3.3 线程池初始化_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 3.3 线程池初始化涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

全部开发者教程

物联网 / 嵌入式工程师

第 24 周 stm32 芯片 – 智能硬件项目实战与企业笔试

未命名节

第 25 周 大厂必备 – linux 内核与文件系统移植

未命名节

第 26 周 嵌入式开发 – 系统移植 – bootloader、yocto

未命名节

第 27 周 嵌入式底层核心技能 – Linux 设备驱动初级

未命名节

第 28 周 嵌入式底层核心技能 – Linux 设备驱动中级

未命名节

第 29 周 嵌入式底层核心技能 – Linux 设备驱动高级 1

未命名节

第 30 周 嵌入式底层核心技能 – Linux 设备驱动高级 2

未命名节

第 31 周 嵌入式人工智能必备 – Python

未命名节

第 32 周 智能家居项目实战之客户端功能开发

未命名节

第 33 周 智能家居项目实战之网关端功能开发

未命名节

第 34 周 智能家居项目实战之设备端功能开发

未命名节

第 35 周 物联网 / 嵌入式项目答辩和就业指导

未命名节

第 36 周 独立开发阶段 – 三大热门领域项目

未命名节

- 线程池的初始化主要实现的内容如下:
    - step 1: 创建线程池结构体对象, 具体如下:
    - ```
      tpool_t *pool = (tpool_t *)malloc(sizeof(tpool_t));
      if (!pool)
              goto err;
      ```
    - step 2 : 分配保存线程 ID 的空间并清零
    - ```
      pool->tp_work_thread_ids = (pthread_t *)malloc(sizeof(pthread_t) * cnt);
       if (!pool->tp_work_thread_ids)
              goto err;
        memset(pool->tp_work_thread_ids,0,sizeof(pthread_t) * cnt);
      ```
    - step 3 : 初始化线程数量
    - ```
        pool->tp_number_of_threads = cnt;
      ```
    - step 4 : 初始化互斥锁与条件变量
    - ```
      ret = pthread_mutex_init(&pool->tp_mutex_pool,NULL);
          if (ret != 0)
              goto err;

      ret = pthread_cond_init(&pool->tp_cond_empty,NULL);
          if (ret != 0)
              goto err;

      ret = pthread_cond_init(&pool->tp_cond_full,NULL);
          if (ret != 0)
              goto err;
      ```
    - step 5 : 初始化任务队列的容量
    - ```
      pool->tp_qcapacity = queuesize;
      ```
    - step 6 : 创建任务队列的空间
    - ```
       pool->tp_task = (tpool_task_t *)malloc(sizeof(tpool_task_t) * pool->tp_qcapacity)
          if (!pool->tp_task)
              goto err;
      ```
    - step 7 : 初始化任务队列的相关属性
    - ```
       pool->tp_qsize = 0;
       pool->tp_qfront = 0;
       pool->tp_qrear = 0;
      ```
    - step 8 : 初始化线程池销毁标志
    - ```
        pool->tp_shutdown = false;
      ```
    - step 9 : 定义线程执行函数
    - ```
      void *tp_worker(void *arg)
      {}
      ```
    - step 9 : 创建线程
    - ```
       for (int i = 0;i < pool->tp_number_of_threads;i++){
              ret = pthread_create(&pool->tp_work_thread_ids[i],NULL,tp_worker,pool);
              if (ret != 0){
                  fprintf(stderr,"pthread_create(): %s\n",strerror(ret));
                  goto err;
              }

          }
      ```
- 完整代码如下:

```
tpool_t *thread_pool_create(int cnt,int queuesize)
{
    int err,ret;
```

```
    tpool_t *pool = (tpool_t *)malloc(sizeof(tpool_t));
    if (!pool)
        goto err;

    pool->tp_work_thread_ids = (pthread_t *)malloc(sizeof(pthread_t) * cnt);
    if (!pool->tp_work_thread_ids)
        goto err;

    memset(pool->tp_work_thread_ids,0,sizeof(pthread_t) * cnt);

    pool->tp_number_of_threads = cnt;

    ret = pthread_mutex_init(&pool->tp_mutex_pool,NULL);
    if (ret != 0)
        goto err;

    ret = pthread_cond_init(&pool->tp_cond_empty,NULL);
    if (ret != 0)
        goto err;

    ret = pthread_cond_init(&pool->tp_cond_full,NULL);
    if (ret != 0)
        goto err;

    pool->tp_qcapacity = queuesize;

     pool->tp_task = (tpool_task_t *)malloc(sizeof(tpool_task_t) * pool->tp_qcapacity);
    if (!pool->tp_task)
        goto err;

    pool->tp_qsize = 0;
    pool->tp_qfront = 0;
    pool->tp_qrear = 0;

    pool->tp_shutdown = false;

     for (int i = 0;i < pool->tp_number_of_threads;i++){
        ret = pthread_create(&pool->tp_work_thread_ids[i],NULL,tp_worker,pool);
        if (ret != 0){
            fprintf(stderr,"pthread_create(): %s\n",strerror(ret));
            goto err;
        }

        pthread_detach(pool->tp_work_thread_ids[i]);
    }

    return pool;
err:
    if (pool && pool->tp_work_thread_ids)
        free(pool->tp_work_thread_ids);
    if (pool && pool->tp_task)
        free(pool->tp_task);
    if (pool)
        free(pool);
    return NULL;
}
```

全文完