On this page

实验3 多线程编程

实验内容与任务

- 使用pthread库创建多线程程序
- 学习与使用互斥锁mutex实现共享信息的互斥访问
- 学习与使用信号量semaphore实现共享信息的互斥访问

相关知识及背景

1. 线程是进程执行的分支，进程可以由多个并行执行的线程组成。多进程可以提高进程执行的效率，将进程的计算任务划分为 多个可并行执行的分支后，进程在计算机系统中执行时，将会分得更多的CPU时间，从而可以提高进程的执行速度。

2. 多线程进程地址空间中，各个线程共享进程的堆空间，即各个线程可以直接访问进程的全局变量；多线程地址空间中，各个线程拥有 自己的栈空间，各个线程执行时的局部变量和函数调用关系保存在各自的栈空间上，不会互相影响或干扰。

3. 多线程共享的堆空间为线程间传递和交换信息提供了便利，所以无需复杂的操作，同一进程的多个线程就就可以利用进程全局变量实现通信。

4. 多线程使用全局变量进行数据交换或通信时，需要使用互斥锁或信号量来保持数据的互斥访问。

实验要求及过程

实验要求

1. 能够理解并使用ptrhead_create函数创建多线程。

2. 能够使用pthread_mutex实现互斥访问全局变量。

3. 能够使用信号量实现互斥访问共享变量。

实验过程

示例1. 创建多线程

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 2

/* 线程函数参数结构体 */
typedef struct _thread_data_t
{
    int tid;
    double stuff;
} thread_data_t;

/* 线程函数 */
void *thr_func(void *arg)
{
    thread_data_t *data = (thread_data_t *)arg;
    printf("hello from thr_func, thread id: %d\n", data->tid);
    pthread_exit(NULL);
}

int main(int argc, char **argv)
{
    pthread_t thr[NUM_THREADS];
    int i, rc;

    /* 线程参数数组 */
    thread_data_t thr_data[NUM_THREADS];

    /* 创建多线程 */
    for (i = 0; i < NUM_THREADS; ++i)
    {
        thr_data[i].tid = i;
        if ((rc = pthread_create( &thr[i], NULL, thr_func, &thr_data[i])))
         {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return EXIT_FAILURE;
        }
    }

    /* 阻塞等待线程执行结束 */
    for (i = 0; i < NUM_THREADS; ++i)
```

```
    {
        pthread_join(thr[i], NULL);
    }
    return EXIT_SUCCESS;
}
```

示例2. 使用互斥锁实现互斥访问全局变量

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 5

/* 线程函数参数结构体 */
typedef struct _thread_data_t
{
  int tid;
  double stuff;
} thread_data_t;

/* 共享变量 */
double shared_x;
pthread_mutex_t lock_x;

void *thr_func(void *arg)
{
  thread_data_t *data = (thread_data_t *)arg;

  printf("hello from thr_func, thread id: %d\n", data->tid);

  /* 修改共享变量前先上锁 */
  pthread_mutex_lock(&lock_x);
    shared_x += data->stuff;
    printf("x = %f\n", shared_x);
  pthread_mutex_unlock(&lock_x);

  pthread_exit(NULL);
}

int main(int argc, char **argv)
{
  pthread_t thr[NUM_THREADS];
  int i, rc;
  /* 线程函数参数数组 */
```

```
  thread_data_t thr_data[NUM_THREADS];

  /* 初始化共享变量 */
  shared_x = 0;

  /* 初始化互斥锁 */
  pthread_mutex_init(&lock_x, NULL);

  /* 创建多线程 */
  for (i = 0; i < NUM_THREADS; ++i)
  {
    thr_data[i].tid = i;
    thr_data[i].stuff = (i + 1) * NUM_THREADS;
    if ((rc = pthread_create(&thr[i], NULL, thr_func, &thr_data[i])))
    {
      fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
      return EXIT_FAILURE;
    }
  }
  /* 阻塞等待线程执行结束 */
  for (i = 0; i < NUM_THREADS; ++i)
  {
    pthread_join(thr[i], NULL);
  }

  return EXIT_SUCCESS;
}
```

示例3. 使用信号量实现互访问全局变量

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t mutex;

void* thread(void* arg)
{
    /* 进入区 */
    sem_wait(&mutex);
    printf("\nEntered..\n");

    /* 临界区 */
    sleep(4);
```

```
    /* 退出区*/
    printf("\nJust Exiting...\n");
    sem_post(&mutex);
}


int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}
```