hfm_honey学习乐园

huangfanmei.blog.chinaunix.net

知之者常乐～

hfm_honey

博客访问： 904201
博文数量： 160
博客积分： 2234
博客等级： 大尉
技术积分： 3122
用 户 组： 普通用户
注册时间： 2012-05-17 21:34

加关注 短消息
论坛 加好友

个人简介

未来很长。

文章分类

线程池的概念及Linux 怎么设计一个简单的线程池

2012-09-03 22:08:09

分类： LINUX

今天看到一篇hao文章，现在跟大家分享一下，但是必须得自己去学会实践！

什么是线程池？

简单点说，线程池就是有一堆已经创建好了的线程，初始它们都处于空闲等待状态，当有新的任务需要处理的时候，就从这个池子里面取一个空闲等 待的线程来处理该任务，当处理完成了就再次把该线程放回池中，以供后面的任务使用。当池子里的线程全都处理忙碌状态时，线程池中没有可用的空闲等待线程，此时，根据需要选择创建一个新的线程并置入池中，或者通知任务线程池忙，稍后再试。
为什么要用线程池？

为什么要用线程池？

我们说，线程的创建和销毁比之进程的创建和销毁是轻量级的，但是当我们的任务需要大量进行大量线程的创建和销毁操作时，这个消耗就会变成的相当大。比如， 当你设计一个压力性能测试框架的时候，需要连续产生大量的并发操作，这个是时候，线程池就可以很好的帮上你的忙。线程池的好处就在于线程复用，一个任务处理完成后，当前线程可以直接处理下一个任务，而不是销毁后再创建，非常适用于连续产生大量并发任务的场合。

线程池工作原理

线程池中每一个线程的工作过程如下：

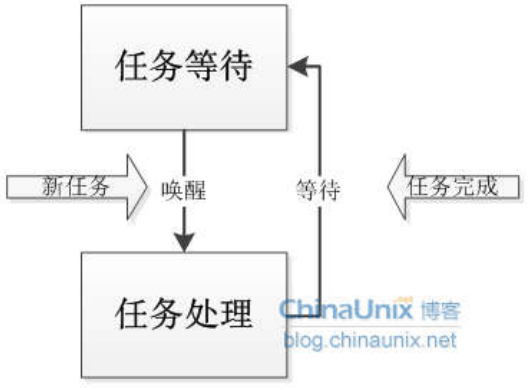


图 1： 线程的工作流程

线程池的任务就在于负责这些线程的创建，销毁和任务处理参数

传递、唤醒和等待。

1. 创建若干线程，置入线程池

查找算法（2）
排序算法（3）
Linux系统编程（16）
未分配的博文（14）

文章存档
2016年（4）
2015年（1）
2014年（6）
2013年（37）
2012年（112）

我的朋友
zhmfairy 绿水方舟 meekwolf
yim_drea sksunhum 星语心愿
kimirse dallygar hsummert

最近访客
HazeC kaka625 qujia_19
呵呵里 CurryTma julien_2
halou90 Jeremy_W wintix

微信关注
IT168企业级官微
微信号：IT168qiye
系统架构师大会
微信号：SACC2013

订阅

推荐博文
• Nginx+Keepalived实现双机热...
• mysql字符集乱码问题
• 统计nginx日志中各服务（目录...
• 切换网站主域名经验总结...
• mysql core文件的正确打开姿...

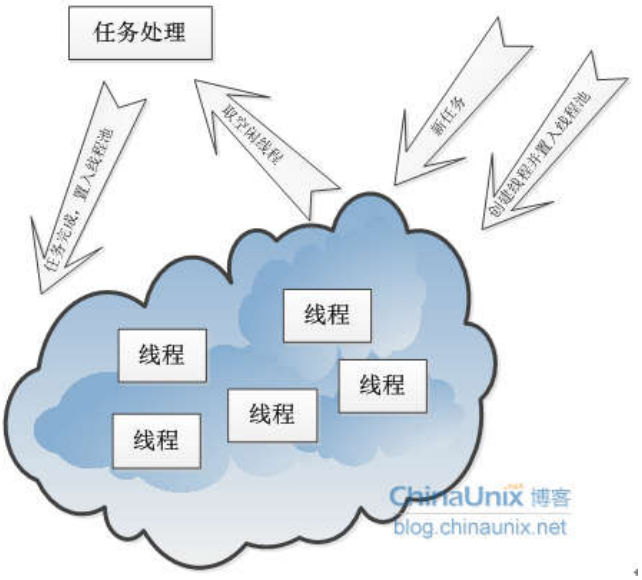
2. 任务达到时，从线程池取空闲线程

3. 取得了空闲线程，立即进行任务处理

4. 否则新建一个线程，并置入线程池，执行3

5. 如果创建失败或者线程池已满，根据设计策略选择返回错误或将任务置入处理队列，等待处理

6. 销毁线程池



线程池设计

数据结构设计

任务设计

[cpp] view plaincopy

```
typedef struct tp_work_desc_s TpWorkDesc;
typedef void (*process_job)(TpWorkDesc*job);
struct tp_work_desc_s {
        void *ret; //call in, that is arguments
        void *arg; //call out, that is return value
};
```

其中，TpWorkDesc是任务参数描述，arg是传递给任务的参数，ret则是任务处理完成后的返回值；
process_job函数是任务处理函数原型，每个任务处理函数都应该这样定义，然后将它作为参数传给线程池处理，线程池将会选择一个空闲线程通过调用该函数来进行任务处理；

线程设计

[cpp] view plaincopy

```
typedef struct tp_thread_info_s TpThreadInfo;
struct tp_thread_info_s {
        pthread_t thread_id; //thread id num
        TPBOOL is_busy; //thread status:true-busy;flase-idle
        pthread_cond_t thread_cond;
        pthread_mutex_t thread_lock;
        process_job proc_fun;
        TpWorkDesc* th_job;
        TpThreadPool* tp_pool;
};
```

TpThreadInfo是对一个线程的描述。
thread_id是该线程的ID;

is_busy用于标识该线程是否正处理忙碌状态;

thread_cond用于任务处理时的唤醒和等待;

thread_lock，用于任务加锁，用于条件变量等待加锁;

proc_fun是当前任务的回调函数地址;

th_job是任务的参数信息;

tp_pool是所在线程池的指针;

线程池设计

[cpp] view plaincopy

```
typedef struct tp_thread_pool_s TpThreadPool;
struct tp_thread_pool_s {
        unsigned min_th_num; //min thread number in the pool
        unsigned cur_th_num; //current thread number in the pool
        unsigned max_th_num; //max thread number in the pool
        pthread_mutex_t tp_lock;
        pthread_t manage_thread_id; //manage thread id num
        TpThreadInfo* thread_info;
        Queue idle_q;
        TPBOOL stop_flag;
};
```

TpThreadPool是对线程池的描述。
min_th_num是线程池中至少存在的线程数，线程池初始化的过程中会创建min_th_num数量的线程;

cur_th_num是线程池当前存在的线程数量;

max_th_num则是线程池最多可以存在的线程数量;

tp_lock用于线程池管理时的互斥;

manage_thread_id是线程池的管理线程ID;

thread_info则是指向线程池数据，这里使用一个数组来存储线程池中线程的信息，该数组的大小为max_th_num;

idle_q是存储线程池空闲线程指针的队列，用于从线程池快速取得空闲线程;

stop_flag用于线程池的销毁，当stop_flag为FALSE时，表明当前线程池需要销毁，所有忙碌线程在处理完当前任务后会退出；

算法设计

线程池的创建和初始化

线程创建

创建伊始，线程池线程容量大小上限为max_th_num，初始容量为min_th_num；

```
[cpp] view plaincopy
TpThreadPool *tp_create(unsigned min_num, unsigned max_num) {
	TpThreadPool *pTp;
	pTp = (TpThreadPool*) malloc(sizeof(TpThreadPool));

	memset(pTp, 0, sizeof(TpThreadPool));

	//init member var
	pTp->min_th_num = min_num;
	pTp->cur_th_num = min_num;
	pTp->max_th_num = max_num;
	pthread_mutex_init(&pTp->tp_lock, NULL);

	//malloc mem for num thread info struct
	if (NULL != pTp->thread_info)
		free(pTp->thread_info);
	pTp->thread_info = (TpThreadInfo*) malloc(sizeof(TpThreadInfo) * pTp->max_th_num);
	memset(pTp->thread_info, 0, sizeof(TpThreadInfo) * pTp->max_th_num);

	return pTp;
}
```

线程初始化

```
[cpp] view plaincopy
TPBOOL tp_init(TpThreadPool *pTp) {
	int i;
	int err;
	TpThreadInfo *pThi;

	initQueue(&pTp->idle_q);
	pTp->stop_flag = FALSE;

	//create work thread and init work thread info
	for (i = 0; i < pTp->min_th_num; i++) {
		pThi = pTp->thread_info +i;
		pThi->tp_pool = pTp;
		pThi->is_busy = FALSE;
		pthread_cond_init(&pThi->thread_cond, NULL);
		pthread_mutex_init(&pThi->thread_lock, NULL);
		pThi->proc_fun = def_proc_fun;
		pThi->th_job = NULL;
		enQueue(&pTp->idle_q, pThi);
```

```
            err = pthread_create(&pThi->thread_id, NULL, tp_work_thread, pThi);
            if (0 != err) {
                  perror("tp_init: create work thread failed.");
                  clearQueue(&pTp->idle_q);
                  return FALSE;
            }
      }

      //create manage thread
      err = pthread_create(&pTp->manage_thread_id, NULL, tp_manage_thread, pTp);
      if (0 != err) {
            clearQueue(&pTp->idle_q);
            printf("tp_init: creat manage thread failed\n");
            return FALSE;
      }

      return TRUE;
}
```

初始线程池中线程数量为min_th_num，对这些线程一一进行初始化；
将这些初始化的空闲线程一一置入空闲队列；

创建管理线程，用于监控线程池的状态，并适当回收多余的线程资源；

线程池的关闭和销毁

[cpp] view plaincopy

```
void tp_close(TpThreadPool *pTp, TPBOOL wait) {
      unsigned i;

      pTp->stop_flag = TRUE;
      if (wait) {
            for (i = 0; i < pTp->cur_th_num; i++) {
                  pthread_cond_signal(&pTp->thread_info[i].thread_cond);
            }
            for (i = 0; i < pTp->cur_th_num; i++) {
                  pthread_join(pTp->thread_info[i].thread_id, NULL);
                  pthread_mutex_destroy(&pTp->thread_info[i].thread_lock);
                  pthread_cond_destroy(&pTp->thread_info[i].thread_cond);
            }
      } else {
            //close work thread
            for (i = 0; i < pTp->cur_th_num; i++) {
                  kill((pid_t)pTp->thread_info[i].thread_id, SIGKILL);
                  pthread_mutex_destroy(&pTp->thread_info[i].thread_lock);
                  pthread_cond_destroy(&pTp->thread_info[i].thread_cond);
            }
      }
      //close manage thread
      kill((pid_t)pTp->manage_thread_id, SIGKILL);
      pthread_mutex_destroy(&pTp->tp_lock);

      //free thread struct
      free(pTp->thread_info);
```

```
	pTp->thread_info = NULL;
}
```

线程池关闭的过程中，可以选择是否对正在处理的任务进行等待，如果是，则会唤醒所有任务，然后等待所有任务执行完成，然后返回；如果不是，则将立即杀死所有线程，然后返回，注意：这可能会导致任务的处理中断而产生错误！

任务处理

[cpp] view plaincopy

```
TPBOOL tp_process_job(TpThreadPool *pTp, process_job proc_fun, TpWorkDesc *job) {
	TpThreadInfo *pThi ;
	//fill pTp->thread_info's relative work key
	pthread_mutex_lock(&pTp->tp_lock);
	pThi = (TpThreadInfo *) deQueue(&pTp->idle_q);
	pthread_mutex_unlock(&pTp->tp_lock);
	if(pThi){
		pThi->is_busy =TRUE;
		pThi->proc_fun = proc_fun;
		pThi->th_job = job;
		pthread_cond_signal(&pThi->thread_cond);
		DEBUG("Fetch a thread from pool.\n");
		return TRUE;
	}
	//if all current thread are busy, new thread is created here
	pthread_mutex_lock(&pTp->tp_lock);
	pThi = tp_add_thread(pTp);
	pthread_mutex_unlock(&pTp->tp_lock);

	if(!pThi){
		DEBUG("The thread pool is full, no more thread available.\n");
		return FALSE;
	}
	DEBUG("No more idle thread, created a new one.\n");
	pThi->proc_fun = proc_fun;
	pThi->th_job = job;

	//send cond to work thread
	pthread_cond_signal(&pThi->thread_cond);
	return TRUE;
}
```

当一个新任务到达是，线程池首先会检查是否有可用的空闲线程，如果是，则采用才空闲线程进行任务处理并返回TRUE，如果不是，则尝试新建一个线程，并使用该线程对任务进行处理，如果失败则返回FALSE，说明线程池忙碌或者出错。

[cpp] view plaincopy

```
static void *tp_work_thread(void *arg) {
	pthread_t curid;//current thread id
	TpThreadInfo *pTinfo = (TpThreadInfo *) arg;

	//wait cond for processing real job.
	while (!(pTinfo->tp_pool->stop_flag)) {
		pthread_mutex_lock(&pTinfo->thread_lock);
		pthread_cond_wait(&pTinfo->thread_cond, &pTinfo->thread_lock);
		pthread_mutex_unlock(&pTinfo->thread_lock);

		//process
		pTinfo->proc_fun(pTinfo->th_job);
```

```
            //thread state be set idle after work
            //pthread_mutex_lock(&pTinfo->thread_lock);
            pTinfo->is_busy = FALSE;
            enQueue(&pTinfo->tp_pool->idle_q, pTinfo);
            //pthread_mutex_unlock(&pTinfo->thread_lock);
            DEBUG("Job done, I am idle now.\n");
      }
}
```

上面这个函数是任务处理函数，该函数将始终处理等待唤醒状态，直到新任务到达或者线程销毁时被唤醒，然后调用任务处理回调函数对任务进行处理；当任务处理完成时，则将自己置入空闲队列中，以供下一个任务处理。

[cpp] view plaincopy

```
TpThreadInfo *tp_add_thread(TpThreadPool *pTp) {
      int err;
      TpThreadInfo *new_thread;

      if (pTp->max_th_num <= pTp->cur_th_num)
            return NULL;

      //malloc new thread info struct
      new_thread = pTp->thread_info + pTp->cur_th_num;

      new_thread->tp_pool = pTp;
      //init new thread's cond & mutex
      pthread_cond_init(&new_thread->thread_cond, NULL);
      pthread_mutex_init(&new_thread->thread_lock, NULL);

      //init status is busy, only new process job will call this function
      new_thread->is_busy = TRUE;
      err = pthread_create(&new_thread->thread_id, NULL, tp_work_thread, new_thread);
      if (0 != err) {
            free(new_thread);
            return NULL;
      }
      //add current thread number in the pool.
      pTp->cur_th_num++;

      return new_thread;
}
```

上面这个函数用于向线程池中添加新的线程，该函数将会在当线程池没有空闲线程可用时被调用。
函数将会新建一个线程，并设置自己的状态为busy（立即就要被用于执行任务）。

线程池管理
线程池的管理主要是监控线程池的整体忙碌状态，当线程池大部分线程处于空闲状态时，管理线程将适当的销毁一定数量的空闲线程，以便减少线程池对系统资源的消耗。

这里设计认为，当空闲线程的数量超过线程池线程数量的1/2时，线程池总体处理空闲状态，可以适当销毁部分线程池的线程，以减少线程池对系统资源的开销。

线程池状态计算

这里的BUSY_THRESHOLD的值是0.5，也即是当空闲线程数量超过一半时，返回0，说明线程池整体状态为闲，否则返回1，说明为忙。

```
[cpp] view plaincopy
int tp_get_tp_status(TpThreadPool *pTp) {
      float busy_num = 0.0;
      int i;

      //get busy thread number
      busy_num = pTp->cur_th_num - pTp->idle_q.count;

      DEBUG("Current thread pool status, current num: %u, busy num: %u, idle num: %u\n", pTp->cur_th_num,
(unsigned)busy_num, pTp->idle_q.count);
      //0.2? or other num?
      if (busy_num / (pTp->cur_th_num) < BUSY_THRESHOLD)
            return 0;//idle status
      else
            return 1;//busy or normal status
}
```

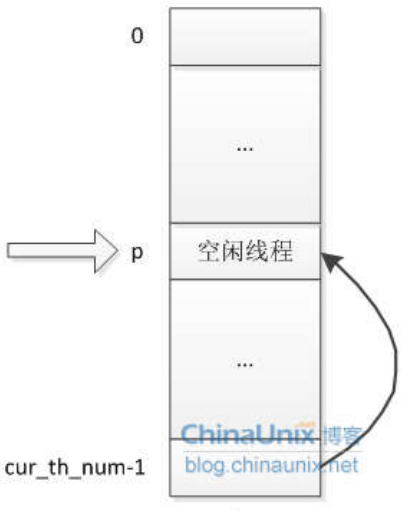
线程的销毁算法

1. 从空闲队列中dequeue一个空闲线程指针，该指针指向线程信息数组的某项，例如这里是p；

2. 销毁该线程

3. 把线程信息数组的最后一项拷贝至位置p

4. 线程池数量减少一，即cur_th_num—




```
[cpp] view plaincopy
TPBOOL tp_delete_thread(TpThreadPool *pTp) {
      unsigned idx;
      TpThreadInfo *pThi;
      TpThreadInfo tT;

      //current thread num can't < min thread num
      if (pTp->cur_th_num <= pTp->min_th_num)
            return FALSE;
      //pthread_mutex_lock(&pTp->tp_lock);
      pThi = deQueue(&pTp->idle_q);
      //pthread_mutex_unlock(&pTp->tp_lock);
```

```
	if(!pThi)
		return FALSE;

	//after deleting idle thread, current thread num -1
	pTp->cur_th_num--;
	memcpy(&tT, pThi, sizeof(TpThreadInfo));
	memcpy(pThi, pTp->thread_info + pTp->cur_th_num, sizeof(TpThreadInfo));

	//kill the idle thread and free info struct
	kill((pid_t)tT.thread_id, SIGKILL);
	pthread_mutex_destroy(&tT.thread_lock);
	pthread_cond_destroy(&tT.thread_cond);

	return TRUE;
}
```

线程池监控

线程池通过一个管理线程来进行监控，管理线程将会每隔一段时间对线程池的状态进行计算，根据线程池的状态适当的销毁部分线程，减少对系统资源的消耗。

```
[cpp] view plaincopy
static void *tp_manage_thread(void *arg) {
	TpThreadPool *pTp = (TpThreadPool*) arg;//main thread pool struct instance

	//1?
	sleep(MANAGE_INTERVAL);

	do {
		if (tp_get_tp_status(pTp) == 0) {
			do {
				if (!tp_delete_thread(pTp))
					break;
			} while (TRUE);
		}//end for if

		//1?
		sleep(MANAGE_INTERVAL);
	} while (!pTp->stop_flag);
	return NULL;
}
```

程序测试

至此，我们的设计需要使用一个测试程序来进行验证。于是，我们写下这样一段代码。

```
[cpp] view plaincopy
#include
#include
#include "thread_pool.h"

#define THD_NUM 10
void proc_fun(TpWorkDesc *job){
	int i;
	int idx=*(int *)job->arg;
```

```
        printf("Begin: thread %d\n", idx);
        sleep(3);
        printf("End:    thread %d\n", idx);
}

int main(int argc, char **argv){
        TpThreadPool *pTp= tp_create(5,10);
        TpWorkDesc pWd[THD_NUM];
        int i, *idx;

        tp_init(pTp);
        for(i=0; i < THD_NUM; i++){
                idx=(int *) malloc(sizeof(int));
                *idx=i;
                pWd[i].arg=idx;
                tp_process_job(pTp, proc_fun, pWd+i);
                usleep(400000);
        }
        //sleep(1);
        tp_close(pTp, TRUE);
        free(pTp);
        printf("All jobs done!\n");
        return 0;
}
```

源码下载地址：https://sourceforge.net/projects/thd-pool-linux/

上一篇：生活中常用的成语
下一篇：深入理解数组与指针的区别

0

相关热门文章

linux 常见服务端口
linux dhcp peizhi roc
xmanager 2.0 for linux配置
关于Unix文件的软链接
【ROOTFS搭建】busybox的httpd...
求教这个命令什么意思，我是新...
openwrt中luci学习笔记
sed -e "/grep/d" 是什么意思...
Linux里如何查找文件内容...
谁能够帮我解决LINUX 2.6 10...

给主人留下些什么吧！~~

lmnos 2012-10-23 17:43:37

拜读了，不错

回复 | 举报

评论热议

请登录后评论。
登录 注册