操作系统实验报告

实验一 进程、线程相关编程经验

1.3 自旋锁实验

实验步骤

步骤一： 根据实验内容要求，编写模拟自旋锁程序代码spinlock.c， 待补充主函数的示例代码如下：

```
/***spinlock.c*in xjtu*2023.8
*/
#include <stdio.h>#include <pthread.h>// 定义自旋锁结构体typedef struct {
int flag;} spinlock_t;
// 初始化自旋锁
void spinlock_init(spinlock_t *lock) {lock->flag = 0;
}
// 获取自旋锁
void spinlock_lock(spinlock_t *lock) {
```

```
while (__sync_lock_test_and_set(&lock-
>flag, 1)) {// 自旋等待
}}
// 释放自旋锁
void spinlock_unlock(spinlock_t *lock)
{__sync_lock_release(&lock->flag);
}
// 共享变量int shared_value = 0;
// 线程函数
void *thread_function(void *arg)
{spinlock_t *lock = (spinlock_t *)arg;for
(int i = 0; i < 5000; ++i) {
spinlock_lock(lock);shared_value++;spinlock
_unlock(lock);
}
return NULL;}
int main() {
pthread_t thread1, thread2;
spinlock_t lock;// 输出共享变量的值
// 初始化自旋锁
// 创建两个线程
// 等待线程结束
// 输出共享变量的值return 0;
}
```

完整代码为:

```
/***spinlock.c*in xjtu*2023.8
 */
#include <pthread.h> // 定义自旋锁结构体
#include <stdio.h>
```

```c
#include <stdlib.h>

typedef struct {
  int flag;
} spinlock_t;
// 初始化自旋锁
void spinlock_init(spinlock_t *lock) {
lock->flag = 0; }
// 获取自旋锁
void spinlock_lock(spinlock_t *lock) {
  while (__sync_lock_test_and_set(&lock-
>flag, 1)) { // 自旋等待
  }
}
// 释放自旋锁
void spinlock_unlock(spinlock_t *lock) {
__sync_lock_release(&lock->flag); }
// 共享变量
int shared_value = 0;
// 线程函数
void *thread_function(void *arg) {
  spinlock_t *lock = (spinlock_t *)arg;
  for (int i = 0; i < 5000; ++i) {
    spinlock_lock(lock);
    shared_value++;
    spinlock_unlock(lock);
  }
  return NULL;
}
int main() {
```

```
    pthread_t thread1, thread2;
    spinlock_t lock; // 输出共享变量的值
    spinlock_init(&lock);

    if (pthread_create(&thread1, NULL,
thread_function, &lock) != 0) {
        perror("Failed to create thread1");
        exit(1);
    }
    if (pthread_create(&thread2, NULL,
thread_function, &lock) != 0) {
        perror("Failed to create thread2");
        exit(1);
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("shared value is %d\n",
shared_value);

    return 0;
}
```

补充完成代码后，编译并运行程序，分析运行结果

```
[root@kp-test01 lab1]# gcc 1-3-1.c -o 1-3 -lpthread
[root@kp-test01 lab1]# ./1-3
shared value is 10000
[root@kp-test01 lab1]# ./1-3
shared value is 10000
[root@kp-test01 lab1]# ./1-3
shared value is 10000
[root@kp-test01 lab1]#
```

自旋锁设定成功，每个线程都正确地对shared_value进行了修改