|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **武汉大学国家网络安全学院教学实验报告** | | | | |
| 课程  名称 | 操作系统及安全设计 | | 实验日期 | 2024.04.27 |
| 实验  名称 | 哲学家就餐问题的pthreads解决方案 | | 实验序号 | 4 |
| 姓 名 | 学 号 | 专业 | 年级-班 | 成绩 |
| 张楚豪 | 2022302191636 | 信息安全 | 2022级10班 |  |
| 程序 | 2022302181131 | 信息安全 | 2022级4班 |  |
| 王竟霖 | 2022302191625 | 网络空间  安全 | 2022级10班 |  |
| 谢志永 | 2022302191205 | 网络空间  安全 | 2022级10班 |  |
| 1. 实验目的及实验内容   （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析） | | | | |
| 1、实验目的：  通过使用Pthreads 或 Windows 线程API来模拟哲学家就餐问题，从而探讨并发编程中的死锁问题以及如何使用同步机制来解决死锁。  2、实验内容：  哲学家就餐问题是一个典型的并发控制问题，描述了五位哲学家围坐在圆桌前，每人面前有一只筷子，共有五只筷子。哲学家需要先拿起左右两只筷子才能进餐，吃完后放下筷子开始思考。  ①使用互斥锁 pthread\_mutex\_t 和条件变量 pthread\_cond\_t 来控制哲学家对筷子的获取和归还。  ②pickup\_forks 函数模拟哲学家进餐的过程，首先通过互斥锁保证最多只有4位哲学家同时拿起筷子，然后分别获取左右两只筷子，如果某只筷子被占用，则等待条件变量的唤醒。  ③return\_forks 函数模拟哲学家进餐完毕后放下筷子的过程，通过互斥锁和条件变量实现筷子的归还和等待哲学家的唤醒。  ④主函数 main 创建了五个线程分别代表五位哲学家，每个线程循环调用 pickup\_forks 和 return\_forks 函数模拟哲学家的行为，直到键盘中断退出。  3、原理分析：  哲学家进餐问题有多种解决方案，如：至多允许4人同时进餐；限制按一定资源顺序申请资源；当左右两只筷子均可用时才能进餐等等。这里选择前两种方法实现。  ①互斥锁和条件变量：  使用 pthread\_mutex\_init 初始化互斥锁，互斥锁可以用于保护代码的临界区，使用互斥锁可以实现类似原子操作。  pthread\_cond\_init 用于初始化条件变量，涉及pthread\_cond\_wait和pthread\_cond\_signal两个函数。  pthread\_cond\_wait()用来等待一个条件变量，一个条件变量标志着一种等待情况，因此需要对所有chopstick以及其他辅助变量（如allow）分别建立一个条件变量。（注意由于被唤醒后不一定能满足检查条件，wait语句须放在循环中，以保证唤醒后重新检查条件）  pthread\_cond\_signal()用来唤醒等待一个条件变量的线程。  这两个函数与互斥锁结合可以实现类似PV操作。  ②方案一：限制进餐人数：  当哲学家想吃饭时，首先检查allow确定正在使用筷子的人数，如果已经为0（无可用资源，即有4人使用筷子），再使用筷子可能会造成死锁，需调用 pthread\_cond\_wait 进入等待状态，等待其他哲学家归还筷子。  一旦允许拿起筷子，将 allow 减一（资源减少一个），并解锁 mutex\_allow。  进餐结束后，哲学家归还筷子，表明自己使用筷子结束，allow加一。  在过程中，可以使用 sleep(2) 模拟进餐和思考过程。  ③方案二：限制申请资源顺序：  根据有序资源分配法，按照资源层次递增的顺序申请资源，相反顺序释放资源可以避免死锁。只需在使用、释放筷子前先计算first,second，即筷子的前后申请顺序（以数组下标大小为排序依据），按first,second的顺序使用筷子，second，first的顺序释放筷子即可。  ④callback 函数：  每个哲学家作为一个线程执行，不断循环调用 pickup\_forks 和 return\_forks 函数，直到程序被中断。  pickup\_forks即申请筷子和用餐的过程，return\_forks即释放筷子和思考的过程。  ⑤主函数 main：  创建五个线程，分别代表五位哲学家，每个线程在创建后将执行 callback 函数。  使用 pthread\_join 使得父进程等待子线程执行完毕并回收。 | | | | |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤） | | | | |
| 1、实验环境：  硬件环境：  ASUS TUF Gaming A15 FA507RM\_FA507RM  CPU：AMD Ryzen 7 6800H with Radeon Graphics 3.20 GHz  GPU：NVIDIA GeForce RTX 3060 Laptop GPU  软件环境：  物理机操作系统：Windows 11 家庭中文版 22H2  虚拟化软件：VMware Workstation Pro 17.5.1  虚拟机操作系统：OpenEuler 22.63(LTS-SF3)   1. 实验步骤：   方案一：限制人数   1. 导入必要的头文件，包括 <pthread.h>、<stdio.h>和<unistd.h>`。 2. 定义全局变量 allow 表示最多允许的哲学家数目，并初始化筷子的使用情况数组 chopstick。 3. 创建互斥锁和条件变量，包括一个用于控制哲学家数目的互斥锁 mutex\_allow 和条件变量 cond\_allow，以及用于每个筷子的互斥锁和条件变量数组。 4. 实现 pickup\_forks 函数，用于哲学家拿取筷子的过程，其中使用互斥锁和条件变量来保证筷子的可用性。 5. 实现 return\_forks 函数，用于哲学家归还筷子的过程，同样使用互斥锁和条件变量来确保筷子的释放。 6. 实现线程执行函数 callback，每个哲学家作为一个线程，循环执行拿取和归还筷子的过程。 7. 在 main 函数中初始化互斥锁和条件变量，创建哲学家线程，并等待线程结束   # include <pthread.h>  # include <stdio.h>  # include <unistd.h> //提供sleep函数  /\*  解决方案：最多允许4个人同时使用筷子  可以允许哲学家拿起一边筷子等待而不归还，  （如：拿起左边筷子发现右边筷子没了，但是不放下左边筷子）  只要拿起筷子的人数小于等于4就不会死锁  \*/  /\*  伪代码：  do{  P(allow); //allow = 4  P(chopstick[i]);  P(chopstick[(i+1)%5]);  eat();  V(chopstick[i]);  V(chopstick[(i+1)%5]);  V(allow);  think();  }while(true);  \*/  //最多允许4个人同时使用筷子来防止死锁  int allow = 4;  //表示筷子使用情况  int chopstick[5] = {1,1,1,1,1};  //创建互斥锁  pthread\_mutex\_t mutex\_allow;  pthread\_mutex\_t mutex\_chopstick[5];  //创建条件变量  pthread\_cond\_t cond\_allow;  pthread\_cond\_t cond\_chopstick[5];  //表示执行次数  int run = 1; //可以不用  //想吃饭调用  void pickup\_forks(int num)  {  printf("哲学家%d想要进餐\n",num);  pthread\_mutex\_lock(&mutex\_allow);  while(allow <= 0){  printf("哲学家%d发现使用筷子人数过多，等待中\n",num);  pthread\_cond\_wait(&cond\_allow,&mutex\_allow); //等待少于4人使用筷子  printf("哲学家%d被唤醒\n",num);  }  allow -= 1;  pthread\_mutex\_unlock(&mutex\_allow);  int left = num;  int right = (num + 1) % 5;  pthread\_mutex\_lock(&(mutex\_chopstick[left]));  while(chopstick[left] <= 0){  printf("哲学家%d发现左边没有筷子，等待中\n",num);  pthread\_cond\_wait(&(cond\_chopstick[left]),&(mutex\_chopstick[left])); //等待左边筷子空闲  printf("哲学家%d被唤醒\n",num);  }  chopstick[left] = 0;  pthread\_mutex\_unlock(&(mutex\_chopstick[left]));  pthread\_mutex\_lock(&(mutex\_chopstick[right]));  while(chopstick[right] <= 0){  printf("哲学家%d发现右边没有筷子，等待中\n",num);  pthread\_cond\_wait(&(cond\_chopstick[right]),&(mutex\_chopstick[right])); //等待右边筷子空闲  printf("哲学家%d被唤醒\n",num);  }  chopstick[right] = 0;  pthread\_mutex\_unlock(&(mutex\_chopstick[right]));  //sleep两秒来模拟吃饭  printf("哲学家%d进餐中\n",num);  printf("此时：allow: %d chopstick: %d %d %d %d %d\n",allow,chopstick[0],chopstick[1],chopstick[2],chopstick[3],chopstick[4]);  sleep(2);  }  //吃完后调用  void return\_forks(int num)  {  printf("哲学家%d进餐完毕\n",num);  int left = num;  int right = (num + 1) % 5;  pthread\_mutex\_lock(&(mutex\_chopstick[left]));  chopstick[left] = 1;  pthread\_cond\_signal(&(cond\_chopstick[left])); //唤醒等待左边筷子的线程  pthread\_mutex\_unlock(&(mutex\_chopstick[left]));  pthread\_mutex\_lock(&(mutex\_chopstick[right]));  chopstick[right] = 1;  pthread\_cond\_signal(&(cond\_chopstick[right])); //唤醒等待右边筷子的线程  pthread\_mutex\_unlock(&(mutex\_chopstick[right]));  pthread\_mutex\_lock(&mutex\_allow);  allow += 1;  pthread\_cond\_signal(&cond\_allow); //唤醒等待使用筷子的线程  pthread\_mutex\_unlock(&mutex\_allow);  //sleep两秒来模拟思考  printf("哲学家%d思考中\n",num);  printf("此时：allow: %d chopstick: %d %d %d %d %d\n",allow,chopstick[0],chopstick[1],chopstick[2],chopstick[3],chopstick[4]);  sleep(2);  }  //每个线程执行的函数  void \*callback(void \*arg)  {  int num = (int)arg;  /\*  for(int i = 0; i < run; i++) //模拟执行次数为run  {  pickup\_forks(num);  return\_forks(num);  }  \*/  //修改为一直执行，直到键盘中断  while(1)  {  pickup\_forks(num);  return\_forks(num);  }  }  int main()  {  int philosopher[5] = {0,1,2,3,4};  pthread\_t tid[5];  pthread\_attr\_t attr[5];  //初始化互斥锁和条件变量  pthread\_mutex\_init(&mutex\_allow,NULL);  pthread\_cond\_init(&cond\_allow,NULL);  for(int i = 0; i < 5; i++)  {  pthread\_mutex\_init(&mutex\_chopstick[i],NULL);  pthread\_cond\_init(&cond\_chopstick[i],NULL);  }  //每个哲学家作为一个单独线程来运行  for(int i = 0; i < 5; i++)  {  //初始化属性  pthread\_attr\_init(&(attr[i]));  //创建线程  pthread\_create(&(tid[i]),&(attr[i]),callback,(void \*)i);  }  //等待线程结束  for(int i = 0; i < 5; i++)  {  pthread\_join(tid[i],NULL);  }  return 0;  }    图3-1.1 方案一哲学家思考进餐一轮结果图      图3-1.2 方案一哲学家思考进餐若干轮结果图  方案二：限制资源申请  ①初始化互斥锁和条件变量： 在 main() 函数中，首先初始化了五把互斥锁和五个条件变量。  ②创建线程： 在 main() 函数中，创建了五个线程，每个线程代表一个哲学家。在每个线程中，会执行 callback 函数，参数是哲学家的编号。  ③哲学家就餐行为的实现：  （1）pickup\_forks() 函数实现哲学家拿起筷子的行为。它首先计算资源的先后申请顺序，尝试按顺序获取两只筷子，如果其中一只筷子被其他哲学家占用，则进入等待状态，直到能够获取到两只筷子。  （2）return\_forks() 函数实现哲学家放下筷子的行为。它首先计算资源的先后释放顺序，按顺序将哲学家手中的两只筷子放回桌子上，并唤醒等待这两只筷子的其他哲学家。  ④线程执行： 每个线程执行 callback 函数，其中循环执行 pickup\_forks() 和 return\_forks() 直到程序被中断。  ⑤等待线程结束： 在 main() 函数中，使用 pthread\_join() 等待所有线程执行结束。  # include <pthread.h>  # include <stdio.h>  # include <unistd.h> //提供sleep函数  /\*  方法二：对资源进行分级  级数小的先申请，后释放，以此避免死锁  \*/  /\*  伪代码：  do{  first,second;  P(chopstick[first]);  P(chopstick[second]);  eat();  V(chopstick[second]);  V(chopstick[first]);  think();  }while(true);  \*/  //表示筷子使用情况  int chopstick[5] = {1,1,1,1,1};  //创建互斥锁  pthread\_mutex\_t mutex\_chopstick[5];  //创建条件变量  pthread\_cond\_t cond\_chopstick[5];  //表示执行次数  int run = 1;  //想吃饭调用  void pickup\_forks(int num)  {  printf("哲学家%d想要进餐\n",num);  int left = num;  int right = (num + 1) % 5;  int first = (left < right) ? left : right;  int second = (left < right) ? right : left;  pthread\_mutex\_lock(&(mutex\_chopstick[first]));  while(chopstick[first] <= 0){  printf("哲学家%d发现%d号筷子被占用，等待中\n",num,first);  pthread\_cond\_wait(&(cond\_chopstick[first]),&(mutex\_chopstick[first]));  printf("哲学家%d被唤醒\n",num);  }  chopstick[first] = 0;  pthread\_mutex\_unlock(&(mutex\_chopstick[first]));  pthread\_mutex\_lock(&(mutex\_chopstick[second]));  while(chopstick[second] <= 0){  printf("哲学家%d发现%d号筷子被占用，等待中\n",num,second);  pthread\_cond\_wait(&(cond\_chopstick[second]),&(mutex\_chopstick[second]));  printf("哲学家%d被唤醒\n",num);  }  chopstick[second] = 0;  pthread\_mutex\_unlock(&(mutex\_chopstick[second]));  //sleep两秒来模拟吃饭  printf("哲学家%d进餐中\n",num);  printf("此时：chopstick: %d %d %d %d %d\n",chopstick[0],chopstick[1],chopstick[2],chopstick[3],chopstick[4]);  sleep(2);  }  //吃完后调用  void return\_forks(int num)  {  printf("哲学家%d进餐完毕\n",num);  int left = num;  int right = (num + 1) % 5;  int first = (left < right) ? left : right;  int second = (left < right) ? right : left;  pthread\_mutex\_lock(&(mutex\_chopstick[second]));  chopstick[second] = 1;  pthread\_cond\_signal(&(cond\_chopstick[second])); //唤醒等待second号筷子的线程  pthread\_mutex\_unlock(&(mutex\_chopstick[second]));  pthread\_mutex\_lock(&(mutex\_chopstick[first]));  chopstick[first] = 1;  pthread\_cond\_signal(&(cond\_chopstick[first])); //唤醒等待first号筷子的线程  pthread\_mutex\_unlock(&(mutex\_chopstick[first]));  //sleep两秒来模拟思考  printf("哲学家%d思考中\n",num);  printf("此时：chopstick: %d %d %d %d %d\n",chopstick[0],chopstick[1],chopstick[2],chopstick[3],chopstick[4]);  sleep(2);  }  //每个线程执行的函数  void \*callback(void \*arg)  {  int num = (int)arg;  /\*  for(int i = 0; i < run; i++) //模拟执行次数为run  {  pickup\_forks(num);  return\_forks(num);  }  \*/  //修改为一直执行，直到键盘中断  while(1)  {  pickup\_forks(num);  return\_forks(num);  }  }  int main()  {  int philosopher[5] = {0,1,2,3,4};  pthread\_t tid[5];  pthread\_attr\_t attr[5];  //初始化互斥锁和条件变量  for(int i = 0; i < 5; i++)  {  pthread\_mutex\_init(&mutex\_chopstick[i],NULL);  pthread\_cond\_init(&cond\_chopstick[i],NULL);  }  //每个哲学家作为一个单独线程来运行  for(int i = 0; i < 5; i++)  {  //初始化属性  pthread\_attr\_init(&(attr[i]));  //创建线程  pthread\_create(&(tid[i]),&(attr[i]),callback,(void \*)i);  }  //等待线程结束  for(int i = 0; i < 5; i++)  {  pthread\_join(tid[i],NULL);  }  return 0;  }    图3-1.3 方案二哲学家思考进餐结果图 | | | | |
| 1. 实验过程分析   （实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） | | | | |
| 1. 实验分工：   程序：  编写方案一代码  张楚豪：  编写方案二代码  王竟霖、谢志永：  代码汇总润色与实验报告撰写   1. 实验故障：     开始时使用&i传参发现传参错误，导致数组越界（接收到了大于等于5的下标）  查阅资料https://blog.csdn.net/modi000/article/details/104728979发现  线程在创建时不能传递一个变化的参数，因此随着for（）循环内部i的递增，等到callback函数读地址时i的值可能已经发生了变化，容易产生错误，因此必须强制把一个值当成地址来传参。 | | | | |
| 1. 实验结果总结   （对实验结果进行分析，完成思考题目，总结实验成员每人的实验心得体会，并提出实验的改进意见）  （1）实验结果分析：  本次实验模拟了每个哲学家轮流进行思考和进餐，每次进餐前会先获取两只筷子，进餐完毕后会将两只筷子放回桌上的情形。这保证了每个哲学家都有机会进餐，避免了饥饿状态。同时通过过使用互斥锁和条件变量，确保了在获取筷子时不会发生死锁情况。当某只筷子被其他哲学家占用时，当前哲学家会进入等待状态，直到能够获取到两只筷子才开始进餐。在实验中，每个哲学家进餐时会休眠两秒，模拟了进餐过程的耗时。这样可以观察到哲学家们的就餐行为是交替进行的，而不是同时进行的，从而保证了资源（筷子）的合理利用。最终我们小组分别完成了限制人数和限制资源申请这两种哲学家问题的解决方案。  （2）实验心得体会：  通过实践，我们深入掌握了互斥锁和条件变量的概念和用法。互斥锁保证了在同一时间只有一个线程可以访问共享资源，而条件变量则用于线程间的通信，使得线程可以等待特定条件的发生。这两者结合起来，能够有效地解决并发编程中的竞态条件和数据竞争问题。除此之外，这个实验也让我们更加深入地理解了死锁问题以及如何避免死锁。在哲学家就餐问题中，如果每个哲学家都按照固定的顺序去拿筷子，那么很容易出现死锁的情况，即每个哲学家都拿起了一只筷子而无法再拿到另一只筷子。通过合理地设计算法和使用同步工具，我们可以避免这种情况的发生，确保系统能够顺利执行。总的来说，通过这个实验，不仅加深了对并发编程的理解，还学到了如何有效地解决并发编程中的常见问题。  （3）实验改进意见：  针对此次实验，我们可以在资源利用率优化、哲学家思考时间可配置化、代码结构优化、优化等待筷子的逻辑、异常处理和错误处理等方面进行优化。除此之外，我们也可以尝试使用其他的思考角度和算法进行哲学家进餐问题的解决，也可以融会贯通，应用类比的思想解决如读者/写者、生产者/消费者、理发师（or睡觉助教）等类似问题。我们需要学会灵活应对并发编程中的一些挑战和技巧。例如，如何合理地设计线程之间的通信和同步机制，以及如何确保资源的合理利用和公平竞争。而如何避免线程间的竞态条件和数据竞争，如何正确地使用同步工具和设计线程之间的通信机制等也是我们在未来的不断实践中应该多加关注和重视的要点。通过不断地实践和探索，这些经验和技巧在今后的编程工作中将会派上很大的用场，我们会继续保持学习和探索的态度，勤加练习实践，不断提升自己在编程领域的能力和水平。 | | | | |
| 1. 教师评语 | | | | |
| 全组评语： | | | | |
|  | | | | |
| 个人评语（请填写本组同学姓名，队长排第一，其他同学随后） | | | | |
| 1. 张楚豪 2. 程序 3. 王竟霖 4. 谢志永 | | | | |
| 教师签名： | | | | |