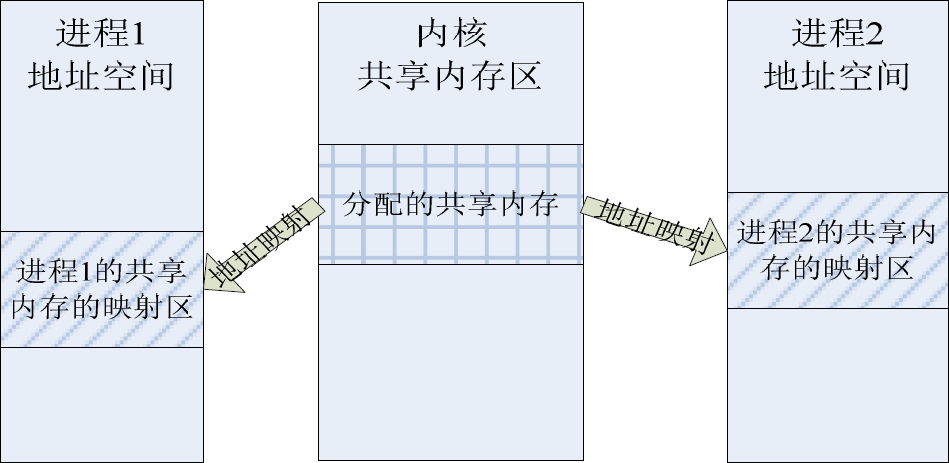
## 第24课 讲稿（进程间通信-4）

1. **共享内存**
2. 共享内存概述

共享内存是一种**最为高效**的进程间通信方式，因为进程可以直接读写内存，不需要任何数据的复制。为了在多个进程间交换信息，内核专门留出了一块内存区，这段内存区可以由需要访问的进程将其映射到自己的私有地址空间。因此，进程就可以直接读写这一内存区而不需要进行数据的复制，从而大大提高了效率。当然，由于多个进程共享一段内存，因此也需要依靠某种同步机制，如互斥锁和信号量等。其原理示意图如下图所示。



1. 共享内存的应用
2. 函数说明

共享内存的实现分为几个步骤，第一步是创建共享内存或获取已存在的共享内存的标识符，这里用到的函数是shmget()，也就是从内存中获得一段共享内存区域；第二步映射共享内存，也就是把这段创建的共享内存映射到具体的进程空间中，这里使用的函数是shmat()；到这里，可以使用**不带缓冲的I/O读写命令**对其这段内存进行操作；除此之外，还有撤销映射的操作，其函数为shmdt()；最后，还需要删除共享内存，函数为shmctl()。这里就主要介绍这4个函数。

1. 函数格式

表20列举了shmget()函数的语法要点。

|  |  |
| --- | --- |
| 表20 shmget()函数语法要点 | |
| 所需头文件 | #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h> |
| 函数原型 | int shmget(key\_t key, int size, int shmflg) |
| 函数传入值 | key：共享内存的键值，可由ftok()创建，多个进程可以通过它访问同一个共享内存，如果键值不存在，第三个参数shmflg应该使用IPC\_CREAT来创建新的共享内存；有个特殊键值0（IPC\_PRIVATE），用于创建当前进程的私有共享内存，该键值不需要使用IPC\_CREAT参数。 |
| size：共享内存区大小 |
| shmflg：同open()函数的权限位，也可以用八进制表示法 |
| 返回值 | 成功：共享内存段标识符 |
| 出错：-1（错误原因在errno中） |

表21列举了shmat()函数的语法要点。

|  |  |  |
| --- | --- | --- |
| 表21 shmat()函数语法要点 | | |
| 所需头文件 | #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h> | |
| 函数原型 | char \*shmat(int shmid, const void \*shmaddr, int shmflg) | |
| 函数传入值 | shmid：要映射的共享内存区标识符 | |
| shmaddr：将共享内存映射到指定地址（若为0则表示系统自动分配地址并把该段共享内存映射到调用进程的地址空间） | |
| shmflg | SHM\_RDONLY：共享内存只读 |
| 默认0：共享内存可读写 |
| 返回值 | 成功：被映射的段地址 | |
| 出错：-1（错误原因在errno中） | |

表22列举了shmdt()函数的语法要点。

|  |  |
| --- | --- |
| 表22 shmdt()函数语法要点 | |
| 所需头文件 | #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h> |
| 函数原型 | int shmdt(const void \*shmaddr) |
| 函数传入值 | shmaddr：被映射的共享内存段地址 |
| 返回值 | 成功：0 |
| 出错：-1（错误原因在errno中） |

表23列举了shmctl()函数的语法要点。

|  |  |  |
| --- | --- | --- |
| 表23 shmctl()函数语法要点 | | |
| 所需头文件 | #include <sys/types.h>  #include <sys/shm.h> | |
| 函数原型 | int shmctl(int shmid, int cmd, struct shmid\_ds \*buf) | |
| 函数传入值 | shmid：共享内存区标识符 | |
| cmd | IPC\_STAT：得到共享内存的状态，把共享内存的shmid\_ds结构复制到buf中 |
| IPC\_SET：改变共享内存的状态，把buf所指的shmid\_ds结构中的uid、gid、mode等复制到共享内存的shmid\_ds结构内 |
| IPC\_RMID：删除这片共享内存 |
| buf：共享内存管理结构体，作用和semctl()中的结构体类似 | |
| 返回值 | 成功：0 | |
| 出错：-1（错误原因在errno中） | |

* shmid\_ds结构体定义如下：

struct shmid\_ds{

struct ipc\_perm shm\_perm; /\*操作权限\*/

int shm\_segsz; /\*段的大小（以字节为单位）\*/

time\_t shm\_atime; /\*最后一个进程附加到该段的时间\*/

time\_t shm\_dtime; /\*最后一个进程离开该段的时间\*/

time\_t shm\_ctime; /\*最后一个进程修改该段的时间\*/

unsigned short shm\_cpid; /\*创建该段进程的pid\*/

unsigned short shm\_lpid; /\*在该段上操作的最后1个进程的pid\*/

short shm\_nattch; /\*当前附加到该段的进程的个数\*/

};

* ipc\_perm结构体定义如下：

struct ipc\_perm

{

key\_t key; /\*调用shmget()时给出的关键字 \*/

uid\_t uid; /\*共享内存所有者的有效用户ID \*/

gid\_t gid; /\*共享内存所有者所属组的有效组ID\*/

uid\_t cuid; /\*共享内存创建 者的有效用户ID\*/

gid\_t cgid; /\*共享内存创建者所属组的有效组ID\*/

unsigned short mode; /\*Permissions + SHM\_DEST和SHM\_LOCKED标志\*/

unsignedshort seq; /\*序列号\*/

};

1. 使用实例

该实例说明如何使用基本的共享内存函数。

* 首先是创建一个共享内存区（采用的共享内存的键值IPC\_PRIVATE，之后创建子进程，在父子两个进程中将共享内存分别映射到各自的进程地址空间之中。
* 父进程先等待用户输入，然后将用户输入的字符串写入到共享内存，之后往共享内存的头部写入“WROTE”字符串表示父进程已成功写入数据。
* 子进程一直等到共享内存的头部字符串为“WROTE”，然后将共享内存的有效数据（在父进程中用户输入的字符串）在屏幕上打印。
* 父子两个进程在完成以上工作之后，分别解除与共享内存的映射关系。
* 最后在子进程中删除共享内存（也可以在父进程中删除）。
* 因为共享内存自身并不提供同步机制，所以应该额外实现不同进程之间的同步（如信号量）。为了简单起见，在本实例中用标志字符串来实现非常简单的父子进程之间的同步。

ipcs是用于报告进程间通信机制状态的命令，它可以查看共享内存（-m）、消息队列（-q）、信号量（-s）等各种进程间通信机制的情况，这里使用了system()函数用于调用shell命令“ipcs”。程序源代码如下所示：

|  |
| --- |
| /\* shmem.c \*/  #include<sys/types.h>  #include<sys/ipc.h>  #include<sys/shm.h>  #include<stdio.h>  #include<stdlib.h>  #include<string.h>  #define BUFFER\_SIZE 2048  int main()  {  pid\_t pid;  int shmid; /\*共享内存标识符 \*/  char \*shm\_addr; /\* 共享内存映射地址 \*/  char flag[] = "WROTE";  char buff[BUFFER\_SIZE];  /\* 创建共享内存 \*/  if ((shmid = shmget(IPC\_PRIVATE, BUFFER\_SIZE, 0666)) < 0)  {  perror("shmget");  exit(1);  }  else  {  printf("Create shared-memory: %d\n",shmid);  }  /\* 显示共享内存情况 \*/  system("ipcs -m");  pid = fork();  if (pid == -1)  {  perror("fork");  exit(1);  }  else if (pid == 0) /\* 子进程处理 \*/  {  /\*映射共享内存\*/  if ((shm\_addr = shmat(shmid, 0, 0)) == (void\*)-1)  {  perror("Child: shmat");  exit(1);  }  else  {  printf("Child: Attach shared-memory: %p\n", shm\_addr);  }  system("ipcs -m");  /\* 通过检查在共享内存的头部是否标志字符串"WROTE"来确认父进程已经向共享内存写入有效数据 \*/  while (strncmp(shm\_addr, flag, strlen(flag)))  {  printf("Child: Wait for enable data...\n");  sleep(5);  }  /\* 获取共享内存的有效数据并显示 \*/  strcpy(buff, shm\_addr + strlen(flag));  printf("Child: Shared-memory :%s\n", buff);  /\* 解除共享内存映射 \*/  if ((shmdt(shm\_addr)) < 0)  {  perror("shmdt");  exit(1);  }  else  {  printf("Child: Deattach shared-memory\n");  }  system("ipcs -m");  /\* 删除共享内存 \*/  if (shmctl(shmid, IPC\_RMID, NULL) == -1)  {  perror("Child: shmctl(IPC\_RMID)\n");  exit(1);  }  else  {  printf("Delete shared-memory\n");  }  system("ipcs -m");  }  else/\* 父进程处理 \*/  {  /\*映射共享内存\*/  if ((shm\_addr = shmat(shmid, 0, 0)) == (void\*)-1)  {  perror("Parent: shmat");  exit(1);  }  else  {  printf("Parent: Attach shared-memory: %p\n", shm\_addr);  }  sleep(1);  printf("\nInput some string:\n");  fgets(buff, BUFFER\_SIZE, stdin);  strncpy(shm\_addr + strlen(flag), buff, strlen(buff));  strncpy(shm\_addr, flag, strlen(flag));  /\* 解除共享内存映射 \*/  if ((shmdt(shm\_addr)) < 0)  {  perror("Parent: shmdt");  exit(1);  }  else  {  printf("Parent: Deattach shared-memory\n");  }  system("ipcs -m");  waitpid(pid, NULL, 0);  printf("Finished\n");  }  exit(0);  } |

下面是运行结果。从该结果可以看出，nattch的值随着共享内存状态的变化而变化，共享内存的值根据不同的系统会有所不同。

|  |
| --- |
| [root@localhost between\_process]# ./a.out  Create shared-memory: 1081365  /\* 在刚创建共享内存时（尚未有任何地址映射）共享内存的情况 \*/  ------ Shared Memory Segments --------  key shmid owner perms bytes nattch status  0x00000000 98304 root 600 393216 2 dest  ……  0x00000000 983060 root 666 2048 0  0x00000000 1081365 root 666 2048 0  Parent: Attach shared-memory: 0xb7723000/\* 共享内存的映射地址 \*/  Child: Attach shared-memory: 0xb7723000  /\* 在父子进程中进行共享内存的地址映射后共享内存的情况 \*/  ------ Shared Memory Segments --------  key shmid owner perms bytes nattch status  0x00000000 98304 root 600 393216 2 dest  ……  0x00000000 983060 root 666 2048 0  0x00000000 1081365 root 666 2048 2  Child: Wait for enable data...  Input some string:  hello/\* 用户输入字符串“hello” \*/  Parent: Deattach shared-memory  /\* 在父进程中解除共享内存的映射关系后共享内存的情况 \*/  ------ Shared Memory Segments --------  key shmid owner perms bytes nattch status  0x00000000 98304 root 600 393216 2 dest  ……  0x00000000 983060 root 666 2048 0  0x00000000 1081365 root 666 2048 1  /\* 在子进程中读取共享内存的有效数据并打印 \*/  Child: Shared-memory :hello  Child: Deattach shared-memory  /\* 在子进程中解除共享内存的映射关系后共享内存的情况 \*/  ------ Shared Memory Segments --------  key shmid owner perms bytes nattch status  0x00000000 98304 root 600 393216 2 dest  ……  0x00000000 983060 root 666 2048 0  0x00000000 1081365 root 666 2048 0  Delete shared-memory  /\* 在删除共享内存后共享内存的情况，shmid为1081365的内存空间已经释放\*/  ------ Shared Memory Segments --------  key shmid owner perms bytes nattch status  0x00000000 98304 root 600 393216 2 dest  ……  0x00000000 983060 root 666 2048 0  Finished |