|  |
| --- |
| 实验目的：  1. 理解Nachos的信号量是如何实现的；  2. 生产者/消费者问题是如何用信号量实现的；  3. 在Nachos中是如何创建并发线程的；  4. 在Nachos下是如何测试和debug的。 |
| 硬件环境：  惠普品牌型号笔记本  Intel Core i5-8300 CPU  8GB内存  512GB SSD |
| 软件环境：  宿主机：Windows 10 21H1 64位  虚拟机软件：VMware Workstation Pro 16.1.2 build-17966106  Linux：Ubuntu 14.04.6 LTS Desktop i386 (Trusty Tahr)  gcc/g++：(Ubuntu 4.8.4-2ubuntu1~14.04.4) 4.8.4  MIPS交叉编译器：gcc-2.8.1-mips.tar.gz  Nachos：Nachos-3.4-UALR-LW |
| 实验步骤与内容：  目录：  [lab3 使用信号量解决生产者/消费者同步问题](#lab3-使用信号量解决生产者消费者同步问题)  [3.1 实验内容](#Xd36cdcc790fa65a5358eff8b8468d1e76d94354)  [3.2 实验思路](#X7b15d6137d4a5439fbef98236499d730934fb6c)  [3.3 实验代码](#Xdaaf9c893e9ea153262de154730ba7c05e48438) lab3 使用信号量解决生产者/消费者同步问题3.1 实验内容 使用操作系统信号量机制，编写程序解决生产者/消费者同步问题。多个生产者和消费者线程访问在共享内存中的环形缓冲。生产者生产产品并将它放入环形缓冲，同时消费者从缓冲中取出产品并消费。当缓冲区满时生产者阻塞并且当缓冲区有空时生产者又重新 工作。类似的，消费者当缓冲区空时阻塞并且当缓冲区有产品时又重新工作。显然， 生产者和消费者需要一种同步机制以协调它们的工作。  **包括：**   1. 理解Nachos的信号量是如何实现的； 2. 生产者/消费者问题是如何用信号量实现的； 3. 在Nachos中是如何创建并发线程的； 4. 在Nachos下是如何测试和debug的。  3.2 实验思路 lab3中的文件包括mian.cc，prodcons++.cc，ring.cc 和 ring.h。 文件ring.cc和ring.h定义和实现了一个为生产者和消费者使用的环形缓冲类ring。ring 类的分析见1.8。这两个文件已经完成了，不需要我们作任何改动。在这个目录中的 main.cc 是../threads 目录中 main.cc 的改进版，我们也不必改动了。 在新的 main.cc 中调用了函数 ProdCons()而不是函数ThreadTest()。  函数ProdCons()定义在文件 prodcons++.cc 中。假设这个文件包含着建立两个生产者和两个消费者线程以及实现两个生产者消费者问题算法的代码。不过，这个文件并不完整，我们的实验任务就是完成prodcons.cc，构造一个能工作的生产者/消费者问题算法。  重点分析prodcons.cc，源代码如下：  // prodcons++.cc // C++ version of producer and consumer problem using a ring buffer. // // Create N\_PROD producer threads and N\_CONS consumer thread.  // Producer and consumer threads are communicating via a shared // ring buffer object. The operations on the shared ring buffer // are synchronized with semaphores. //  //  #include <stdio.h> #include <stdlib.h> #include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> #include <unistd.h> #include "copyright.h" #include "system.h"  #include "synch.h" #include "ring.h"  #define BUFF\_SIZE 3 // the size of the round buffer #define N\_PROD 2 // the number of producers  #define N\_CONS 2 // the number of consumers #define N\_MESSG 5 // the number of messages produced by each producer #define MAX\_NAME 16 // the maximum lengh of a name  #define MAXLEN 48  #define LINELEN 24   Thread \*producers[N\_PROD]; //array of pointers to the producer Thread \*consumers[N\_CONS]; // and consumer threads;  char prod\_names[N\_PROD][MAX\_NAME]; //array of charater string for prod names char cons\_names[N\_CONS][MAX\_NAME]; //array of charater string for cons names  Semaphore \*nempty, \*nfull; //two semaphores for empty and full slots Semaphore \*mutex; //semaphore for the mutual exclusion   Ring \*ring;  void Producer(\_int which){} void Consumer(\_int which){} void ProdCons(){}  该源文件主要定义了以下宏与全局变量：   |  |  | | --- | --- | | 宏或全局变量 | 含义 | | BUFF\_SIZE=3 | 环形缓冲区大小为3 | | N\_PROD=2 | 生产者线程数量 | | N\_CONS=2 | 消费者线程数量 | | N\_MESSG 5 | 每个生产者线程产生的消息数 | | MAX\_NAME=16 | 线程名称最大长度 | | MAXLEN、LINELEN | 与最终输出的文件有关 | | Thread \*producers[N\_PROD] | 生产者线程数组 | | Thread \*consumers[N\_CONS] | 消费者线程数组 | | prod\_names | 生产者名称数组 | | cons\_names | 消费者名称数组 | | Semaphore \*nempty | 缓冲区已生产空间信号量 | | Semaphore \*nfull | 缓冲区未生产空间信号量 | | Semaphore \*mutex | 缓冲区互斥锁 | | Ring \*ring | 环形缓冲区 |   该源文件还定义了三个方法：   * Producer(\_int which)：生产者线程调用的函数 * void Producer(\_int which) {  int num;  slot \*message = new slot(0,0);  // This loop is to generate N\_MESSG messages to put into to ring buffer // by calling ring->Put(message). Each message carries a message id  // which is represened by integer "num". This message id should be put  // into "value" field of the slot. It should also carry the id  // of the producer thread to be stored in "thread\_id" field so that  // consumer threads can know which producer generates the message later // on. You need to put synchronization code // before and after the call ring->Put(message). See the algorithms in // page 182 of the textbook.   for (num = 0; num < N\_MESSG ; num++) {  // Put the code to prepare the message here.  // ...   // Put the code for synchronization before ring->Put(message) here.  // ...   ring->Put(message);   // Put the code for synchronization after ring->Put(message) here.  // ...   } } * 开头先初始化了一个消息插槽作为生产者生产的消息（消息包括消息的值与创建该消息的线程），对于每个生产者进程来说循环N\_MESSG次，我们需要在ring->Put(message)前加入对缓冲区未生产信号量判断，后加上对缓冲区已生产信号量的判断，同时为了防止写冲突应该加上一个互斥信号量，在一个线程写入时其他线程无法写入。 * Consumer(\_int which)：消费者线程调用的函数 * void Consumer(\_int which) {  char str[MAXLEN];  char fname[LINELEN];  int fd;    slot \*message = new slot(0,0);   // to form a output file name for this consumer thread.  // all the messages received by this consumer will be recorded in   // this file.  sprintf(fname, "tmp\_%d", which);   // create a file. Note that this is a UNIX system call.  if ( (fd = creat(fname, 0600) ) == -1)   {  perror("creat: file create failed");  exit(1);  }    for (; ; ) {  // Put the code for synchronization before ring->Get(message) here.  // ...   ring->Get(message);   // Put the code for synchronization after ring->Get(message) here.  // ...    // form a string to record the message  sprintf(str,"producer id --> %d; Message number --> %d;\n",   message->thread\_id,  message->value);  // write this string into the output file of this consumer.   // note that this is another UNIX system call.  if ( write(fd, str, strlen(str)) == -1 ) {  perror("write: write failed");  exit(1);  }  } } * 前面是关于消费者读取到消息并且写入文件的配置，for循环内是我们要修改的地方，同样需要在ring->Get(message);前后添加对缓冲区已生产空间信号量与缓冲区未生产空间信号量的PV操作 * ProdCons()：main.cc调用的方法，实现生产者消费者问题初始化 * void ProdCons() {  int i;  DEBUG('t', "Entering ProdCons");   // Put the code to construct all the semaphores here.  // ....   // Put the code to construct a ring buffer object with size   //BUFF\_SIZE here.  // ...     // create and fork N\_PROD of producer threads   for (i=0; i < N\_PROD; i++)   {  // this statemet is to form a string to be used as the name for   // produder i.   sprintf(prod\_names[i], "producer\_%d", i);   // Put the code to create and fork a new producer thread using  // the name in prod\_names[i] and   // integer i as the argument of function "Producer"  // ...   };   // create and fork N\_CONS of consumer threads   for (i=0; i < N\_CONS; i++)   {  // this statemet is to form a string to be used as the name for   // consumer i.   sprintf(cons\_names[i], "consumer\_%d", i);  // Put the code to create and fork a new consumer thread using  // the name in cons\_names[i] and   // integer i as the argument of function "Consumer"  // ...   }; } * 我们需要添加对信号量的初始化、对缓冲区的初始化、以及对生产者线程与消费者线程的创建到转变为running态（调用fork）  3.3 实验代码 修改后的prodcons++.cc如下：  // prodcons++.cc // C++ version of producer and consumer problem using a ring buffer. // // Create N\_PROD producer threads and N\_CONS consumer thread.  // Producer and consumer threads are communicating via a shared // ring buffer object. The operations on the shared ring buffer // are synchronized with semaphores. //  //  // Copyright (c) 1995 The Regents of the University of Southern Queensland. // All rights reserved. See copyright.h for copyright notice and limitation  // of liability and disclaimer of warranty provisions.  #include <stdio.h> #include <stdlib.h> #include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> #include <unistd.h> #include "copyright.h" #include "system.h"  #include "synch.h" #include "ring.h"  #define BUFF\_SIZE 3 // the size of the round buffer #define N\_PROD 2 // the number of producers  #define N\_CONS 2 // the number of consumers #define N\_MESSG 5 // the number of messages produced by each producer #define MAX\_NAME 16 // the maximum lengh of a name  #define MAXLEN 48  #define LINELEN 24   Thread \*producers[N\_PROD]; //array of pointers to the producer Thread \*consumers[N\_CONS]; // and consumer threads;  char prod\_names[N\_PROD][MAX\_NAME]; //array of charater string for prod names char cons\_names[N\_CONS][MAX\_NAME]; //array of charater string for cons names  Semaphore \*nempty, \*nfull; //two semaphores for empty and full slots 未生产空间与已生产空间 Semaphore \*mutex; //semaphore for the mutual exclusion 互斥信号量   Ring \*ring;// 环缓冲区  void Producer(\_int which) {  int num;  slot \*message = new slot(0,0);// 初始化一个消息slot，但未设置消息具体内容以及生产者   for (num = 0; num < N\_MESSG ; num++) {  // Put the code to prepare the message here.  // ...  message->value = num;// 生产消息的值  message->thread\_id = which;// 生产者线程id  // Put the code for synchronization before ring->Put(message) here.  // ...   nempty->P();// 执行P操作 未生产空间-1  mutex->P();// 开启互斥锁   ring->Put(message);   // Put the code for synchronization after ring->Put(message) here.  // ...  nfull->V();// 执行V操作 已生产空间+1  mutex->V();// 关闭互斥锁   }  }  void Consumer(\_int which) {  char str[MAXLEN];  char fname[LINELEN];  int fd;    slot \*message = new slot(0,0);   // to form a output file name for this consumer thread.  // all the messages received by this consumer will be recorded in   // this file.  sprintf(fname, "tmp\_%d", which);   // create a file. Note that this is a UNIX system call.  if ( (fd = creat(fname, 0600) ) == -1)   {  perror("creat: file create failed");  exit(1);  }    for (; ;) {   // Put the code for synchronization before ring->Get(message) here.  // ...  nfull->P();// 执行P操作，已生产空间-1  mutex->P();// 开启互斥锁   ring->Get(message);   // Put the code for synchronization after ring->Get(message) here.  // ...   nempty->V();// 执行V操作，未生产空间+1  mutex->V();// 关闭互斥锁   // form a string to record the message  sprintf(str,"producer id --> %d; Message number --> %d;\n",  message->thread\_id,  message->value);  // write this string into the output file of this consumer.   // note that this is another UNIX system call.  if ( write(fd, str, strlen(str)) == -1 ) {  perror("write: write failed");  exit(1);  }  }  }  void ProdCons() {  int i;  DEBUG('t', "Entering ProdCons");   // Put the code to construct all the semaphores here.  // ....  nempty = new Semaphore("nempty",BUFF\_SIZE);// 初始化缓冲区未生产空间=BUFF\_SIZE  nfull = new Semaphore("nfull",0);// 初始化缓冲区已生产空间=0  mutex = new Semaphore("mutex",1);// 初始化互斥锁   // Put the code to construct a ring buffer object with size   //BUFF\_SIZE here.  // ...   ring = new Ring(BUFF\_SIZE);   // create and fork N\_PROD of producer threads   for (i=0; i < N\_PROD; i++)   {  // this statemet is to form a string to be used as the name for   // produder i.   sprintf(prod\_names[i], "producer\_%d", i);   // Put the code to create and fork a new producer thread using  // the name in prod\_names[i] and   // integer i as the argument of function "Producer"  // ...  producers[i] = new Thread(prod\_names[i]);// 初始化生产者线程  producers[i]->Fork(Producer, i);// 线程fork 执行Producer(i)  };   // create and fork N\_CONS of consumer threads   for (i=0; i < N\_CONS; i++)   {  // this statemet is to form a string to be used as the name for   // consumer i.   sprintf(cons\_names[i], "consumer\_%d", i);  // Put the code to create and fork a new consumer thread using  // the name in cons\_names[i] and   // integer i as the argument of function "Consumer"  // ...  consumers[i]=new Thread(cons\_names[i]);// 初始化消费者线程  consumers[i]->Fork(Consumer,i);// 线程fork 执行Consume(i)  }; } |
| 结论分析与体会：  注意，本次运行操作系统应添加命令行参数./nachos -rs 123，表示随机切换线程。运行后生成两个文件表示0号消费者与1号消费者读取的消息，分别读取可以看到，两个消费者将生产者生产的所有消息随机有顺序的读出：    输入./nachos -d t转换为DEBUG模式，t**表示只读取有关线程的DEBUG信息**： |