实验 6 地址映射与共享

## 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

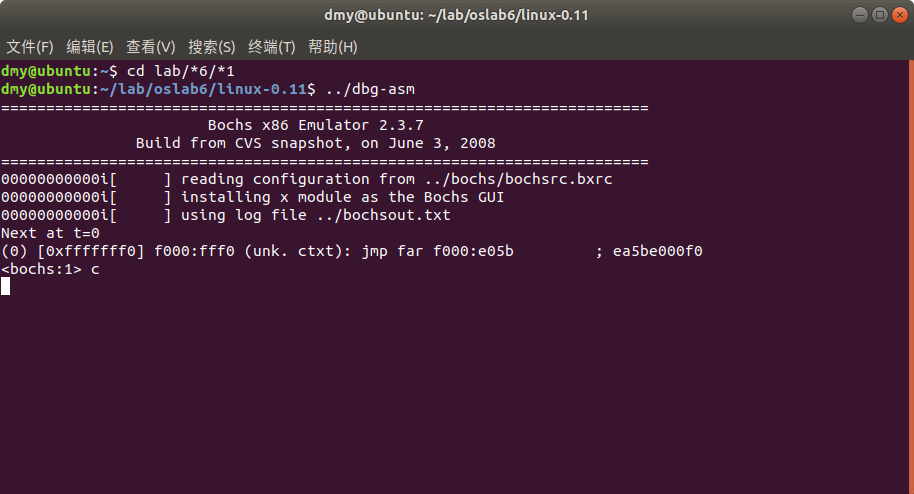
* 用 Bochs 调试工具跟踪 Linux 0.11 的地址翻译（地址映射）过程，了解 IA-32 和 Linux 0.11 的内存管理机制；
* 在 Ubuntu 上编写多进程的生产者—消费者程序，用共享内存做缓冲区；
* 在信号量实验的基础上，为 Linux 0.11 增加共享内存功能，并将生产者—消费者程序移植到 Linux 0.11。

## 实验报告

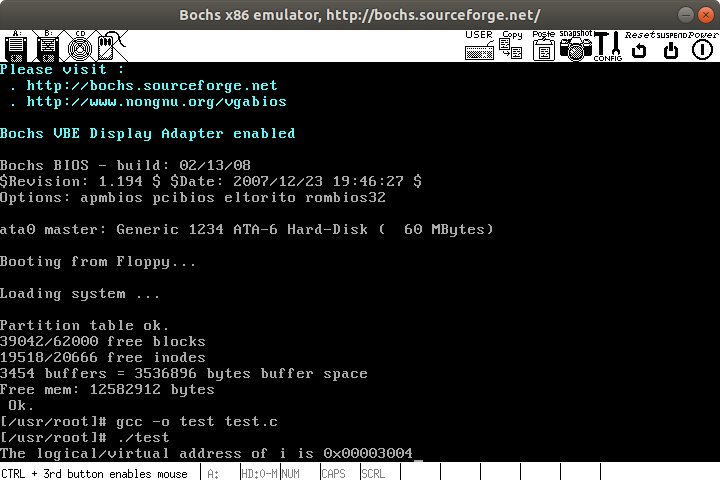
|  |
| --- |
| * 对于地址映射实验部分，列出你认为最重要的那几步（不超过 4 步），并给出你获得的实验数据。 * test.c 退出后，如果马上再运行一次，并再进行地址跟踪，你发现有哪些异同？为什么？ |
| 最重要：获线段基址0x10000000  最关键：获得物理地址0x00fa6004 |
| 物理地址变了，线性地址没变（至少3次内没变化），因为可能是因为分配的页面被分配到别的物理页框。 |

## 实验步骤

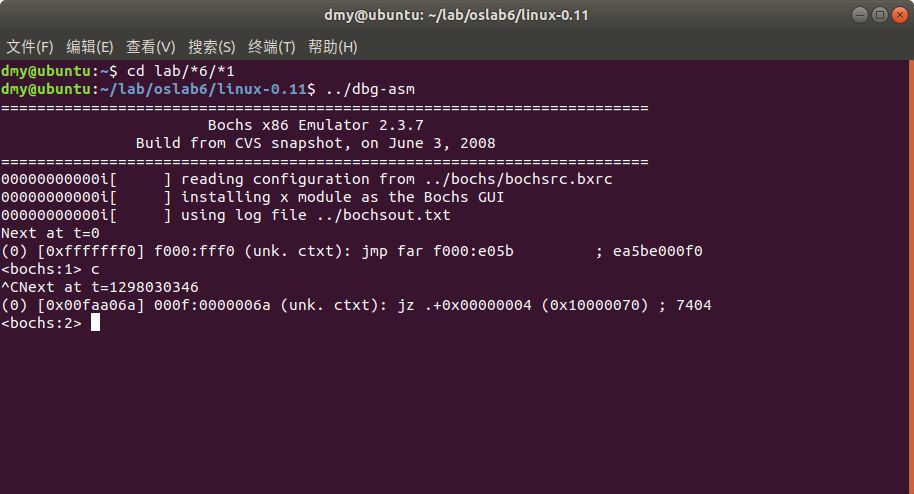
1. 用 Bochs 调试工具跟踪 Linux 0.11 的地址翻译（地址映射）过程，了解 IA-32 和 Linux 0.11 的内存管理机制；
   1. 启动调试器



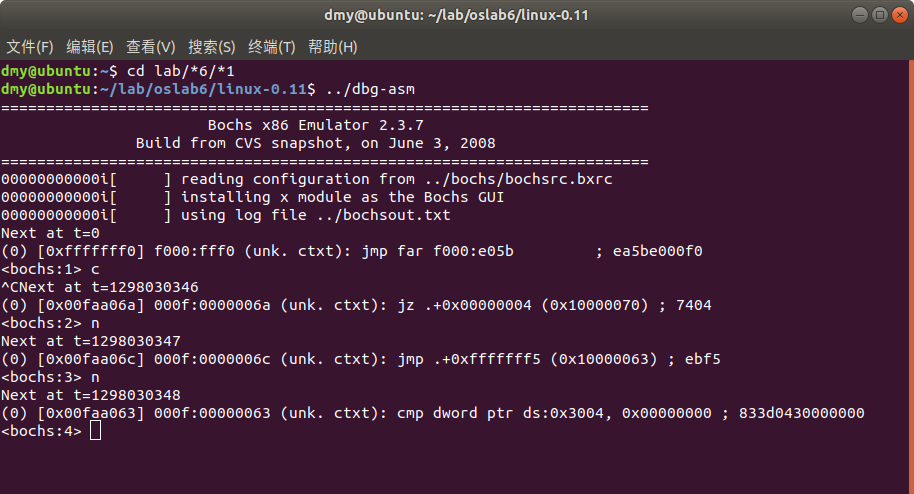
* 1. 编译运行test.c



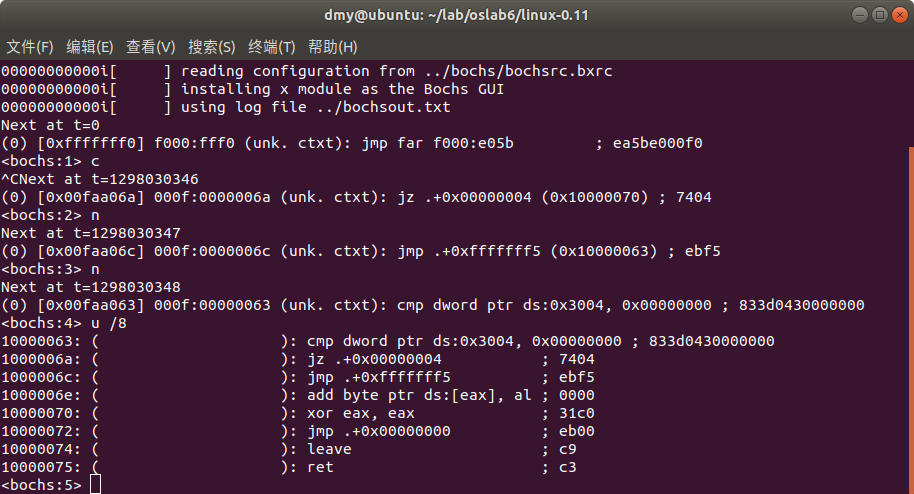
* 1. 暂停运行，进入调试状态



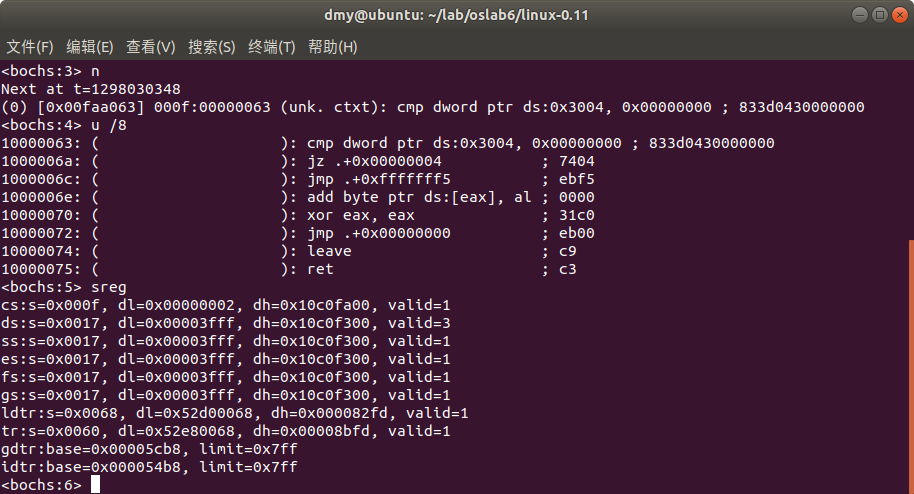
* 1. 执行到cmp语句



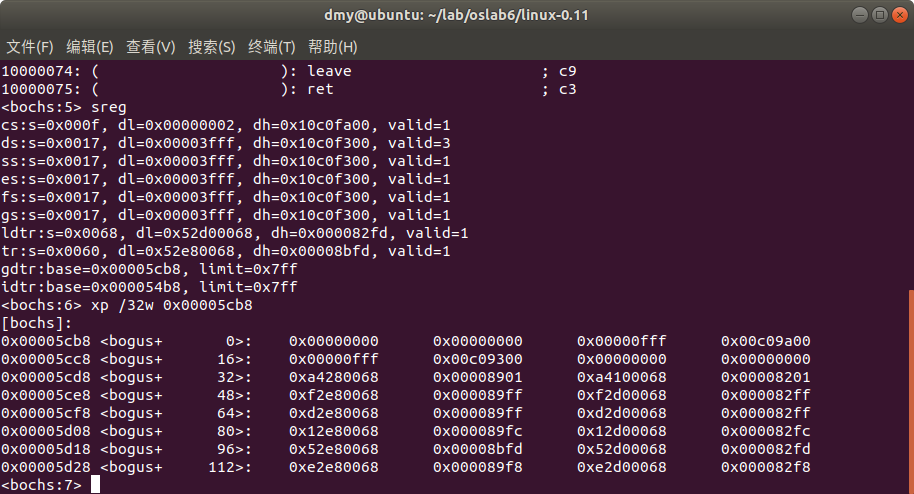
* 1. 查看当前位置开始 8 条指令的反汇编代码



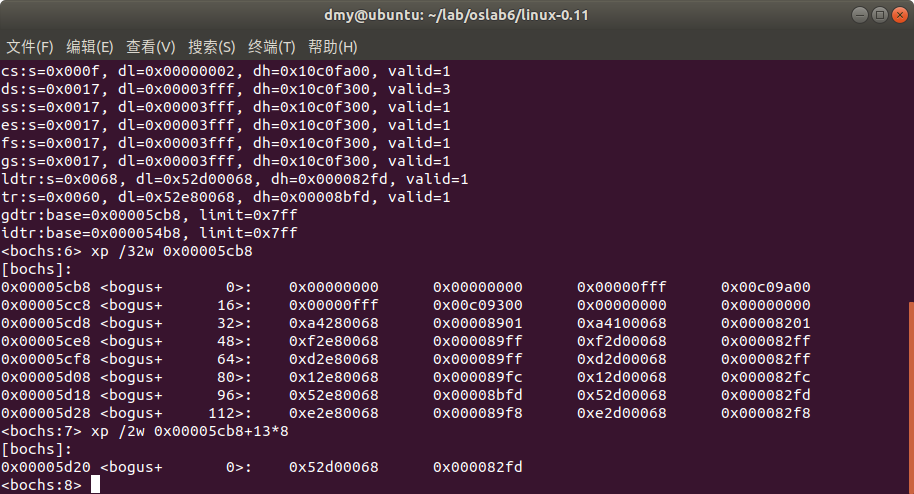
* 1. 查找ldtr和gdtr ，找段表位置

 ldtr 的值是 0x0068=0000000001101000（二进制），表示 LDT 表存放在 GDT 表的 1101（二进制）=13（十进制）号位置，GDT 的位置已经由 gdtr 明确给出，在物理地址的 0x00005cb8

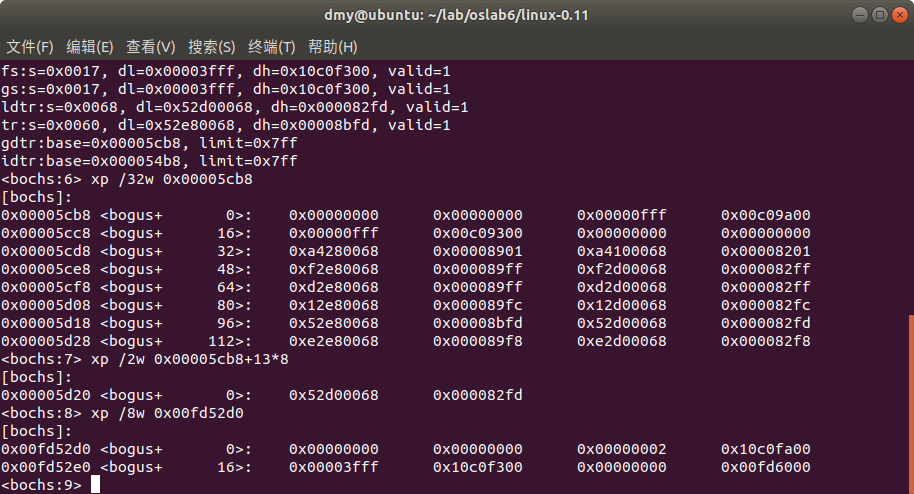
* 1. 查看GDT 表的前 16 项



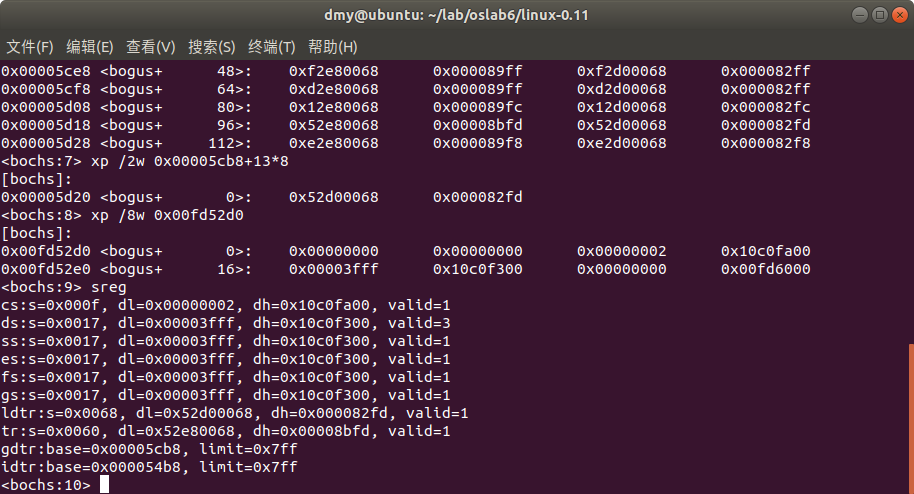
* 1. 查看GDT中LDT的段描述符（在0x00005cb8+13\*8）



* 1. 得到 LDT 表的物理地址0X00FD52D0，查询知LDT 表的前 4 项



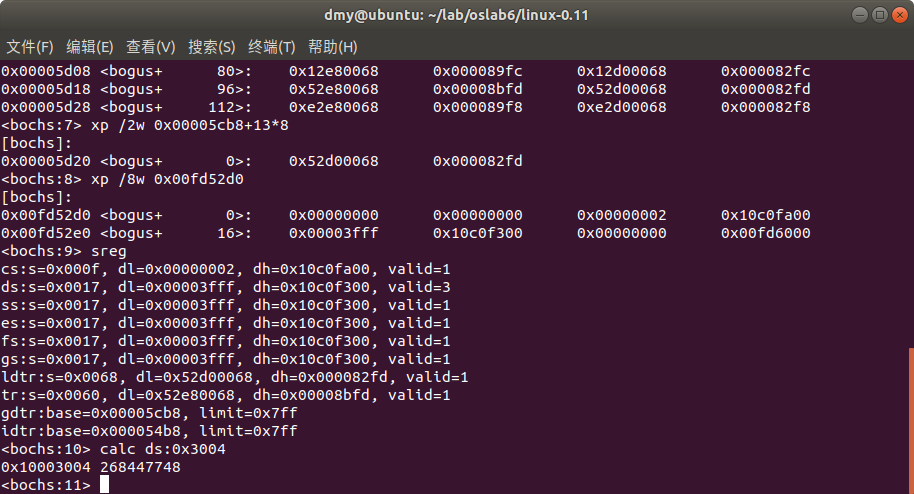
* 1. 查看ds选择子内容



0x0017=0000000000010111，段描述符索引为2

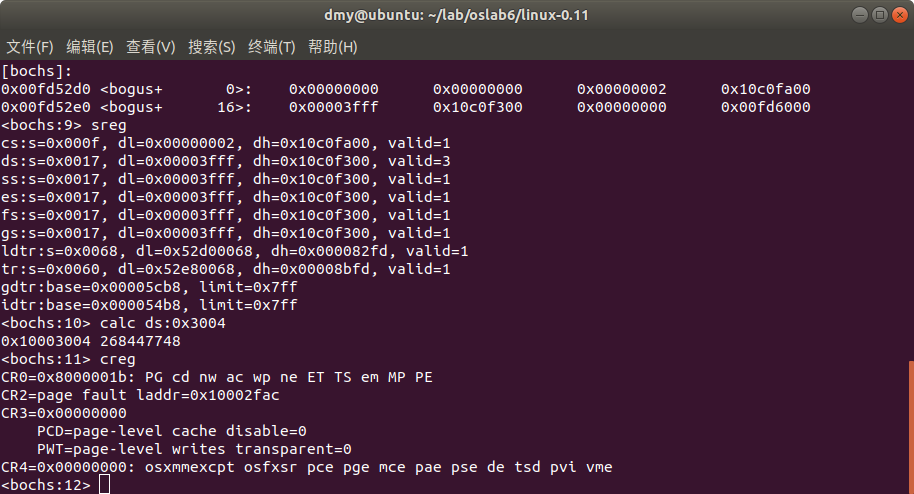
得到段描述符为0x00003fff 0x10c0f300与LDT中索引值为2的项一致，计算得到段基址为0x10000000，加上段内偏移0x3004，知道地址为0x10003004

* 1. 验证结果

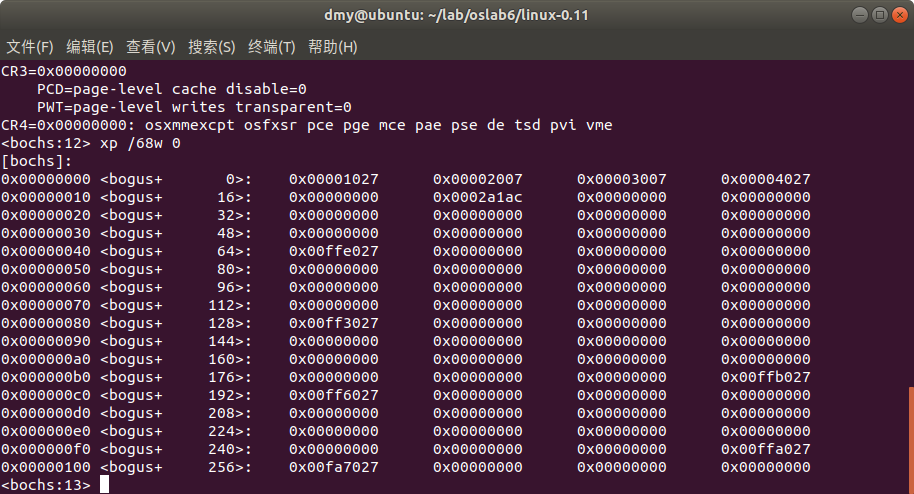


这是一个虚地址要用页表变成物理地址

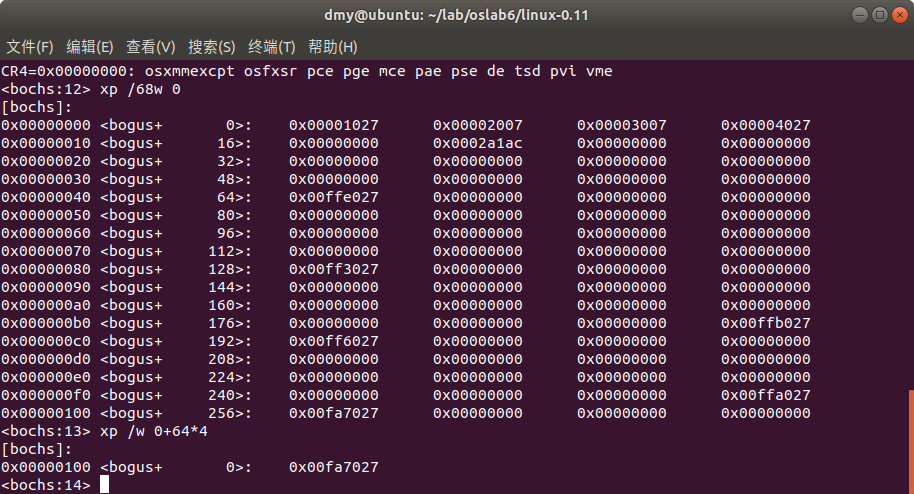
* 1. 0x10003004 的页目录号是 64，页号 3，页内偏移是 4，要找物理地址，首先要找页表目录位置



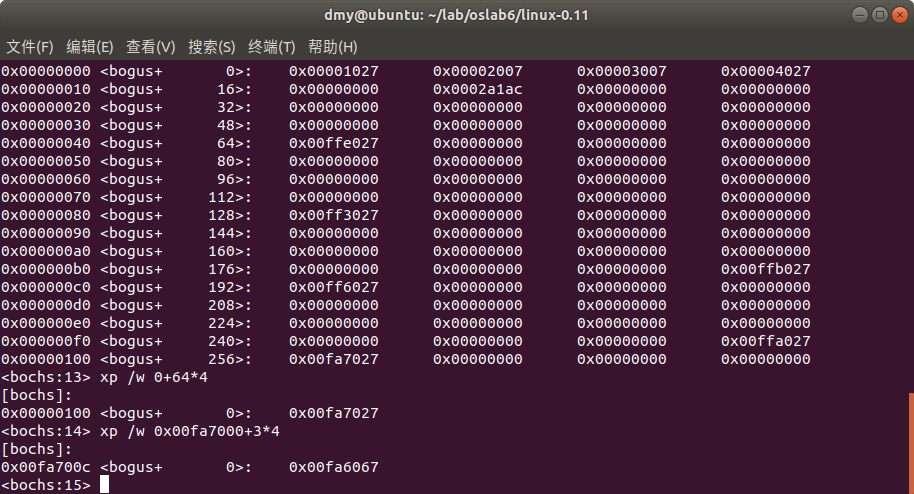
* 1. 页目录表的基址为 0，查看其内容



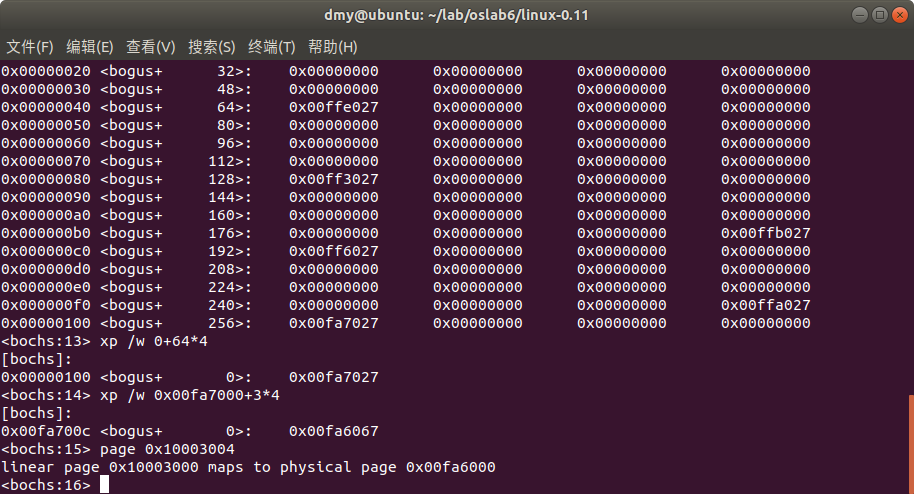
* 1. 查看页目录号为64的页目录项



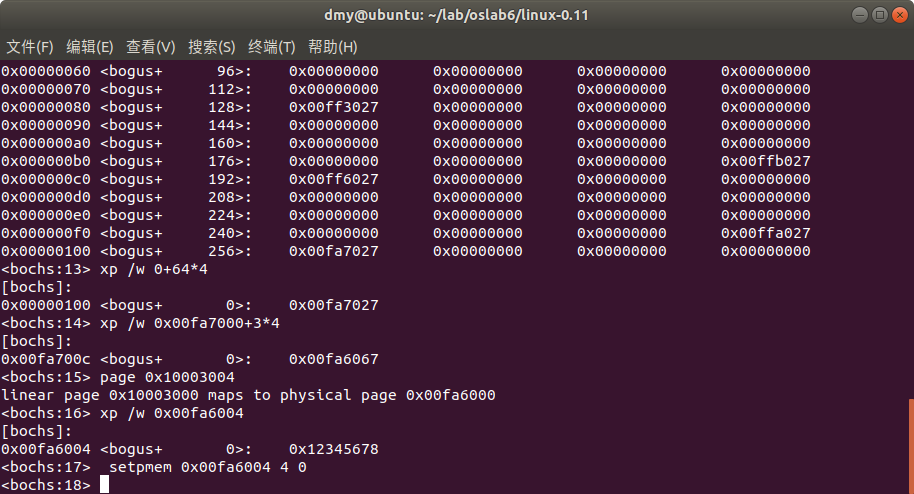
* 1. 页表所在物理页框号为 0x00fa7，查找 3 号页表项



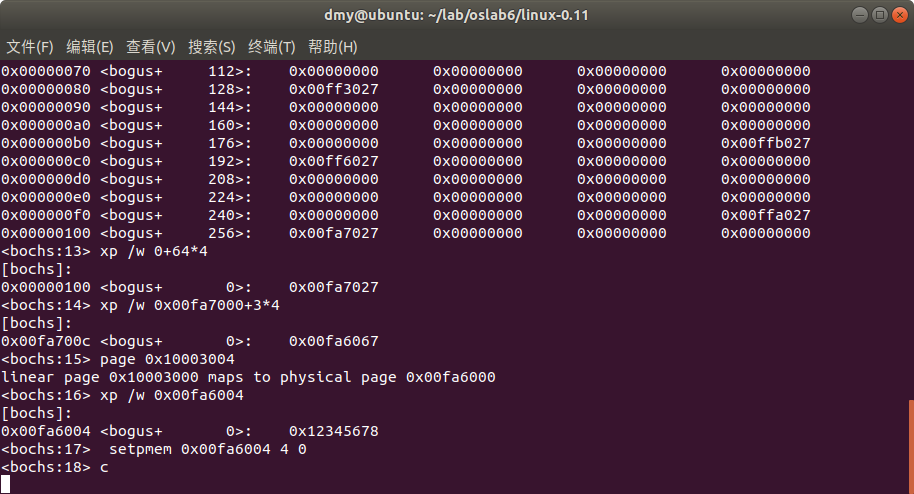
* 1. 得到物理地址为0x00fa6004，进行验证



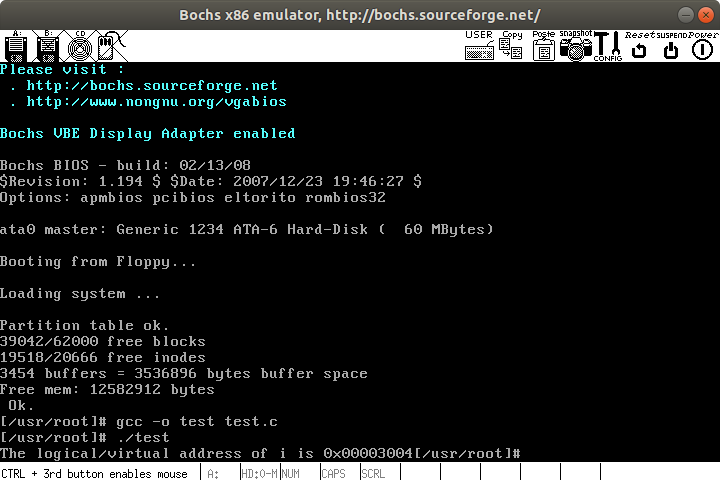
修改内存



继续运行



结束



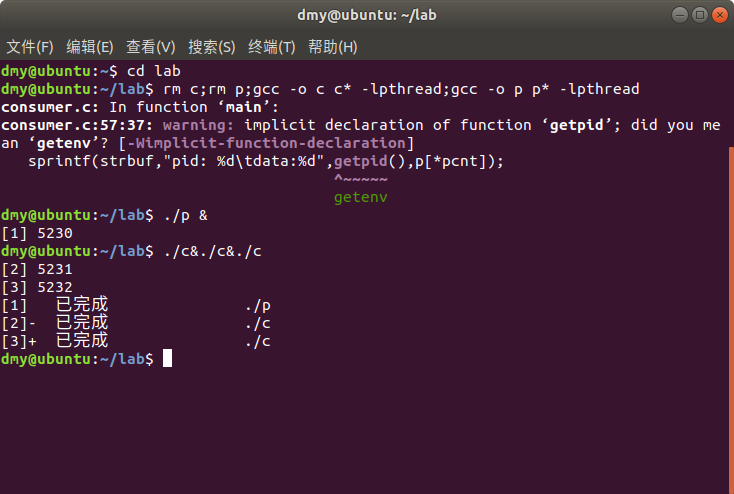
1. 在 Ubuntu 上编写多进程的生产者—消费者程序，用共享内存做缓冲区；
   1. Producer.c

|  |
| --- |
| #define \_\_LIBRARY\_\_  #include<stdio.h>  #include<unistd.h>  #include<stdlib.h>  #include<fcntl.h>  #define N 3  #define M 10  #define BUFSIZE 2  \_syscall2(sem\_t \*, sem\_open, const char\*,name,unsigned int,value);  \_syscall1(int,sem\_wait,sem\_t \*,sem);  \_syscall1(int,sem\_post,sem\_t \*,sem);  \_syscall1(int,sem\_unlink,const char \*,name);  \_syscall2(int,shmget,unsigned int,key,unsigned int,size);  \_syscall1(int,shmat,int,shmid);  sem\_t \*empty=NULL,\*mutex=NULL,\*full=NULL;  char strbuf[32];  int main()  {  int i=0;  int \*p=NULL;  int \*pcnt=NULL;  int data=1;  int shmid;  int shmcntid;  #ifndef \_DEBUG\_  freopen("out.txt","w",stdout);  #endif  empty=sem\_open("empty",BUFSIZE);  mutex=sem\_open("mutex",1);  full=sem\_open("full",0);  #ifdef \_DEBUG\_  if(!(empty&&mutex&&full))  perror("producer open sem fail\r\n");  else  perror("producer open sem sucess\r\n");  #endif  shmid=shmget(1,(BUFSIZE)\*sizeof(int));  shmcntid=shmget(2,sizeof(int));  #ifdef \_DEBUG\_  if(-1==shmid||-1==shmcntid)  perror("producer shmget fail\r\n");  else  perror("producer shmget sucess\r\n");  #endif  p=(int\*)shmat(shmid);  pcnt=(int\*)shmat(shmcntid);  \*pcnt=0;  while(data<=M)  {  sem\_wait(empty);  sem\_wait(mutex);  #ifdef \_DEBUG\_  sprintf(strbuf,"%d put data %d\r\n",getpid(),data);  write(1,strbuf,strlen(strbuf));  #endif  p[data-1]=data;  data++;  sem\_post(mutex);  sem\_post(full);  }  #ifdef \_DEBUG\_  sprintf(strbuf,"producer %d end\r\n",getpid());  write(1,strbuf,strlen(strbuf));  #endif  return 0;  } |

* 1. Consumer.c

|  |
| --- |
| #define \_\_LIBRARY\_\_  #include<stdio.h>  #include<unistd.h>  #include<stdlib.h>  #include<fcntl.h>  #define N 3  #define M 10  #define BUFSIZE 2  \_syscall2(sem\_t \*, sem\_open, const char\*,name,unsigned int,value);  \_syscall1(int,sem\_wait,sem\_t \*,sem);  \_syscall1(int,sem\_post,sem\_t \*,sem);  \_syscall1(int,sem\_unlink,const char \*,name);  \_syscall2(int,shmget,unsigned int,key,unsigned int,size);  \_syscall1(int,shmat,int,shmid);  sem\_t \*empty=NULL,\*mutex=NULL,\*full=NULL;  char strbuf[32];  int main()  {  int i=0;  int shmid=0;  int shmcntid;  int endflag;  int \*p=NULL;  int \*pcnt=NULL;  #ifndef \_DEBUG\_  freopen("./out.txt","a",stdout);  #endif  empty=sem\_open("empty",0);  mutex=sem\_open("mutex",0);  full=sem\_open("full",0);  #ifdef \_DEBUG\_  if(!(empty&&mutex&&full))  perror("consumer open sem fail\r\n");  else  perror("consumer open sem sucess\r\n");  #endif  shmid=shmget(1,(BUFSIZE)\*sizeof(int));  shmcntid=shmget(2,sizeof(int));  #ifdef \_DEBUG\_  if(-1==shmid||-1==shmcntid)  perror("consumer shmget fail\r\n");  else  perror("consumer shmget sucess\r\n");  #endif  p=(int\*)shmat(shmid);  pcnt=(int\*)shmat(shmcntid);  while(\*pcnt<M)  {  sem\_wait(full);  sem\_wait(mutex);  fflush(stdout);  #if defined \_DEBUG\_  sprintf(strbuf,"p:%d d:%d@%d\r\n",getpid(),p[\*pcnt],\*pcnt);  #else  sprintf(strbuf,"pid: %d\tdata:%d\r\n",getpid(),p[\*pcnt]);  #endif  write(1,strbuf,strlen(strbuf));  p[\*pcnt]=-1;  (\*pcnt)++;  fflush(stdout);  sem\_post(mutex);  sem\_post(empty);  }  #ifdef \_DEBUG\_  sprintf(strbuf,"consumer %d end\r\n",getpid());  write(1,strbuf,strlen(strbuf));  #endif  if(\*pcnt>=M)sem\_post(empty);  return 0;  } |

* 1. 编译运行



* 1. 运行效果

|  |
| --- |
| 2020-05-18 12-06-00 的屏幕截图  2020-05-18 12-06-07 的屏幕截图 |

1. 在信号量实验的基础上，为 Linux 0.11 增加共享内存功能，并将生产者—消费者程序移植到 Linux 0.11。
   1. 在include/unistd.h中添加用户系统调用接口和系统调用号

|  |
| --- |
| 2020-05-19 13-17-31 的屏幕截图2020-05-19 13-18-56 的屏幕截图 |

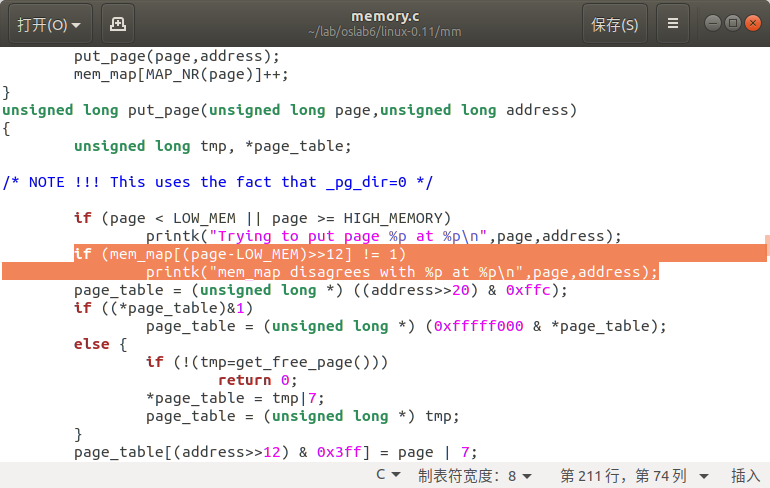
* 1. 在kernel/system\_call.s中修改调用总数

|  |
| --- |
| 2020-05-19 13-20-09 的屏幕截图 |

* 1. 修改 include/linux/sys.h的函数调用表

|  |
| --- |
| 2020-05-19 13-21-23 的屏幕截图  2020-05-19 13-21-45 的屏幕截图 |

注：要在memory.c中写，因为要修改mem\_map中的数据，防止共享内存的一个进程退出后释放该区域，另一个进程退出再释放，提示trying to free free page并卡死；实际上linux0.11并未实现共享内存，mem\_map的值大于1会有警告，注释掉相关代码就没警告了，本次实验暂不注释，但不影响使用，警告的相关代码如下图



* 1. put\_shm\_page编写和对应的头文件，填写线性地址 address 对应的页目录和页表

|  |
| --- |
| 2020-05-19 22-47-53 的屏幕截图 |
| 2020-05-19 22-39-10 的屏幕截图 |

* 1. 编写函数实现系统调用 shm.c

|  |
| --- |
| #include<linux/mm.h>  #include<linux/sched.h>  #include<unistd.h>  #define SHM\_TABLE\_SIZE 16  struct shm\_table  {  unsigned int key;  unsigned int size;  int used;  unsigned long p\_address;  }shmt[SHM\_TABLE\_SIZE]={};  int sys\_shmget(unsigned int key, unsigned int size)  {  int i;  if(size>PAGE\_SIZE)return -1;  for(i=0;i<SHM\_TABLE\_SIZE;i++)  if(shmt[i].used&&key==shmt[i].key)  return i;  for(i=0;i<SHM\_TABLE\_SIZE;i++)  if(!shmt[i].used)  {  shmt[i].key=key;  shmt[i].size=size;  shmt[i].used=1;  shmt[i].p\_address=get\_free\_page();  return i;  }  return -1;  }  void\* sys\_shmat(int shmid)  {  if(!shmt[shmid].used||shmid<0||shmid>SHM\_TABLE\_SIZE)return -1;  current->brk += PAGE\_SIZE;  put\_shm\_page(shmt[shmid].p\_address, get\_base(current->ldt[2])+current->brk);  current->brk+=+PAGE\_SIZE;  return (void \*)(current->brk - PAGE\_SIZE);  } |

* 1. 修改makefile

|  |
| --- |
| 2020-05-19 13-23-48 的屏幕截图2020-05-19 13-24-40 的屏幕截图 |

* 1. 替换内核头文件

|  |
| --- |
| 2020-05-19 13-25-32 的屏幕截图 |

* 1. 改编后的程序，其中消费者

|  |
| --- |
| 生产者 |
| #define \_\_LIBRARY\_\_  #include<stdio.h>  #include<unistd.h>  #include<stdlib.h>  #include<fcntl.h>  #define N 3  #define M 10  #define BUFSIZE 2  \_syscall2(sem\_t \*, sem\_open, const char\*,name,unsigned int,value);  \_syscall1(int,sem\_wait,sem\_t \*,sem);  \_syscall1(int,sem\_post,sem\_t \*,sem);  \_syscall1(int,sem\_unlink,const char \*,name);  \_syscall2(int,shmget,unsigned int,key,unsigned int,size);  \_syscall1(int,shmat,int,shmid);  sem\_t \*empty=NULL,\*mutex=NULL,\*full=NULL;  char strbuf[32];  int main()  {  int i=0;  int \*p=NULL;  int \*pcnt=NULL;  int data=1;  int shmid;  int shmcntid;  #ifndef \_DEBUG\_  freopen("out.txt","w",stdout);  #endif  empty=sem\_open("empty",BUFSIZE);  mutex=sem\_open("mutex",1);  full=sem\_open("full",0);  #ifdef \_DEBUG\_  if(!(empty&&mutex&&full))  perror("producer open sem fail\r\n");  else  perror("producer open sem sucess\r\n");  #endif  shmid=shmget(1,(BUFSIZE)\*sizeof(int));  shmcntid=shmget(2,sizeof(int));  #ifdef \_DEBUG\_  if(-1==shmid||-1==shmcntid)  perror("producer shmget fail\r\n");  else  perror("producer shmget sucess\r\n");  #endif  p=(int\*)shmat(shmid);  pcnt=(int\*)shmat(shmcntid);  \*pcnt=0;  while(data<=M)  {  sem\_wait(empty);  sem\_wait(mutex);  #ifdef \_DEBUG\_  sprintf(strbuf,"%d put data %d\r\n",getpid(),data);  write(1,strbuf,strlen(strbuf));  #endif  p[data-1]=data;  data++;  sem\_post(mutex);  sem\_post(full);  }  #ifdef \_DEBUG\_  sprintf(strbuf,"producer %d end\r\n",getpid());  write(1,strbuf,strlen(strbuf));  #endif  return 0;  } |
| 消费者 |
| #define \_\_LIBRARY\_\_  #include<stdio.h>  #include<unistd.h>  #include<stdlib.h>  #include<fcntl.h>  #define N 3  #define M 500  #define BUFSIZE 10  \_syscall2(sem\_t \*, sem\_open, const char\*,name,unsigned int,value);  \_syscall1(int,sem\_wait,sem\_t \*,sem);  \_syscall1(int,sem\_post,sem\_t \*,sem);  \_syscall1(int,sem\_unlink,const char \*,name);  \_syscall2(int,shmget,unsigned int,key,unsigned int,size);  \_syscall1(void\*,shmat,int,shmid);  sem\_t \*empty=NULL,\*mutex=NULL,\*full=NULL;  char strbuf[32];  int main()  {  int i=0;  int shmid=0;  int shmcntid;  int endflag;  int \*p=NULL;  int \*pcnt=NULL;  int \*pendflag=NULL;  #ifndef \_DEBUG\_  freopen("./out.txt","a",stdout);  #endif  #ifdef \_DEBUG\_  if(!(empty&&mutex&&full))  perror("consumer open sem fail\r\n");  else  perror("consumer open sem sucess\r\n");  #endif  shmid=shmget(1,(BUFSIZE)\*sizeof(int));  shmcntid=shmget(2,sizeof(int));  #ifdef \_DEBUG\_  if(-1==shmid||-1==shmcntid)  perror("consumer shmget fail\r\n");  else  perror("consumer shmget sucess\r\n");  #endif  p=(int\*)shmat(shmid);  pcnt=(int\*)shmat(shmcntid);  while(\*pcnt<M)  {  sem\_wait(full);  if(\*pcnt>=M)break;  sem\_wait(mutex);  fflush(stdout);  #if defined \_DEBUG\_  sem\_getvalue(full,&fullvalue);  sprintf(strbuf,"p:%d d:%d@%d r:%d\r\n",getpid(),p[\*pcnt],\*pcnt,fullvalue);  #else  sprintf(strbuf,"pid: %d\tdata:%d",getpid(),p[\*pcnt]);  puts(strbuf);    #endif  p[\*pcnt]=-1;  (\*pcnt)++;  fflush(stdout);  sem\_post(mutex);  sem\_post(empty);  }  #ifdef \_DEBUG\_  sprintf(strbuf,"consumer %d end\r\n",getpid());  write(1,strbuf,strlen(strbuf));  #endif  if(\*pcnt>=M)sem\_post(full);  return 0;  } |

* 1. 编译运行

|  |
| --- |
| 2020-05-19 22-54-14 的屏幕截图 |

* 1. 测试

|  |
| --- |
| 2020-05-19 23-01-18 的屏幕截图2020-05-19 23-04-36 的屏幕截图 |