《Linux系统及编程》综合报告

学号：

姓名：

1. 问题描述

linux编程：c语言，创建两个线程,实现两个线程轮流数数的功能

1. 思路分析（可使用自然语言、伪代码、流程图等方式呈现）

伪代码如下：

// 定义一个全局变量count，表示当前输出的数字

int count = 0;

// 定义一个互斥锁，用于保护count变量的访问

mutex\_t mutex;

// 定义一个函数，用于输出数字

void print\_numbers(int thread\_id) {

while (count < 10) {

// 加锁，保护count变量的访问

mutex.lock();

// 输出数字并更新count变量

printf("Thread %d: %d\n", thread\_id, count);

count++;

// 解锁，释放锁资源

mutex.unlock();

}

}

int main() {

// 创建互斥锁

mutex = create\_mutex();

// 创建两个线程，分别执行print\_numbers函数

thread\_t thread1 = create\_thread(print\_numbers, 1);

thread\_t thread2 = create\_thread(print\_numbers, 2);

// 等待两个线程执行完毕

join\_thread(thread1);

join\_thread(thread2);

// 销毁互斥锁

destroy\_mutex(mutex);

return 0;

}

1. 程序实现

#include <stdio.h>

#include <pthread.h>

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

int count = 0;

void \*thread1(void \*arg) {

while (1) {

pthread\_mutex\_lock(&mutex);

while (count % 2 == 1) {

pthread\_cond\_wait(&cond, &mutex);

}

printf("Thread 1: %d\n", count++);

pthread\_cond\_signal(&cond);

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

void \*thread2(void \*arg) {

while (1) {

pthread\_mutex\_lock(&mutex);

while (count % 2 == 0) {

pthread\_cond\_wait(&cond, &mutex);

}

printf("Thread 2: %d\n", count++);

pthread\_cond\_signal(&cond);

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

int main() {

pthread\_t t1, t2;

pthread\_mutex\_init(&mutex, NULL);

pthread\_cond\_init(&cond, NULL);

pthread\_create(&t1, NULL, thread1, NULL);

pthread\_create(&t2, NULL, thread2, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

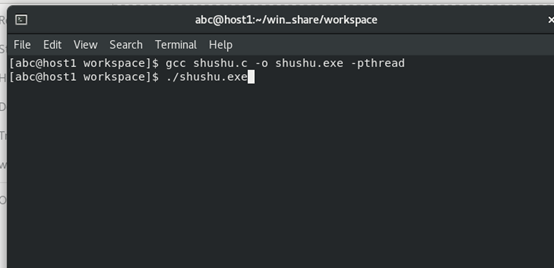
pthread\_mutex\_destroy(&mutex);

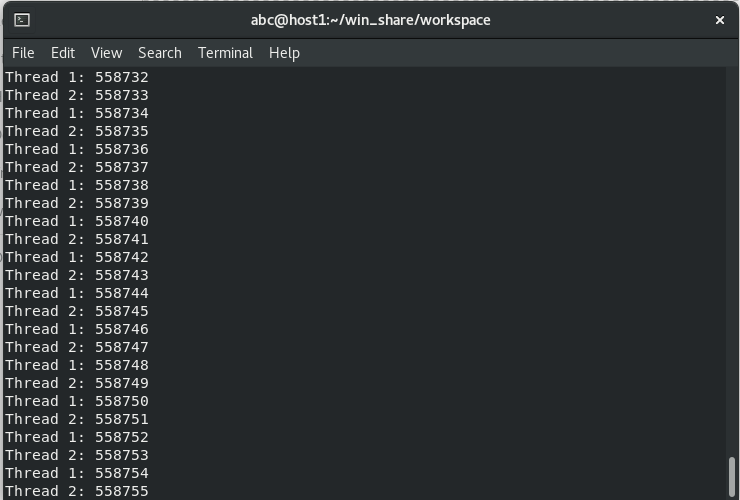
pthread\_cond\_destroy(&cond);

return 0;

}

1. 运行结果





五、反思与总结

这个程序是一个简单的并行程序，使用了共享内存模型和线程来实现两个线程轮流数数的功能。其中，使用了互斥锁和条件变量来保证线程之间的同步和互斥。

这个程序的优点是实现简单，容易理解。同时，使用了互斥锁和条件变量来保证线程之间的同步和互斥，避免了竞态条件和死锁的问题。

不足之处是程序可能存在性能瓶颈。因为两个线程在轮流数数时，每次都需要获取互斥锁和条件变量，这可能会导致线程的阻塞和唤醒，从而影响程序的性能。此外，在多核处理器上运行时，由于两个线程共享内存，可能会存在缓存一致性的问题，也会影响程序的性能。

因此，在实际的并行程序设计中，需要根据具体的应用场景来选择合适的并行模型和技术，以及优化程序的性能。