**操作系统**

**实 验 报 告**

实验名称： 地址映射与共享

学 员： 侯华玮 学 号： 202102001015

培养类型： 无军籍学员 年 级： 21级

专业：计算机科学与技术（天河拔尖班） 所属学院： 计算机学院

指导教员： 文艳军 职 称： 教授

实 验 室： 实验日期： 2023.04.12

国防科技大学训练部制

《实验报告》填写说明

1．学员完成人才培养方案和课程标准要所要求的每个实验后，均须提交实验报告。

2．实验报告封面必须打印，报告内容可以手写或打印。

3．实验报告内容编排及打印应符合以下要求：

（1）采用A4（21cm×29.7cm）白色复印纸，单面黑字打印。上下左右各侧的页边距均为3cm；缺省文档网格：字号为小4号，中文为宋体，英文和阿拉伯数字为Times New Roman，每页30行，每行36字；页脚距边界为2.5cm，页码置于页脚、居中，采用小5号阿拉伯数字从1开始连续编排，封面不编页码。

（2）报告正文最多可设四级标题，字体均为黑体，第一级标题字号为4号，其余各级标题为小4号；标题序号第一级用“一、”、“二、”……，第二级用“（一）”、“（二）” ……，第三级用“1.”、“2.” ……，第四级用“（1）”、“（2）” ……，分别按序连续编排。

（3）正文插图、表格中的文字字号均为5号。

一、实验目的和内容

**实验目的：**

* 深入理解操作系统的段、页式内存管理，深入理解段表、页表、逻辑地址、线性地址、物理地址等概念；
* 实践段、页式内存管理的地址映射过程；
* 编程实现段、页式内存管理上的内存共享，从而深入理解操作系统的内存管理。

**实验内容：**

* 用 Bochs 调试工具跟踪 Linux 0.11 的地址翻译（地址映射）过程，了解 IA-32 和 Linux 0.11 的内存管理机制；
* 在 Ubuntu 上编写多进程的生产者—消费者程序，用共享内存做缓冲区；
* 为 Linux 0.11 增加共享内存功能，编写并测试生产者—消费者程序。

二、操作方法与实验步骤

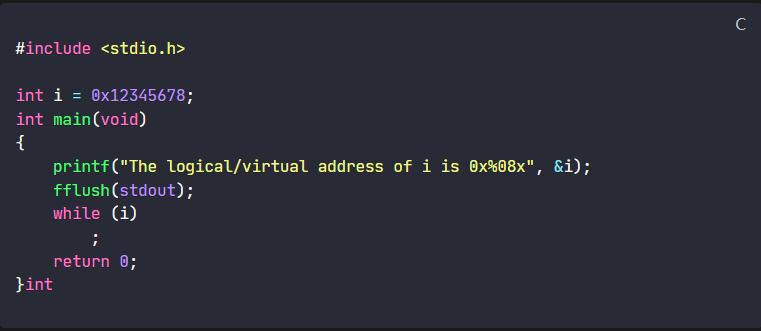
**（一）地址映：跟踪Liunx的地址映射过程**

**1.主要操作方法：**

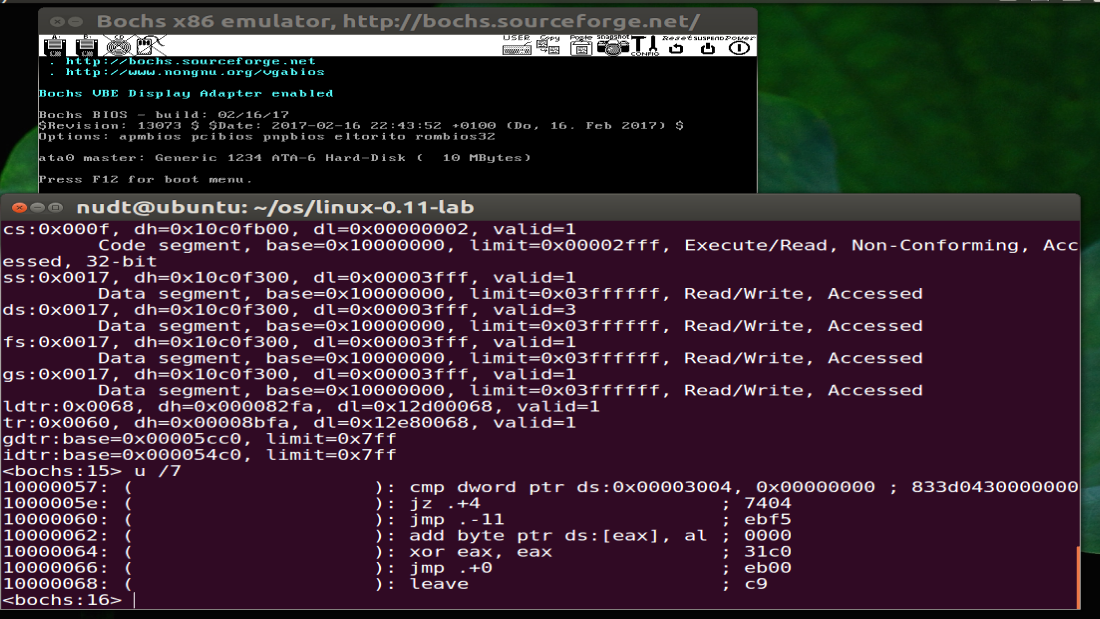
1. 以汇编级调试的方式启动Bochs。
2. 在Linux 0.11下编译运行测试程序test.c，该程序会进入死循环，不自动退出。
3. 跟踪运行中的测试程序中的变量i的地址映射过程：  
   在调试器中通过查看各项系统参数，从逻辑地址、LDT 表、GDT 表、线性地址到页表，计算出变量 i 的物理地址。
4. 查看物理地址处变量i的值，直接修改物理内存，使程序退出。

**2.实验步骤与过程：**

* 编译0号内核源码，用bochsdbg开始调试内核。
* 编写并运行测试程序test.c：



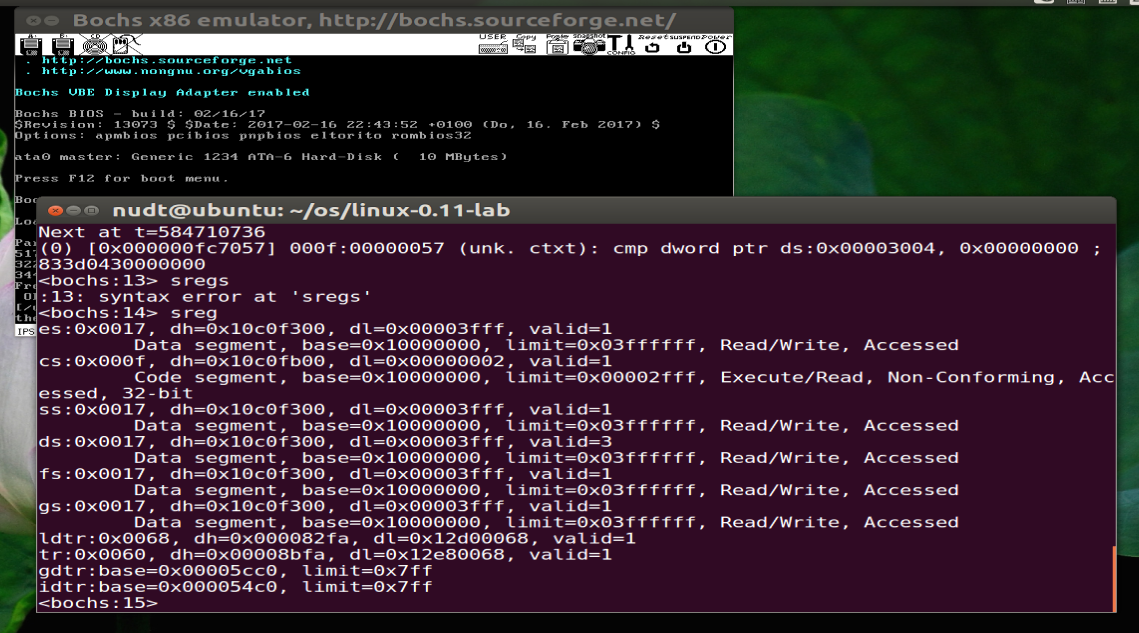
* 在调试器中中断程序执行，并反汇编查看程序指令：



发现当前指令将ds:0x3004处的值与 0比较，并由跳转构成的循环可知，变量i就保存在地址ds:0x3004中。下面通过段翻译和页翻译找到变量i的物理地址。

* **段翻译：逻辑地址→线性地址**

使用sreg命令查看段寄存器



得到对应数据段的选择符ds = 0x17，则逻辑地址为：0x17：0x3004

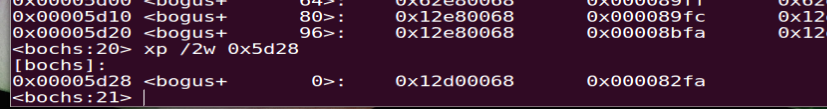
接下来要通过分析段选择符ds，找到对应的段描述符，获得变量i所在段的基地址。

ds = 0x17 = b(0001 0111)，其中TI = 1，INDEX = b(10) = 2，即所寻段的描述符位于局部描述符表LDT的2号索引处。

如何查局部描述符表呢？根据386寻址机制，局部描述符表也被视作“段”，其段描述符位于全局描述符表中，由内存管理寄存器LDTR为其选择符。全局描述符表GDT起始地址由全局描述符表寄存器GDTR给出。

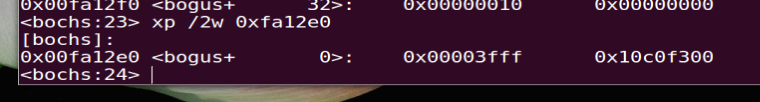
在本示例中，gdtr = 0x54c0，ldtr = 0x68 = b(0110 1000)，INDEX = b(01101) = 13 则当前进程的LDT的段描述符位于以0x5cc0为起始地址的GDT的13号索引处。由于一个描述符长度为8个字节，则应查看0x54c0 + 13 \* 8 = 处两个字长的内容。

使用命令xp /2w 0x5d28，结果如下：



根据段描述符的组成结构可得，当前LDT的基地址为：0x00fa12d0，则该LDT的第2号索引处0xfa12d0 + 2 \* 8 = 0xfa12e0就是ds对应段的描述符了。

使用命令xp /2w 0xfa12e0，结果如下：



有dl = 0x3fff, dh = 0x10c0f300，则可得ds段在虚拟内存中的基地址为0x10000000。

结合offset = 0x3004，可得变量i的线性（虚拟）地址为0x1000 3004。

* **第二步：线性地址→物理地址**

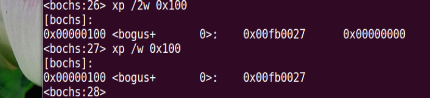
根据线性地址的组成结构，可获得：

* + 页目录号DIR = b(0001 0000 00) = 64
  + 页表号PAGE = b(00 0000 0011) = 3
  + 页内偏移量OFFSET = b(0000 0000 0100) = 4

由CR3 = 0x00获得页目录表的基地址0x00。

查看页目录表的第64号索引项：0x00 + 64\*4 = 0x100。

xp /w 0x100



得到页目录项为0x00fb 0027，其中P=1， R/W = 1则该页目录项有效。则二级页表所在物理页框号为0xfb0，即起始地址为0xfb 0000，从该处查看第三号索引项：0xfb 0000 + 3 \* 4 = 0xfb 000c。

xp /w 0xfb 000c

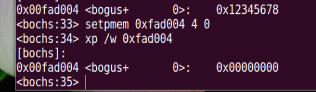


页表项内容为0xfad067，则该页表项有效，可得物理页的页框号为0xfad，起始地址为0xfa d000，与页内偏移量0x4相加，可得变量i所在的物理地址0xfa d004

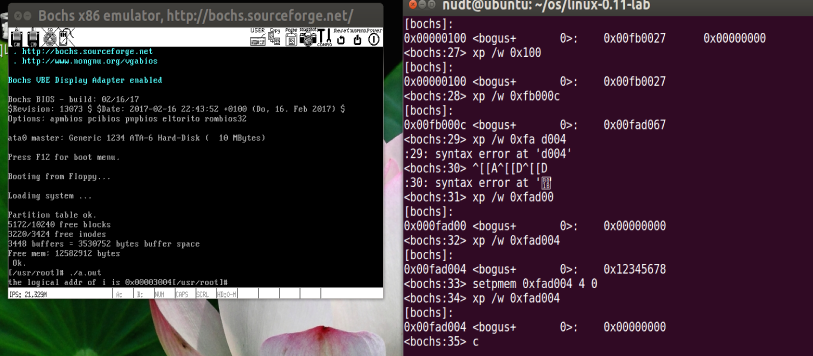
查看该地址处内容，发现确为i的值0x12345678。



直接修改i的值：



在调试器中输入命令c继续运行，发现程序退出，符合预期。



**（二）内存共享：添加内存共享系统调用并编写消费者-生产者测试程序**

**1.主要操作方法：**

1. 修改版本0的Linux 0.11内核源码，添加shmget与shmat两个与内存共享相关的系统调用。
2. 在bochs虚拟机中，编写并测试多进程的生产者消费者程序。

**2.实验步骤与过程：**

* **添加系统调用**

int shmget(key\_t key, size\_t size, int shmflg);

void \*shmat(int shmid, const void \*shmaddr, int shmflg);

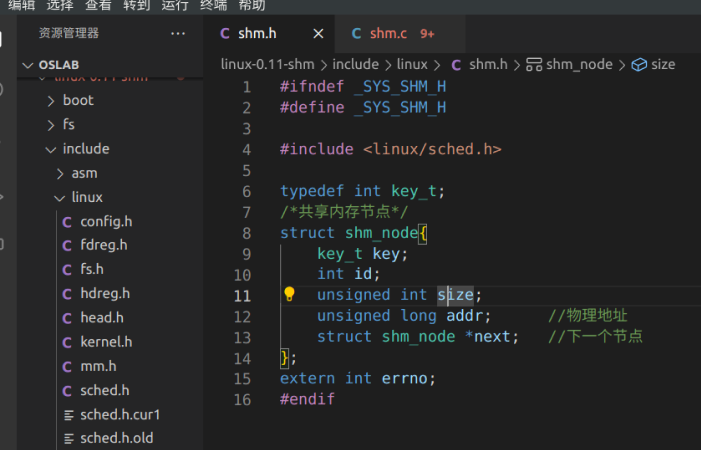
shmget系统调用

该系统调用会新建/打开一页物理内存作为共享内存，返回该页共享内存的shmid，即该页共享内存在操作系统中的标识id。如果多个进程使用相同的key调用shmget，则这些进程就会获得相同的shmid，即得到了同一块共享内存的id。在shmget实现时，如果key所对应的共享内存已经建立，则直接返回shmid，否则新建。如果size超过一页内存的大小，返回-1，并置errno为EINVAL。如果系统无空闲内存，返回-1，并置errno为ENOMEM。shmflg参数可以忽略。

shmat系统调用

该系统调用会将shmid指定的共享页面映射到当前进程的虚拟地址空间中，并返回一个逻辑地址p，调用进程可以读写逻辑地址p来读写这一页共享内存。两个进程都调用shmat可以关联到同一页内存上，从而形成进程共享内存页结构，此时两个进程读写p指针就是在读写同一页内存，从而实现了基于共享内存的进程间通信。如果shmid非法，返回-1，并置errno为EINVAL，shmaddr 和 shmflg 这两个参数都可忽略。

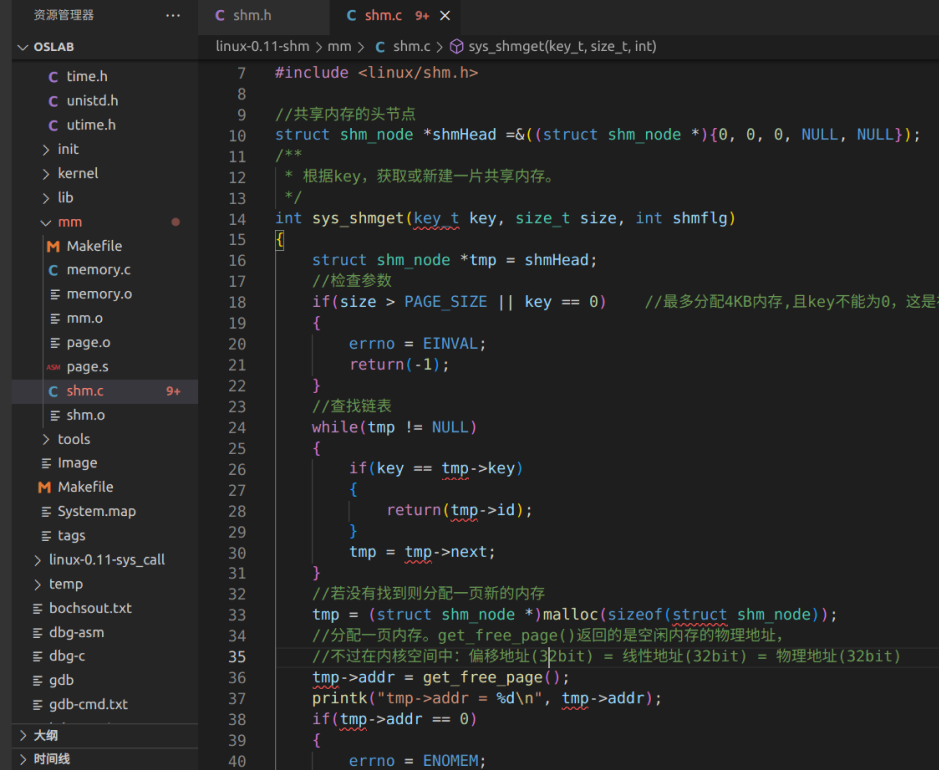
在/linux/include/linux/目录下，添加头文件shm.h，声明共享内存的结点结构体shm\_node，即每块分配的共享内存，，都会将其对应的共享内存结点添加到一个链表中，从而进行管理维护。



由于这两个系统调用都是对内存进行操作的函数，故在/linux/mm/目录下实现两个系统调用函数：

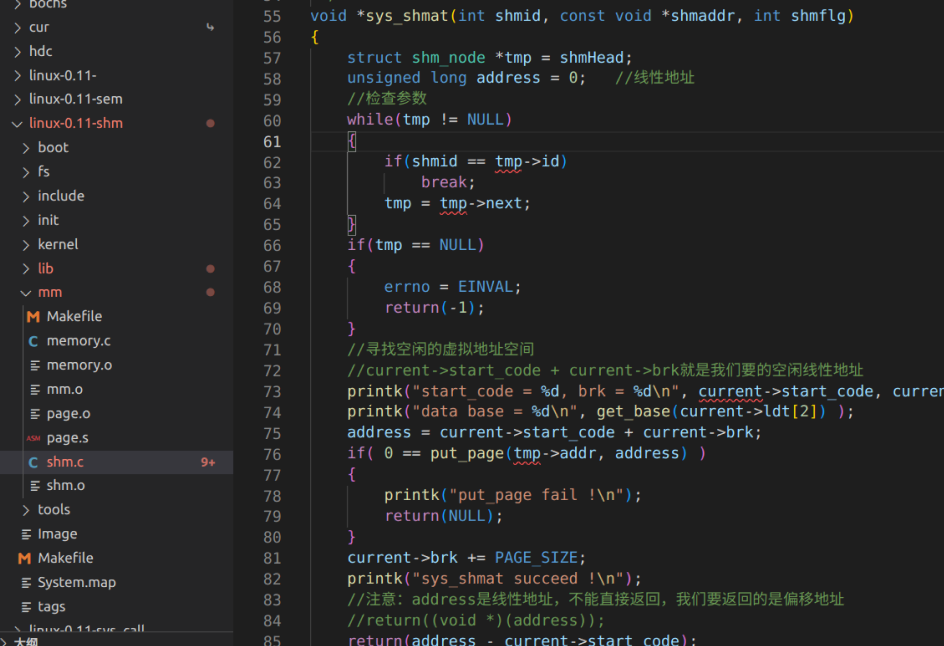
sys\_shmget的实现逻辑：

* 先对传入的参数进行检验。
* 根据传入的key，查找共享内存结点链表，若key存在，则直接返回其shmid。
* 若未找到key，说明尚未分配，则调用malloc分配一页新的虚拟内存。
* 将用get\_free\_gage获得的物理页帧与该块虚拟内存建立映射。



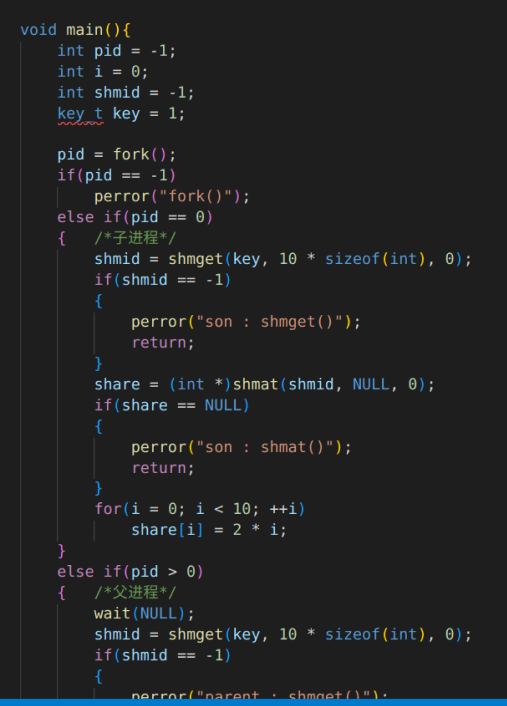
sys\_shmat的实现逻辑：

* 首先检查传入的参数。
* 遍历共享内存链表，查找与给定shmid相匹配的共享内存块。
* 如果找不到相应的共享内存块，则设置errno为EINVAL并返回-1。
* 在进程地址空间中寻找空闲的虚拟地址空间，以便将共享内存映射到该地址空间中。
* 如果找不到可用的地址空间，则返回NULL。
* 如果找到可用的地址空间，则将共享内存块的物理地址映射到该地址空间中。
* 更新进程的brk指针，以便下一次映射能够使用不同的地址空间。
* 返回偏移地址。



**共享内存测试程序**

首先，定义一个int类型的指针变量share，用于指向共享内存区域。然后，它定义了一个key\_t类型的变量key，并将其设置为1。这个key将用于创建共享内存区域。使用fork()函数创建一个子进程。如果fork()返回值为-1，则输出错误信息并退出程序。如果fork()返回值为0，则进入子进程代码块。

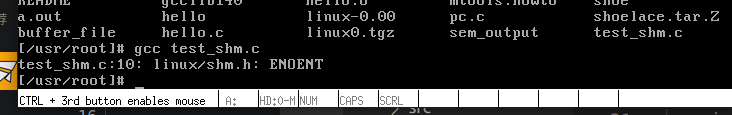


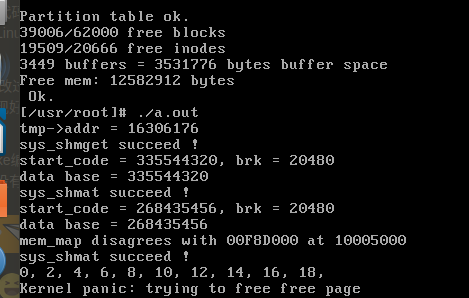
在子进程代码块中，代码使用shmget()函数创建一个共享内存区域，并将共享内存区域附加到子进程的地址空间中。若分配成功，则通过shmat()获得逻辑地址偏移量，在共享内存区域中写入一些数据。

在父进程代码块中，代码等待子进程结束。然后，它使用shmget()函数创建一个共享内存区域，并将共享内存区域附加到父进程的地址空间中。如果获取到正确的偏移量，则通过shmat()获得逻辑地址偏移量，从共享内存区域中读取数据，并在屏幕上输出这些数据。

其中，子进程是生产者进程，父进程是消费者进程，实现了消费者-生产者之间的内存共享。

测试结果如下：





三、实验结果与分析

**（一）地址映射：**

成功通过地址映射过程找到了变量的物理地址。

逻辑地址->线性地址->物理地址

在i386架构下的Linux 0.11中，地址映射是通过段式存储管理实现的。段式存储管理是将进程的虚拟地址空间划分为多个段，每个段都有自己的段基址和长度。当进程访问虚拟地址时，系统将根据虚拟地址所属的段来计算出对应的物理地址。具体实现时，可以使用描述符表（即GDT和LDT）来存储每个段的信息，并使用段选择子来指定当前进程使用哪个段描述符。

**（二）内存共享：**

通过实现shmget和shmat系统调用，编写生产者-消费者程序，实现了内存共享。

在i386架构下的Linux 0.11中内存共享是通过共享内存机制实现的。共享内存是一种进程间通信的方式，它允许多个进程共享同一个物理内存区域。实现共享内存需要使用系统调用shmget和shmat。shmget用于创建或获取一个共享内存区域的标识符，而shmat用于将共享内存区域映射到进程的地址空间中。具体实现时，需要使用内存管理功能来实现共享内存区域的分配和管理，但本实验没有使用同步机制（如信号量或互斥锁）来保证数据的一致性和完整性，以避免生产者和消费者之间的竞争和冲突，这是需要改善的地方。

四、问题与建议

**问题：**

1. 编译错误或警告：需要仔细阅读编译器的错误或警告信息，并根据信息对代码进行修改。
2. 文件系统损坏：在进行文件系统实验时，可能会出现文件系统损坏的情况。建议在操作之前备份重要文件，以防止文件丢失。
3. 硬件或虚拟机配置问题：实验需要在特定的硬件或虚拟机配置下运行，如果配置不正确，则可能导致实验无法正常进行。

**建议：**

1. 仔细阅读实验指导书，理解实验要求和操作步骤。
2. 详细记录实验过程和结果，便于后期分析和调试。
3. 遇到问题时，先仔细查看和分析错误信息，然后针对性地进行调试和修改。
4. 在实验完成后，可以通过查看系统日志等方式对实验结果进行验证，以确保实验的正确性。
5. 对实验过程中遇到的问题汇总归纳，获得一般性的的解决方法和经验。