实验报告

东北林业大学

信息与计算机科学技术实验中心

|  |
| --- |
| 一、实验目的  **1. 掌握基于内存映射的进程通信方式；**  **2. 掌握信号量机制，能够用信号量实现进程之间的同步；**  **3.了解线程的通信及同步的基本方法。** |
| **二、实验环境**  实验环境：VMware, Ubuntu 22.04.5 LTS(内核版本2.6.18)  硬件环境：内存2G以上，硬盘空间20G以上 |
| **三、实验内容及结果**  1、编写程序，实现两个进程的通信，一个进程负责计数（循环计数执行10次），另一个进程负责输出，打印；要求第一个进程每记一次数，就二个进程就要输出一次。  （1）编写程序1.c  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <semaphore.h>  #include <fcntl.h>  #define SHM\_SIZE 1024  int main() {  pid\_t pid;  key\_t key = 1234; // 共享内存键值  int shmid;  int \*shared\_mem;  sem\_t \*sem\_count = sem\_open("/sem\_count", O\_CREAT, 0644, 1);  sem\_t \*sem\_print = sem\_open("/sem\_print", O\_CREAT, 0644, 0);  // 创建共享内存  if ((shmid = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666)) == -1) {  perror("shmget");  exit(1);  }  // 关联共享内存  if ((shared\_mem = shmat(shmid, NULL, 0)) == (int \*)-1) {  perror("shmat");  exit(1);  }  pid = fork();  if (pid < 0) {  perror("fork");  exit(1);  } else if (pid == 0) {  // 子进程负责输出  for (int i = 0; i < 10; i++) {  sem\_wait(sem\_print);  printf("Output process: Count = %d\n", \*shared\_mem);  sem\_post(sem\_count);  }  } else {  // 父进程负责计数  for (int i = 0; i < 10; i++) {  sem\_wait(sem\_count);  \*shared\_mem = i + 1;  printf("Counting process: Count = %d\n", \*shared\_mem);  sem\_post(sem\_print);  }  wait(NULL);  // 释放资源  sem\_unlink("/sem\_count");  sem\_unlink("/sem\_print");  shmctl(shmid, IPC\_RMID, NULL);  }  return 0;  }  （2）运行程序  2、用线程的同步，完成和1相同功能的程序。  （1）编写程序2.c  #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <semaphore.h>  #define LOOP\_COUNT 10  sem\_t sem\_count, sem\_print;  int shared\_data = 0;  void \*counting\_thread(void \*arg) {  for (int i = 0; i < LOOP\_COUNT; i++) {  sem\_wait(&sem\_count);  shared\_data = i + 1;  printf("Counting thread: Count = %d\n", shared\_data);  sem\_post(&sem\_print);  }  return NULL;  }  void \*printing\_thread(void \*arg) {  for (int i = 0; i < LOOP\_COUNT; i++) {  sem\_wait(&sem\_print);  printf("Printing thread: Count = %d\n", shared\_data);  sem\_post(&sem\_count);  }  return NULL;  }  int main() {  pthread\_t thread1, thread2;  sem\_init(&sem\_count, 0, 1);  sem\_init(&sem\_print, 0, 0);  pthread\_create(&thread1, NULL, counting\_thread, NULL);  pthread\_create(&thread2, NULL, printing\_thread, NULL);  pthread\_join(thread1, NULL);  pthread\_join(thread2, NULL);  sem\_destroy(&sem\_count);  sem\_destroy(&sem\_print);  return 0;  }  （2）运行程序 |
| **四、实验过程分析与讨论**  共享内存是一种高效的进程间通信机制，它允许多个进程通过内存段共享数据。本实验中，父进程与子进程通过共享内存传递计数值。信号量用于控制进程或线程间的同步，避免资源竞争。本实验中，通过信号量实现了两个进程的同步通信。线程是比进程更轻量级的执行单元，同一进程的多个线程共享相同的内存空间。因此，线程之间的通信比进程更高效。为避免多个线程同时访问共享数据造成混乱，使用信号量控制线程同步。父进程/计数线程执行P操作，保证子进程/打印线程完成后才能继续；子进程/打印线程执行V操作，通知父进程/计数线程可以继续。  但是在实验中的仍有很多容易错的地方  共享内存使用后未及时释放，导致内存泄漏。在父进程结束时，调用shmctl删除共享内存段。  对于进程信号量，使用sem\_open并指定初始值；  对于线程信号量，使用sem\_init并确保初始化正确。  多个线程访问共享数据可能导致读取到未更新的值。通过信号量控制对共享数据的访问，确保数据在一个线程操作完成后才被另一个线程访问。  本实验中，使用进程时，需要通过共享内存实现数据共享；使用线程时，共享数据直接位于进程内存空间中，相对简单。线程通信效率更高，但进程通信更适合需要更高隔离度的场景。  通过本实验，我学习了进程和线程同步的基本方法，加深了对信号量和共享内存的理解。实验中遇到的关键问题主要集中在同步逻辑的设计和资源的正确管理上。解决这些问题后，我更加清楚了如何在实际项目中使用这些工具和技术。  实验虽然简单，但其核心思想在分布式计算、并行编程中广泛应用。通过本次实验，我不仅巩固了理论知识，还锻炼了问题排查能力，为后续学习打下了坚实基础。 |
| 五、指导教师意见  ☐ 实验完成内容完整，分析及讨论充分，写作规范  ☐ 实验完成内容基本完整，分析及讨论基本充分，写作基本规范  ☐ 实验完成内容不完整，分析及讨论不充分，写作不规范  指导教师签字：侯畅  **2024年12月11日** |