

实验五：Linux环境多线程编程

郑海刚



HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

本讲概述

- 主要内容
 - 线程介绍
 - pthread库
 - 多线程实现DGEMM（实验内容）

线程和进程

- 多核处理器的出现，想利用多核的性能，但又不想数据隔离
- 线程更轻量：需要的资源更少，数据传递更简单，创建速度、切换速度更快
- 通常的编程模型：多进程或者多线程，不会同时多进程多线程
- 现代操作系统，调度单位是线程，同一进程下的不同线程可以并行执行在不同的CPU核上

线程 (thread) 创建

- lab5/thread/t0.c
 - 创建了两个线程
 - 相同的pid, 不同的tid
- 编译: gcc t0.c -lpthread

```
6 void *mythread(void *arg) {
7     pthread_t tid = pthread_self();
8     printf("pid:%d, tid:%u, %s\n", (int) getpid(), (unsigned int)tid, (char *) arg);
9     return NULL;
10 }
11
12 int main(int argc, char *argv[]) {
13     pthread_t p0, p1, p2;
14     int rc;
15
16     p0 = pthread_self();
17     printf("main begin, tid p0:%u\n", (unsigned int)p0);
18     rc = pthread_create(&p1, NULL, mythread, "A"); assert(rc == 0);
19     rc = pthread_create(&p2, NULL, mythread, "B"); assert(rc == 0);
20     // join waits for the threads to finish
21     printf("pthread created, tid p1:%u, p2:%u\n", (unsigned int)p1, (unsigned int)p2);
22     rc = pthread_join(p1, NULL); assert(rc == 0);
23     rc = pthread_join(p2, NULL); assert(rc == 0);
24     printf("main end\n");
25     return 0;
26 }
```

```
$ lab5/thread <master*> »gcc t0.c -lpthread
$ lab5/thread <master*> »./a.out
main begin, tid p0:3770799936
pthread created, tid p1:3770795776, p2:3762403072
pid:21724, tid:3762403072, B
pid:21724, tid:3770795776, A
main end
```

线程API: pthread_create

- 功能: 创建线程
- 参数
 - thread
 - attr
 - start_routine: 函数指针, 线程执行的功能
 - arg: 传给start_routine的参数
- 返回值: 成功返回0, 失败返回error number

NAME

pthread_create - create a new thread

SYNOPSIS

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

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,  
void *(*start_routine) (void *), void *arg);
```

Compile and link with `-pthread`.

多线程查看

- lab5/thread/t0.c
 - mythread 函数中加一行: while(1);
 - top查看进程信息
 - 多线程任务CPU利用率可以大于100%
 - 键入V开启树形显示 (forest view) , 再键入H开启分行列出子线程
- [fork\(\) vs clone\(\) vs Pthread](#)
 - fork和pthread都是调用的clone
 - 线程也有PID

pthread_create 传入多个参数

- lab5/thread/thread_create_with_return_args.c
 - 自定义结构体

```
6  typedef struct {
7      int a;
8      int b;
9  } myarg_t;
10
11  typedef struct {
12      int x;
13      int y;
14  } myret_t;
15
16  void *mythread(void *arg) {
17      myarg_t *args = (myarg_t *) arg;
18      printf("args %d %d\n", args->a, args->b);
19      myret_t *rvals = malloc(sizeof(myret_t));
20      assert(rvals != NULL);
21      rvals->x = 1;
22      rvals->y = 2;
23      return (void *) rvals;
24  }
25
26  int main(int argc, char *argv[]) {
27      pthread_t p;
28      myret_t *rvals;
29      myarg_t args = { 10, 20 };
30      Pthread_create(&p, NULL, mythread, &args);
31      Pthread_join(p, (void **) &rvals);
32      printf("returned %d %d\n", rvals->x, rvals->y);
33      free(rvals);
34      return 0;
35  }
```

线程API: pthread_join

- 功能: 等待线程结束并回收资源
- 参数
 - thread: 指定线程id
 - retval: 如果非null则获取返回结果
- 返回值: 成功返回0, 失败返回error number

NAME

pthread_join - join with a terminated thread

SYNOPSIS

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

```
int pthread_join(pthread_t thread, void **retval);
```

Compile and link with `-pthread`.

线程结合（joinable）、分离（detached）属性

- [如果不调用pthread_join](#)
 - 则会变成“僵尸线程”，每个僵尸线程都会消耗一些系统资源，当有太多的僵尸线程的时候，可能会导致创建线程失败。
- 分离线程（detached）
 - 使用pthread_detach（）设置线程为分离状态后，线程结束时资源会被系统自动回收，而不再需要进行pthread_join() 操作。

线程数据共享

- lab5/thread/t1.c
 - 两个线程操作全局变量counter互相都能看到

```
(base) $ lab5-thread/thread »./a.out 10000
main: begin [counter = 0] [55c42eb1602c]
A: begin [addr of i: 0x7fe727f71edc]
B: begin [addr of i: 0x7fe727770edc]
B: done
A: done
main: done
[counter: 20000]
[should: 20000]
```

```
11 void *mythread(void *arg) {
12     char *letter = arg;
13     int i; // stack (private per thread)
14     printf("%s: begin [addr of i: %p]\n", letter, &i);
15     for (i = 0; i < max; i++) {
16         counter = counter + 1; // shared: only one
17     }
18     printf("%s: done\n", letter);
19     return NULL;
20 }
21
22 int main(int argc, char *argv[]) {
23     if (argc != 2) {
24         fprintf(stderr, "usage: main-first <loopcount>\n");
25         exit(1);
26     }
27     max = atoi(argv[1]);
28
29     pthread_t p1, p2;
30     printf("main: begin [counter = %d] [%lx]\n", counter,
31         (long unsigned int) &counter);
32     Pthread_create(&p1, NULL, mythread, "A");
33     Pthread_create(&p2, NULL, mythread, "B");
34     // join waits for the threads to finish
35     Pthread_join(p1, NULL);
36     Pthread_join(p2, NULL);
37     printf("main: done\n [counter: %d]\n [should: %d]\n",
38         counter, max*2);
39     return 0;
40 }
```

线程同步

- 奇怪的现象
 - ./a.out 10000累加值正确
 - ./a.out 1000000累加大概率不正确
 - 线程同步的问题，单核多核都会出现

```
(base) $ lab5-thread/thread >> ./a.out 1000000
main: begin [counter = 0] [55f322acd02c]
A: begin [addr of i: 0x7f93617ededc]
B: begin [addr of i: 0x7f9360fecedc]
B: done
A: done
main: done
[counter: 1207872]
[should: 2000000]
```