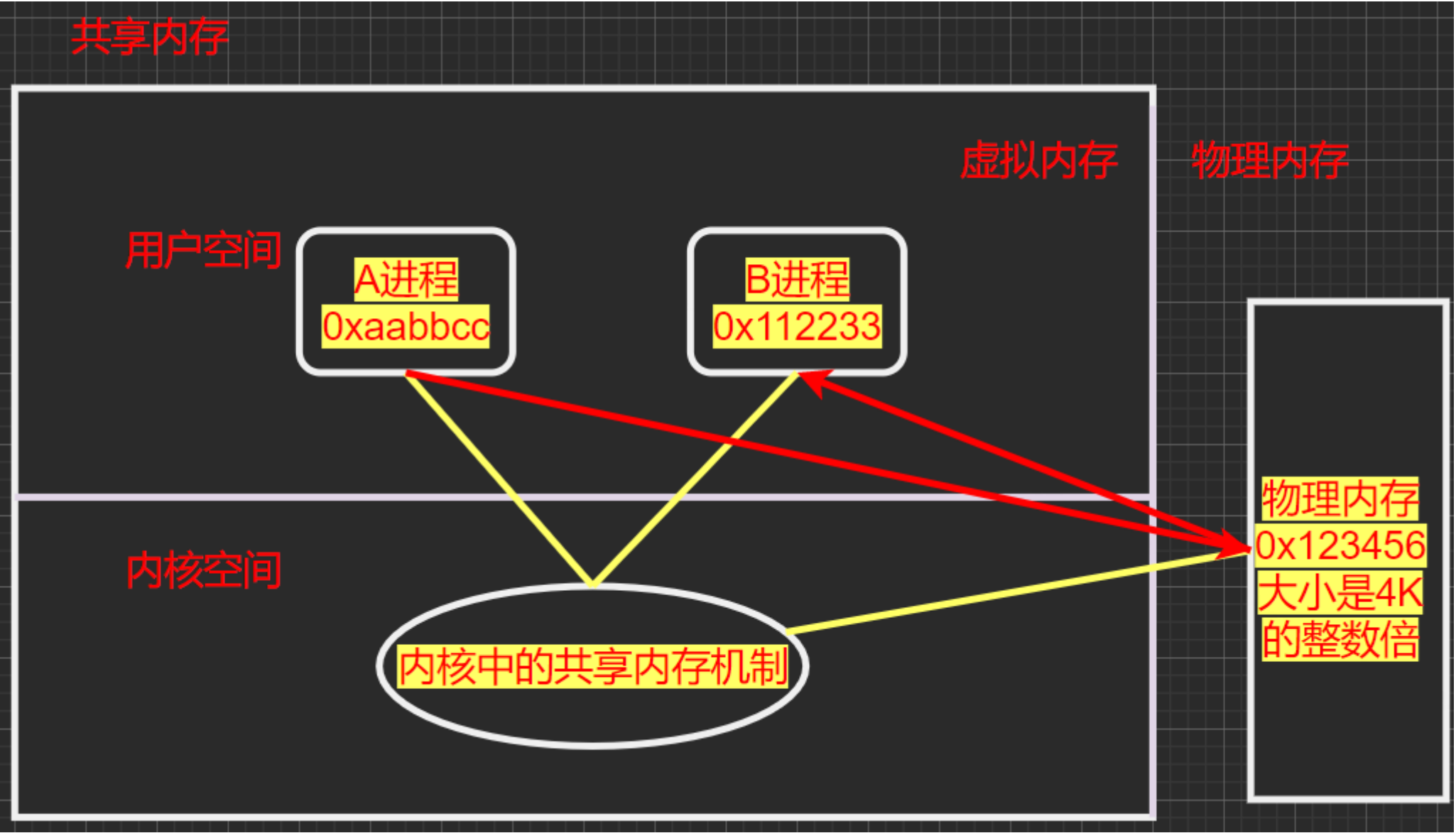
```
34 struct ipc_perm
35 {
36     __key_t __key;          //键值
37     __uid_t uid;            //消息队列所属的uid
38     __gid_t gid;            //消息队列所属的gid
39     __uid_t cuid;           //创建消息队列的uid
40     __gid_t cgid;           //创建消息队列的gid
41     unsigned short int mode; //消息队列的读写权限
42 };
```

```
1  #include <head.h>
2  typedef struct {
3      long mtype;
4      char name[19];
5      char sex;
6      int age;
7  } msg_t;
8
9  #define MSGSIZE (sizeof(msg_t) - sizeof(long))
10
11 int main(int argc, const char* argv[])
12 {
13     key_t key;
14     int msgqid;
15     // 1.获取键值
16     if ((key = ftok("/home/linux/", 'p')) == -1)
17         PRINT_ERR("ftok error");
18
19     // 2.创建消息队列
20     if ((msgqid = msgget(key, IPC_CREAT | 0666)) == -1)
21         PRINT_ERR("msgget error");
22
23     // 3.向消息队列中发消息
24     int ret;
25     msg_t msg;
26     while (1) {
27         retry:
28         printf("input (type name sex age) > ");
29         ret = scanf("%ld %s %c %d", &msg.mtype, msg.name, &msg.sex, &msg.age);
30         if (ret != 4) {
31             printf("input error,try again\n");
32             while (getchar() != '\n')
33                 ;
34             goto retry;
35         }
36
37         if (msgsnd(msgqid, &msg, MSGSIZE, 0))
38             PRINT_ERR("msgsnd error");
39
40         if (msg.mtype == 1000)
41             break;
42     }
43
44     // 4.获取消息队列属性
45     struct msqid_ds msqds;
46     if (msgctl(msgqid, IPC_STAT, &msqds))
47         PRINT_ERR("msgctl error");
48
49     printf("key=%#x,uid=%d,gid=%d,mode=%#o,nq=%ld,nb=%ld,mb=%ld\n",
50         msqds.msg_perm.__key, msqds.msg_perm.uid, msqds.msg_perm.gid,
51         msqds.msg_perm.mode, msqds.msg_qnum, msqds.__msg_cbytes, msqds.msg_qbytes);
52
53     // 5.设置消息队列属性
54     msqds.msg_qbytes = 32768;
55     if (msgctl(msgqid, IPC_SET, &msqds))
56         PRINT_ERR("msgctl error");
57
58     // 6.再次获取消息队列属性
59     memset(&msqds, 0, sizeof(msqds));
60     if (msgctl(msgqid, IPC_STAT, &msqds))
61         PRINT_ERR("msgctl error");
62
63     printf("key=%#x,uid=%d,gid=%d,mode=%#o,nq=%ld,nb=%ld,mb=%ld\n",
64         msqds.msg_perm.__key, msqds.msg_perm.uid, msqds.msg_perm.gid,
65         msqds.msg_perm.mode, msqds.msg_qnum, msqds.__msg_cbytes, msqds.msg_qbytes);
66
67     // 7.删除消息队列
68     if (msgctl(msgqid, IPC_RMID, NULL))
69         PRINT_ERR("msgctl error");
70     return 0;
71 }
```

1.3共享内存

1.3.1共享内存的工作原理

共享内存是所有进程间通信方式中效率最高的一个，因为当创建共享内存之后，需要通信的A和B进程可以直接操作这块物理内存空间，省去了向内核拷贝数据的过程。共享内存的大小是4K的整数倍。



1.3.2共享内存的API (shmget|shmat|shmdt|shmctl)

```
1 #include <sys/shm.h>
2 int shmget(key_t key, size_t size, int shmflg);
3 功能：创建共享内存
4 参数：
5     @key:键值
6     key:通过ftok获取
7     IPC_PRIVATE: 只能用于亲缘间进程的通信
8     @size:共享内存的大小 4k整数倍
9     @msgflag:共享的标志位
10    IPC_CREAT|0666 或 IPC_CREAT|IPC_EXCL|0666
11 返回值：成功返回共享内存编号，失败返回-1置位错误码
12
13 void *shmat(int shmid, const void *shmaddr, int shmflg);
14 功能：映射共享内存到当前的进程空间
15 参数：
16     @shmid:共享内存的编号
17     @shmaddr:NULL，让系统自动分配
18     @shmflg:共享内存的操作方式
19     0: 读写
20     SHM_RDONLY: 只读
21 返回值：成功返回共享内存的首地址，失败返回（void *）-1,并置位错误码
22
23 int shmdt(const void *shmaddr);
24 功能：取消地址映射
25 参数：
26     @shmaddr:指向共享内存的指针
27 返回值：成功返回0，失败返回-1置位错误码
28
29 int shmctl(int shmid, int cmd, struct shmid_ds *buf);
30 功能：共享内存控制的函数
31 参数：
32     @shmid:共享内存的编号
33     @cmd:操作的命令码
34     IPC_STAT : 获取
35     IPC_SET: 设置
36     IPC_RMID: 删除共享内存
37     标记要销毁的段。实际上，只有在最后一个进程将其分离之后
38     （也就是说，关联结构shmid_ds的shm_nattch成员为零时），
39     段才会被销毁。调用者必须是段的所有者或创建者，或具有特权。buf参数被忽略。
40     @buf:共享内存属性结构体指针
41 返回值:成功返回0，失败返回-1置位错误码
```


1.3.3共享内存的实例

01write.c

```
1  #include <head.h>
2  #define SHMSIZE (4096)
3  int main(int argc, const char* argv[])
4  {
5      key_t key;
6      int shmid;
7      char* waddr;
8      // 1.获取键值 ftok
9      if ((key = ftok("/home", 'g')) == -1)
10         PRINT_ERR("ftok error");
11     // 2.创建共享内存 shmget
12     if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) == -1)
13         PRINT_ERR("shmget error");
14     // 3.将共享内存映射到用户空间 shmat
15     if ((waddr = shmat(shmid, NULL, 0)) == (void*)-1)
16         PRINT_ERR("shmat error");
17     // 4.共享内存操作（写）
18     while (1) {
19         printf("input > ");
20         fgets(waddr, SHMSIZE, stdin);
21         if (waddr[strlen(waddr) - 1] == '\n')
22             waddr[strlen(waddr) - 1] = '\0';
23         if (strcmp(waddr, "quit") == 0)
24             break;
25     }
26     // 5.取消映射 shmdt
27     if(shmdt(waddr))
28         PRINT_ERR("shmdt error");
29     return 0;
30 }
```

02read.c

```
1  #include <head.h>
2  #define SHMSIZE (4096)
3  int main(int argc, const char* argv[])
4  {
5      key_t key;
6      int shmid;
7      char* raddr;
8      // 1.获取键值 ftok
9      if ((key = ftok("/home", 'g')) == -1)
10         PRINT_ERR("ftok error");
11     // 2.创建共享内存 shmget
12     if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) == -1)
13         PRINT_ERR("shmget error");
14     // 3.将共享内存映射到用户空间 shmat
15     if ((raddr = shmat(shmid, NULL, 0)) == (void*)-1)
16         PRINT_ERR("shmat error");
17     // 4.从共享内存中数数据
18     while (1) {
19         getchar(); // 敲回车读一次数据，如果不写这句话会疯狂刷屏
20         printf("read:%s\n", raddr);
21         if (strncmp(raddr, "quit", 4) == 0)
22             break;
23     }
24     // 5.取消映射 shmdt
25     if (shmdt(raddr))
26         PRINT_ERR("shmdt error");
27     // 6.删除共享内存 shmctl
28     if (shmctl(shmid, IPC_RMID, NULL))
29         PRINT_ERR("shmctl error");
30     return 0;
31 }
```

•linux@ubuntu:~/work/day8/03shm\$./write

input > hello

input > sdfjlassdf;kalsdlfasd

input > 123

input > 456

input > quit

◦linux@ubuntu:~/work/day8/03shm\$

•linux@ubuntu:~/work/day8/03shm\$./read

read:hello

read:sdfjlassdf;kalsdlfasd

read:456

read:quit

◦linux@ubuntu:~/work/day8/03shm\$

向共享内存写

敲回车才第一次

从共享内存读

这里123写入后没敲回车，写了456后敲回车，456把123覆盖了，只读取到了456

1.3.4shmctl函数详解

```
1  int shmctl(int shmid, int cmd, struct shmid_ds *buf);
2  功能：共享内存控制的函数
3  参数：
4      @shmid:共享内存的编号
5      @cmd:操作的命令码
6          IPC_STAT : 获取    (ipcs -m)
7          IPC_SET: 设置
8          IPC_RMID: 删除共享内存 (ipcrm -m shmid)
9              标记要销毁的段。实际上，只有在最后一个进程将其分离之后
10             (也就是说，关联结构shmid_ds的shm_nattch成员为零时)，
11             段才会被销毁。调用者必须是段的所有者或创建者，或具有特权。buf参数被忽略。
12      @buf:共享内存属性结构体指针
13  返回值:成功返回0，失败返回-1置位错误码
14
15  eg1:
16      shmctl(shmid,IPC_RMID,NULL); //删除共享内存
17  eg2:
18      struct shmid_ds buf;
19      shmctl(shmid,IPC_STAT,&buf); //获取消息队列的属性
20      //以下是shmid_ds结果的详解
21      struct shmid_ds
22      {
23          struct ipc_perm shm_perm; //权限结构体
24          size_t shm_segsz;          //共享内存的大小，单位是字节
25          __time_t shm_atime;         //最后一次调用shmat的时间
26          __time_t shm_dtime;         //最后一次调用shmdt的时间
27          __time_t shm_ctime;         //最后一次调用shmctl改变属性的时间
28          __pid_t shm_cpid;           //创建共享内存的PID
29          __pid_t shm_lpid;           //最后一次操作共享内存的PID
30          shmatt_t shm_nattch;        //当前多少个进程关联共享内存
31      };
32      struct ipc_perm
33      {
34          __key_t __key;              //键值
35          __uid_t uid;                //消息队列所属的uid
36          __gid_t gid;                //消息队列所属的gid
37          __uid_t cuid;               //创建消息队列的uid
38          __gid_t cgid;               //创建消息队列的gid
39          unsigned short int mode;    //消息队列的读写权限
40      };
```

1.4信号灯集

1.4.1信号灯集工作原理

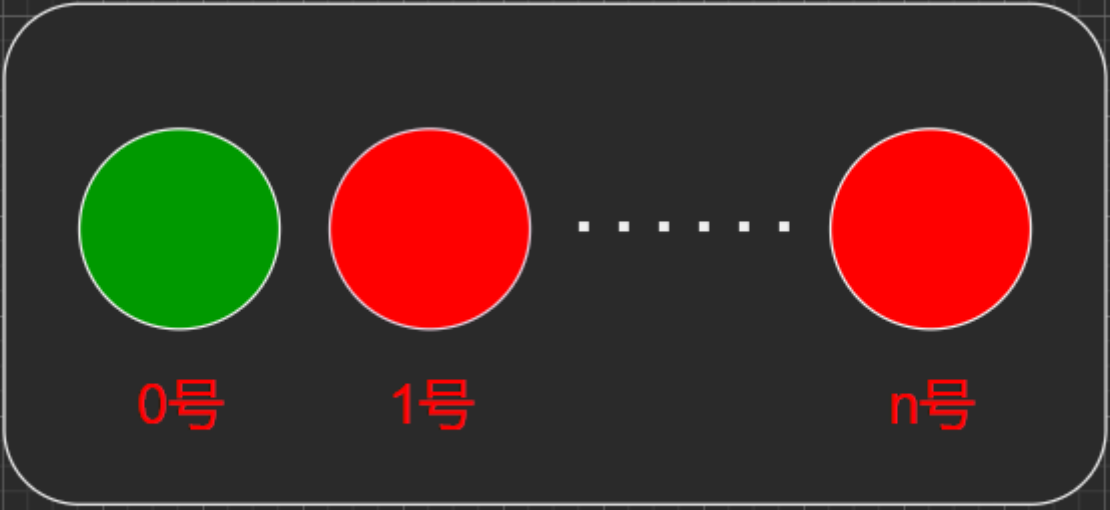
信号量（信号灯集）：是实现进程同步的机制，在一个信号

灯集中可以有很多个信号灯。在信号灯集内信号灯相互独立，

每个灯的值的改变不会影响其他的信号灯，信号灯的值一般

设置为二值量（1或者0，1代表有资源，0代表没有资源）。

信号灯集



绿色代表有资源，红色代表没有资源

1.4.2信号灯集的API (semget|semctl|semop)

```
1 #include <sys/sem.h>
2
3 int semget(key_t key, int nsems, int semflg);
4 功能：创建一个信号灯集
5 参数：
6     @key:键值
7         IPC_PRIVATE
8     key
9     @nsems:信号灯集中信号灯的个数
10    @semflg:创建的标志位
11        IPC_CREAT|0666 或 IPC_CREAT|IPC_EXCL|0666
12 返回值：成功返回semid,失败返回-1置位错误码
13
14
15 int semctl(int semid, int semnum, int cmd, ...);
16 功能：信号灯集的控制函数
17 参数：
18     @semid信号灯集的ID
19     @senum:信号灯的编号
20     @cmd:命令码
21         SETVAL: 设置信号灯的值 --->第四个参数val选项
22         GETVAL: 获取信号灯的值 --->不需要第四个参数
23         IPC_STAT: 获取信号灯集的属性--->第二个参数被忽略，第四个参数buf选项
24         IPC_SET : 设置信号灯集的属性--->第四个参数buf选项
25         IPC_RMID:第二参数被忽略，第4个参数不用填写
26     @...:
27         union semun {
28             int val; /* value for SETVAL */
29             struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
30         };
31 返回值：失败返回-1置位错误码
32     成功：
33         GETVAL:成功返回信号灯的值
34         其余的命令码成功返回0
35 eg:设置灯的值
36     union semun sem = {
37         .val = 1,
38     }
39     semctl(semid,0,SETVAL,sem); //将0号灯初始值设置为1
40     union semun sem = {
41         .val = 0,
42     }
43     semctl(semid,1,SETVAL,sem); //将1号灯初始值设置为0
44 eg:获取信号灯的值
45     val = semctl(semid,0,GETVAL); //获取0号灯的值
46
47 eg:获取信号灯集的属性
48     struct semid_ds buf;
49     union semun sem = {
50         .buf = &buf,
51     }
52     semctl(semid,0, IPC_STAT,sem); //获取信号灯集的属性，第二参数被忽略
53 eg:删除信号等集
54     semctl(semid,0,IPC_RMID); //删除信号等集
55
56 int semop(int semid, struct sembuf *sops, size_t nsops);
57 功能：信号灯集中信号灯的操作函数
58 参数：
59     @semid:信号灯集的编号
```

```
60      @sops:操作方式
61      struct sembuf{
62          unsigned short sem_num;    //信号灯的编号
63          short          sem_op;      //操作方式（PV）
64                                     -1:P操作，申请资源
65                                     1:V操作，释放资源
66          short          sem_flg;    //操作的标志位
67                                     0: 阻塞
68                                     IPC_NOWAIT: 非阻塞方式操作
69      }
70      @nsops:本次操作信号灯的个数
71      返回值: 成功返回0，失败返回-1置位错误码
```

1.4.3信号灯集的实例

01write.c

```
1  #include <head.h>
2  #define SHMSIZE (4096)
3  union semun {
4      int val; /* Value for SETVAL */
5      struct semid_ds* buf; /* Buffer for IPC_STAT, IPC_SET */
6  };
7  int mysem_init(int semid, int which, int value)
8  {
9      union semun sem = {
10         .val = value,
11     };
12     if (semctl(semid, which, SETVAL, sem) == -1)
13         PRINT_ERR("semctl error");
14
15     return 0;
16 }
17 int P(int semid, int which)
18 {
19     struct sembuf op = {
20         .sem_num = which, // 那个灯
21         .sem_op = -1, // -1 P申请   1 V释放
22         .sem_flg = 0, // 阻塞
23     };
24     if (semop(semid, &op, 1))
25         PRINT_ERR("semop P error");
26     return 0;
27 }
28 int V(int semid, int which)
29 {
30     struct sembuf op = {
31         .sem_num = which, // 那个灯
32         .sem_op = 1, // -1 P申请   1 V释放
33         .sem_flg = 0, // 阻塞
34     };
35     if (semop(semid, &op, 1))
36         PRINT_ERR("semop V error");
37     return 0;
38 }
39 int main(int argc, const char* argv[])
40 {
41     key_t key;
42     int shmid, semid;
43     char* waddr;
44     // 1.获取键值 ftok
45     if ((key = ftok("/home", 'g')) == -1)
46         PRINT_ERR("ftok error");
47     // 2.创建共享内存 shmget
48     if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) == -1)
49         PRINT_ERR("shmget error");
50     // 3.将共享内存映射到用户空间 shmat
51     if ((waddr = shmat(shmid, NULL, 0)) == (void*)-1)
52         PRINT_ERR("shmat error");
53
54     // 4.创建信号灯集，并初始化信号灯
55     if ((semid = semget(key, 2, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
56         if (errno = EEXIST) {
57             semid = semget(key, 2, IPC_CREAT|0666);
58         } else {
59             PRINT_ERR("semget error");
60         }
61     } else {
62         mysem_init(semid, 0, 1); // 将0号灯设置为1
63         mysem_init(semid, 1, 0); // 将1号灯设置为0
64     }
65 }
```

```
66 // 5.共享内存操作（写）
67 while (1) {
68     P(semid, 0); // 申请0号灯资源
69     printf("input > ");
70     fgets(waddr, SHMSIZE, stdin);
71     if (waddr[strlen(waddr) - 1] == '\n')
72         waddr[strlen(waddr) - 1] = '\0';
73     V(semid, 1); // 释放1号灯资源
74     if (strcmp(waddr, "quit") == 0)
75         break;
76 }
77 // 6.取消映射 shmdt
78 if (shmdt(waddr))
79     PRINT_ERR("shmdt error");
80 return 0;
81 }
```

02read.c

```
1  #include <head.h>
2  #define SHMSIZE (4096)
3  union semun {
4      int val; /* value for SETVAL */
5      struct semid_ds* buf; /* Buffer for IPC_STAT, IPC_SET */
6  };
7  int mysem_init(int semid, int which, int value)
8  {
9      union semun sem = {
10         .val = value,
11     };
12     if (semctl(semid, which, SETVAL, sem) == -1)
13         PRINT_ERR("semctl error");
14
15     return 0;
16 }
17 int P(int semid, int which)
18 {
19     struct sembuf op = {
20         .sem_num = which, // 那个灯
21         .sem_op = -1, // -1 P申请   1 V释放
22         .sem_flg = 0, // 阻塞
23     };
24     if (semop(semid, &op, 1))
25         PRINT_ERR("semop P error");
26     return 0;
27 }
28 int V(int semid, int which)
29 {
30     struct sembuf op = {
31         .sem_num = which, // 那个灯
32         .sem_op = 1, // -1 P申请   1 V释放
33         .sem_flg = 0, // 阻塞
34     };
35     if (semop(semid, &op, 1))
36         PRINT_ERR("semop V error");
37     return 0;
38 }
39 int main(int argc, const char* argv[])
40 {
41     key_t key;
42     int shmid, semid;
43     char* raddr;
44     // 1.获取键值 ftok
45     if ((key = ftok("/home", 'g')) == -1)
46         PRINT_ERR("ftok error");
47     // 2.创建共享内存 shmget
48     if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) == -1)
49         PRINT_ERR("shmget error");
50     // 3.将共享内存映射到用户空间 shmat
51     if ((raddr = shmat(shmid, NULL, 0)) == (void*)-1)
52         PRINT_ERR("shmat error");
53
54     // 4.创建信号灯集，并初始化信号灯
55     if ((semid = semget(key, 2, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
56         if (errno == EEXIST) {
57             semid = semget(key, 2, IPC_CREAT | 0666);
58         } else {
59             PRINT_ERR("semget error");
60         }
61     } else {
62         mysem_init(semid, 0, 1); // 将0号灯设置为1
63         mysem_init(semid, 1, 0); // 将1号灯设置为0
64     }
65 }
```

```
66 // 5.从共享内存中数数据
67 while (1) {
68     P(semid, 1); // 申请1号灯资源
69     printf("read:%s\n", raddr);
70     V(semid, 0); // 释放0号灯资源
71     if (strncmp(raddr, "quit", 4) == 0)
72         break;
73 }
74 // 6.取消映射 shmdt
75 if (shmdt(raddr))
76     PRINT_ERR("shmdt error");
77 // 7.删除共享内存 shmctl
78 if (shmctl(semid, IPC_RMID, NULL))
79     PRINT_ERR("shmctl error");
80 // 8.删除信号灯集
81 if (semctl(semid, 0, IPC_RMID))
82     PRINT_ERR("semctl error");
83 return 0;
84 }
```

2.总结

一、数据结构

- 1.顺序表
- 2.单链表
- 3.单向循环链表
- 4.双向链表
- 5.双向循环链表
- 6.栈（顺序栈，链式栈）
- 7.队列（循环队列，链式队列）
- 8.二叉树（创建，遍历）
- 9.快排
- 10.哈希

二、IO进程

fopen/fclose/fgetc/fputc/fgets/fputs/fread/fwrite/fseek/ftell/rewind

perror/strerror/snprintf/sprintf/printf/time/localtime

open/read/write/close/lseek/stat/lstat/getpwuid/getgrgid

opendir/readdir/closedir

fork/getpid/getppid/exit/_exit/wait/waitpid/dup/dup2/system

pthread_create/pthread_self/pthread_exit/pthread_join/pthread_detach/pthread_cancel

pthread_mutex_init/pthread_mutex_lock/pthread_mutex_unlock/pthread_mutex_destroy

sem_init/sem_wait/sem_post/sem_destroy

pthread_cond_init/pthread_cond_wait/pthread_cond_signal/

pthread_cond_broatcast/pthread_cond_destroy

pipe/mkfifo/signal/kill/raise/alarm

ftok/msgget/msgsnd/msgrcv/msgctl

shmget/shmat/shmdt/shmctl

semget/semctl/semop