

OS 课设4

信息科学与工程学院

2019011777 计算机 19-3 刘康来

实验 4 Web 服务器的线程池模型

代码见后

- 跑多了，还是有 pipe 错误

调整个数

- 观察 3 个代表：

```
> ../http_load -p 75 -s 10 ../http
65484 fetches, 75 max parallel, 1.827e+07 bytes, in 10 seconds
279 mean bytes/connection
6548.39 fetches/sec, 1.827e+06 bytes/sec
msecs/connect: 0.142869 mean, 1009.16 max, 0.036 min
msecs/first-response: 11.2898 mean, 44.595 max, 1.547 min
HTTP response codes:
  code 200 -- 65484
```

```
> ../http_load -p 80 -s 10 ../http
71606 fetches, 80 max parallel, 1.99781e+07 bytes, in 10 seconds
279 mean bytes/connection
7160.6 fetches/sec, 1.99781e+06 bytes/sec
msecs/connect: 0.177508 mean, 1029.44 max, 0.036 min
msecs/first-response: 10.9747 mean, 70.971 max, 1.439 min
HTTP response codes:
  code 200 -- 71606
```

```
> ../http_load -p 90 -s 10 ../http
68963 fetches, 90 max parallel, 1.92407e+07 bytes, in 10 seconds
279 mean bytes/connection
6896.29 fetches/sec, 1.92406e+06 bytes/sec
msecs/connect: 0.243595 mean, 1015.89 max, 0.037 min
msecs/first-response: 12.7873 mean, 48.84 max, 1.135 min
HTTP response codes:
  code 200 -- 68963
```

当线程数 < 请求客户端数时，获取大体不会增加，在我电脑上，估计 80 最优了。

比较

- 30s 的对比（各自的最优）：
- 线程池(开 10 个客户端一样)

```
> ../http_load -p 80 -s 30 ../http
211538 fetches, 80 max parallel, 5.90191e+07 bytes, in 30 seconds
279 mean bytes/connection
7051.27 fetches/sec, 1.9673e+06 bytes/sec
msecs/connect: 0.104847 mean, 1024.52 max, 0.042 min
msecs/first-response: 11.2318 mean, 55.047 max, 1.817 min
HTTP response codes:
  code 200 -- 211538
```

```

8
7 :211539
6 INFO: time:共用 31990.868000ms 成功处理 211539 个客户端请求,其中
5 平均每个客户端完成请求处理时间为 0.070682ms
4 平均每个客户端完成读 socket 时间为 0.002535ms
3 平均每个客户端完成写 socket 时间为 0.010530ms
2 平均每个客户端完成读网页数据时间为 0.010530ms
1 平均每个客户端完成写日志数据时间为 0.008722ms
081 :211539

```

- 多线程

```

) ../http_load -p 10 -s 30 ../http
425946 fetches, 10 max parallel, 1.18839e+08 bytes, in 30 seconds
279 mean bytes/connection
14198.2 fetches/sec, 3.96129e+06 bytes/sec
msecs/connect: 0.156725 mean, 17.064 max, 0.021 min
msecs/first-response: 0.359118 mean, 65.95 max, 0.076 min
HTTP response codes:
code 200 -- 425946

```

```

:425951
INFO: time:共用 32075.803000ms 成功处理 425951 个客户端请求,其中
平均每个客户端完成请求处理时间为 0.089118ms
平均每个客户端完成读 socket 时间为 0.020777ms
平均每个客户端完成写 socket 时间为 0.009900ms
平均每个客户端完成读网页数据时间为 0.009900ms
平均每个客户端完成写日志数据时间为 0.010336ms
:425951

```

- 多线程好，可能是获取页面太少？，试了一下 300s：
- 线程池

```

) ../http_load -p 80 -s 300 ../http
1973901 fetches, 80 max parallel, 5.50718e+08 bytes, in 300 seconds
279 mean bytes/connection
6579.67 fetches/sec, 1.83573e+06 bytes/sec
msecs/connect: 0.0675552 mean, 1004.35 max, 0.043 min
msecs/first-response: 12.0876 mean, 77.036 max, 0.695 min
HTTP response codes:
code 200 -- 1973901

```

- 多线程

```

) ../http_load -p 10 -s 300 ../http
http://127.0.0.1:8000/index.html: byte count wrong
4260004 fetches, 10 max parallel, 1.18854e+09 bytes, in 300.001 seconds
279 mean bytes/connection
14200 fetches/sec, 3.96179e+06 bytes/sec
msecs/connect: 0.156746 mean, 20.076 max, 0.022 min
msecs/first-response: 0.35981 mean, 64.176 max, 0.048 min
1 bad byte counts
HTTP response codes:
code 200 -- 4260003

```

多线程还是比线程池获取的多。。。

- 对比时间，可以看出线程池平均客户端处理时间是要更优的，
- 至于获取的页面，大概是加锁的时间限制了？

- 就简单分析吧，理论就是线程池少了线程创建销毁的时间，切换更迅速。
- 可能还是我的线程池写的不是很好？，分析没啥意义。

小计

- 端口有一点缓冲时间.
- 了解了关于结构体（内含指针）的 malloc 与 free
- 学习了 cond 与 mutex 的搭配
- 大体会了一点 thread 的 gdb debug，但感觉用处不大。
- segmentation 错误，真难找。。。.
- segmentation fault (core dumped)
- "thread" received signal SIGSEGV, Segmentation fault.

signal: spurious wakeup? <https://www.showdoc.com.cn/lizhicheng/1229635626051961>;
多线程中的 signal 发多个；惊群效应? <https://www.cnblogs.com/cthon/p/9084735.html>

关于原子性思考: <https://zhuanlan.zhihu.com/p/55123862>

Code:

threadPool.h:

```
#include <errno.h> // error
#include <pthread.h>
#include <stdbool.h> // use bool
#include <stdio.h>
#include <stdlib.h> // malloc
#include <sys/prctl.h> //prctl

/* queue status and conditional variable*/
typedef struct staconv {
    pthread_mutex_t mutex;
    pthread_cond_t cond; /*用于阻塞和唤醒线程池中线程*/
    int status;
    /*表示任务队列状态:false 为无任务,true 为有任务*/ // 那 len 用来干嘛呢?
} staconv;

typedef struct task { // 怎样添加 web ?
    struct task *next; /* 指向下一任务 */
    // void (*function)(void *arg);
    // 只是个函数声明，不是对类型的定义，感谢强大的代码报错工具 (clang ale)
    // `https://stackoverflow.com/questions/21708566/void-functionvoid-argument-how-to-return-the-function-results`
    // void* vfunc(void *); is a function declaration, not a pointer object
    // definition. You probably want void *(*vfunc)(void *)
    void *(*function)(void *arg); // 函数指针
    void *arg;
} task;

typedef struct taskqueue {
    pthread_mutex_t mutex; /* 用于互斥读写任务队列 */
    task *front; // 指向队首
    task *rear; // 指向队尾
    staconv *has_jobs; // 根据状态,阻塞线程
    int len; // 队列中任务个数
} taskqueue;

typedef struct thread {
```

```

    int id;
    pthread_t pthread;
    struct threadpool *pool;
} thread;

// volatile 变量, 变化?
typedef struct threadpool {
    thread **threads;           // 线程指针数组
    volatile int num_threads;   /* 线程池中线程数量 */
    volatile int num_working;   /* 目前正在工作的线程个数 */
    pthread_mutex_t thcount_lock; /* 线程池锁用于修改上面两个变量 */
    pthread_cond_t threads_all_idle; /* 用于销毁线程的条件变量 */
    taskqueue queue;           /* 任务队列 */
    // 这玩意不定义指针, 真好, 学到了
    volatile bool is_alive;
    /* 表示线程池是否还存活 */ // 有用吗? destoryThreadPool
                                // 时不全干掉了, 防止其他导致线程的存活问题?
} threadpool;

void init_taskqueue(taskqueue *poolQueue) {
    // poolQueue = (taskqueue *)malloc(sizeof(taskqueue));
    // 这的问题? ? ? ? 为什么这一行去掉后才对.....那是个结构体, 不是指针。。。
    pthread_mutex_init(&(poolQueue->mutex), NULL);
    poolQueue->front = NULL;
    poolQueue->rear = NULL;

    poolQueue->has_jobs = (struct staconv *)malloc(sizeof(struct staconv));
    pthread_mutex_init(&(poolQueue->has_jobs->mutex), NULL);
    pthread_cond_init(&(poolQueue->has_jobs->cond), NULL);
    poolQueue->has_jobs->status = false;

    poolQueue->len = 0;
}

void destory_taskqueue(taskqueue *poolQueue) {
    pthread_mutex_destroy(&poolQueue->mutex);
    free(poolQueue->front);
    free(poolQueue->rear);
    pthread_mutex_destroy(&poolQueue->has_jobs->mutex);
    pthread_cond_destroy(&poolQueue->has_jobs->cond);
    free(poolQueue->has_jobs);
    // free(poolQueue);
}

void push_taskqueue(taskqueue *poolQueue, task *curtask) {
    pthread_mutex_lock(&poolQueue->mutex); // lock!
    if (poolQueue->front == NULL) {        // 分空队和非空讨论
        poolQueue->front = curtask;
    } else {
        poolQueue->rear->next = curtask;
    }
    poolQueue->rear = curtask;
    poolQueue->len++;
    pthread_mutex_unlock(&poolQueue->mutex);
}

task *take_taskqueue(
    taskqueue *poolQueue) { // take_taskqueue
    // 从任务队列头部提取任务, 并在队列中删除此任务

```

```

task *tem_task;
// pthread_mutex_lock(&poolQueue->mutex); // lock!
if (poolQueue->front == NULL) //
    return NULL;
pthread_mutex_lock(&poolQueue->mutex); // lock!
tem_task = poolQueue->front;          // 可返回 NULL
poolQueue->front = poolQueue->front->next;
tem_task->next = NULL; //
// poolQueue->len--;
pthread_mutex_unlock(&poolQueue->mutex);
return tem_task;
}

/*线程运行的逻辑函数*/
void *thread_do(void *tem_pthread) {
    thread *pthread = (thread *)tem_pthread;
    /* 设置线程名字 */
    char thread_name[128] = {0};
    sprintf(thread_name, "thread-pool-%d", pthread->id);
    prctl(PR_SET_NAME, thread_name); // 重命名进程

    threadpool *pool = pthread->pool; /* 获得线程池*/
    /* 在线程池初始化时,用于已经创建线程的计数,执行 pool->num_threads++ */
    /*.....*/
    pthread_mutex_lock(&pool->thcount_lock);
    pool->num_threads++;
    pthread_mutex_unlock(&pool->thcount_lock);

    /*线程一直循环往复运行,直到 pool->is_alive 变为 false*/
    while (pool->is_alive) {
        /*如果任务队列中还要任务,则继续运行,否则阻塞*/
        /*..... */
        pthread_mutex_lock(&(pool->queue.has_jobs->mutex));
        // while (!pool->queue.has_jobs->status) { // vs queue.len ?
        while (!pool->queue.len) { // 一个 signal 放多个线程。。。那么多个
            // take_taskqueue, 里面空指针。。。而且时间大大延长
            pthread_cond_wait(&pool->queue.has_jobs->cond,
                              &pool->queue.has_jobs->mutex);
        }
        pool->queue.len--; // 保证一次只要一个线程下来,还要保证所有任务都被完成
        // pool->queue.has_jobs->status = false;
        // 这个不行,任务数大于线程数时,有些任务不会 signal.
        pthread_mutex_unlock(&(pool->queue.has_jobs->mutex));
        // printf("len:%d,id:%d\n", pool->queue.len, pthread->id);

        if (pool->is_alive) {
            /*执行到此位置,表明线程在工作,需要对工作线程数量进行计数*/
            /*.....*/
            pthread_mutex_lock(&pool->thcount_lock);
            pool->num_working++;
            pthread_mutex_unlock(&pool->thcount_lock);
            // printf("working,id:%d,num_working:%d\n", pthread->id,
            // pool->num_working);

            /* 从任务队列的队首提取任务,并执行*/
            void *(*func)(void *);
            void *arg;
            // take_taskqueue 从任务队列头部提取任务,并在队列中删除此任务

```

```

    /****需实现 take_taskqueue***/
    task *curtask = take_taskqueue(&pool->queue);
    if (curtask) { // 有非空的判断
        func = curtask->function;
        arg = curtask->arg;
        //执行任务
        func(arg);
        //释放任务
        free(curtask);
    }
    /*执行到此位置,表明线程已经将任务执行完成,需更改工作线程数量*/
    //此处还请注意,当工作线程数量为 0,表示任务全部完成,要让阻塞在
    // waitThreadPool 函数上的线程继续运行
    /*.....*/
    pthread_mutex_lock(&pool->thcount_lock);
    pool->num_working--;
    // if (pool->num_threads == 0) {
    // pthread_cond_signal(&pool->threads_all_idle);
    //}
    pthread_mutex_unlock(&pool->thcount_lock);
}
}
/*运行到此位置表明,线程将要退出,需更改当前线程池中的线程数量*/
/*.....*/
pthread_mutex_lock(&pool->thcount_lock);
pool->num_threads--;
pthread_mutex_unlock(&pool->thcount_lock);
return NULL;
}

/*创建线程*/
int create_thread(threadpool *pool, thread **pthread, int id) {
    //为 thread 分配内存空间
    *pthread = (struct thread *)malloc(sizeof(struct thread));
    if (pthread == NULL) {
        // error("creat_thread(): Could not allocate memory for thread\n"); ???
        perror("creat_thread(): Could not allocate memory for thread\n");
        return -1;
    }
    //设置这个 thread 的属性
    (*pthread)->pool = pool; // pool 地址?
    (*pthread)->id = id;
    //创建线程
    pthread_create(&((*pthread)->pthread), NULL, thread_do, (void *)(*pthread));
    // printf("end?id:%d\n", id);
    pthread_detach(
        (*pthread)->pthread); // 设置为 detach 属性,一旦结束,自动释放,不用 join
    /*
    * pthread_attr_t attr;
    * pthread_attr_init(&attr);
    * pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    * 线程创建非常快,没有返回就结束,在创建线程里调用pthread_cond_wait;
    * 这为对条件变量的修改,需要加锁
    */
    return 0;
}

struct threadpool *initTheadPool(int num_threads) {

```

```

threadpool *pool; //创建线程池空间
pool = (threadpool *)malloc(sizeof(struct threadpool));
pool->num_threads = 0;
pool->num_working = 0;

pool->is_alive = true; //????????????????????
//初始化互斥量和条件变量
pthread_mutex_init(&(pool->thcount_lock), NULL);
pthread_cond_init(&pool->threads_all_idle, NULL);

//初始化任务队列
init_taskqueue(&pool->queue); //****需实现****
pool->threads = //创建线程数组
    (struct thread **)malloc(
        num_threads *
        sizeof(
            struct thread *)); // use pool->num_threads ..... 啊, 终于找到错了
for (int i = 0; i < num_threads; ++i) { // i 为线程 id
    create_thread(pool, &(pool->threads[i]), i);
    // printf("create: %d\n", i);
}
//等所有的线程创建完毕, 在每个线程运行函数中将进行 pool->num_threads++ 操作
//因此, 此处为忙等待, 直到所有的线程创建完毕, 并马上运行阻塞代码时才返回。
while (pool->num_threads != num_threads) {
}
return pool;
}

/*向线程池中添加任务*/
void addTask2ThreadPool(threadpool *pool, task *curtask) {
    //将任务加入队列
    //****需实现****
    push_taskqueue(&pool->queue, curtask);
    pthread_mutex_lock(&pool->queue.has_jobs->mutex);
    // pool->queue.has_jobs->status = true;
    if (pool->num_working != pool->num_threads)
        pthread_cond_signal(&pool->queue.has_jobs->cond); // 提出阻塞线程
    // 任务数大于线程数时, signal 浪费了。
    // pthread_cond_broadcast(&pool->queue.has_jobs->cond); // 提出阻塞线程
    pthread_mutex_unlock(&pool->queue.has_jobs->mutex);
}

/*等待当前任务全部运行完*/
void waitThreadPool(threadpool *pool) {
    pthread_mutex_lock(&pool->thcount_lock);
    while (pool->queue.len || pool->num_working) { // cond 的机制不是很懂。。
        pthread_cond_wait(&pool->threads_all_idle,
            &pool->thcount_lock); // 这玩意会释放锁, 嗯
    }
    pthread_mutex_unlock(&pool->thcount_lock);
}

/*销毁线程池*/
void destoryThreadPool(threadpool *pool) {
    //如果当前任务队列中有任务, 需等待任务队列为空, 并且运行线程执行完任务后
    /*.....*/
    waitThreadPool(pool);
    //销毁任务队列

```



```

//****需实现****
destroy_taskqueue(&pool->queue);
//销毁线程指针数组, 并释放所有为线程池分配的内存
/*.....*/
for (int i = 0; i < pool->num_threads; ++i) { // i 为线程 id
    free(pool->threads[i]);
}
free(pool->threads);
pthread_mutex_destroy(&pool->thcount_lock);
pthread_cond_destroy(&pool->threads_all_idle);
free(pool);
}
/*获得当前线程池中正在运行线程的数量*/
int getNumofThreadWorking(threadpool *pool) { return pool->num_working; }

```

thread.c:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/wait.h>
#include <wait.h>

#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <pthread.h>
#include <signal.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#include <semaphore.h>
#include <sys/mman.h> // shared memory

#include <sys/time.h>

#include "threadPool.h"

#define SEM_NAME "sem_count"
#define SHM_NAME "mmap_example"

#define VERSION 23
#define BUFSIZE 8096
#define ERROR 42
#define LOG 44
#define FORBIDDEN 403
#define NOTFOUND 404

#ifndef SIGCLD
#define SIGCLD SIGCHLD
#endif

```

```

struct {
    char *ext;
    char *filetype;
} extensions[] = {"gif", "image/gif"},
                 {"jpg", "image/jpg"},
                 {"jpeg", "image/jpeg"},
                 {"png", "image/png"},
                 {"ico", "image/ico"},
                 {"zip", "image/zip"},
                 {"gz", "image/gz"},
                 {"tar", "image/tar"},
                 {"htm", "text/html"},
                 {"html", "text/html"},
                 {0, 0}};

struct timeval start, end;
struct timeval start_web;           // 统计单个 web 的时间
struct timeval start_totol, end_totol; // 统计总时间
double timeuse = 0, time_totol = 0;
sem_t *psem;
int shm_fd;
void *memPtr;
typedef struct {
    int hit;
    int fd;
} webparam;

unsigned long get_file_size(const char *path) {
    unsigned long filesize = -1;
    struct stat statbuff;
    if (stat(path, &statbuff) < 0) {
        return filesize;
    } else {
        filesize = statbuff.st_size;
    }
    return filesize;
}

void logger(int type, char *s1, char *s2, int socket_fd) {
    // s1 is request:, s2 is GET /
    int fd;
    char logbuffer[BUFSIZE * 2];
    switch (type) {
        case ERROR:
            (void)sprintf(logbuffer, "ERROR: %s:%s Errno=%d exiting pid=%d", s1, s2,
                          errno, getpid());
            break;
        case FORBIDDEN:
            (void)write(socket_fd,
                        "HTTP/1.1 403 Forbidden\nContent-Length: 185\nConnection: "
                        "close\nContent-Type: text/html\n\n<html><head>\n<title>403 "
                        "Forbidden</title>\n</head><body>\n<h1>Forbidden</h1>\n<h1>The "
                        "requested URL, file type or operation is not allowed on this "
                        "simple static file webserver.\n</body></html>\n",
                        271);
            (void)sprintf(logbuffer, "FORBIDDEN: %s:%s", s1, s2);
            break;
        case NOTFOUND:
    }
}

```

```

        (void)write(socket_fd,
            "HTTP/1.1 404 Not Found\nContent-Length: 136\nConnection: "
            "close\nContent-Type: text/html\n\n<html><head>\n<title>404 "
            "Not Found</title>\n</head><body>\n<h1>Not Found</h1>\nThe "
            "requested URL was not found on this server.\n</body></html>\n",
            224);
        (void)sprintf(logbuffer, "NOT FOUND: %s:%s", s1, s2);
        break;
    case LOG:
        (void)sprintf(logbuffer, " INFO: %s:%s:%d", s1, s2, socket_fd);
        break;
    }
    /* No checks here, nothing can be done with a failure anyway */
    if ((fd = open("nweb.log", O_CREAT | O_WRONLY | O_APPEND, 0644)) >= 0) {
        (void)write(fd, logbuffer, strlen(logbuffer));
        (void)write(fd, "\n", 1);
        (void)close(fd);
    }
    // if(type == ERROR || type == NOTFOUND || type == FORBIDDEN) exit(3);
}

/* this is a web thread, so we can exit on errors */
void *web(void *data) {
    sem_wait(psem); // 进程数加 1
    *((double *)memPtr + 5)++;
    sem_post(psem);
    gettimeofday(&start_web, NULL);

    int fd;
    int hit;
    int j, file_fd, buflen;
    long i, ret, len;
    char *fstr;
    char buffer[BUFSIZE + 1]; /* static so zero filled */
    webparam *param = (webparam *)data;
    fd = param->fd;
    hit = param->hit;

    gettimeofday(&start, NULL);
    ret = read(fd, buffer, BUFSIZE); /* read web request in one go */
    gettimeofday(&end, NULL);
    double timeuse = (end.tv_sec - start.tv_sec) +
        (double)(end.tv_usec - start.tv_usec) / 1000000.0;
    /*printf("平均每个客户端完成读 socket 时间为 %fms\n", timeuse * 1000);*/
    sem_wait(psem);
    *((double *)memPtr + 1) += timeuse * 1000;
    sem_post(psem);

    if (ret == 0 || ret == -1) { /* read failure stop now */
        logger(FORBIDDEN, "failed to read browser request", "", fd);
    } else {
        if (ret > 0 && ret < BUFSIZE) /* return code is valid chars */
            buffer[ret] = 0;
        /* terminate the buffer */
        else
            buffer[0] = 0;
        for (i = 0; i < ret; i++) /* remove cf and lf characters */
            if (buffer[i] == '\r' || buffer[i] == '\n')

```

```

        buffer[i] = '*';
    logger(LOG, "request", buffer, hit);
    if (strncmp(buffer, "GET ", 4) &&
        strncmp(buffer, "get ", 4)) { // GET 从何而来, socket
        logger(FORBIDDEN, "only simple get operation supported", buffer, fd);
    }
    for (i = 4; i < BUFSIZE; i++) { /* null terminate after the second space to
                                    ignore extra stuff */
        if (buffer[i] == ' ') { /* string is "get url " +lots of other stuff */
            buffer[i] = 0;
            break;
        }
    }

    for (j = 0; j < i - 1; j++) /* check for illegal parent directory use .. */
        if (buffer[j] == '.' && buffer[j + 1] == '.') {
            logger(FORBIDDEN, "parent directory (..) path names not supported",
                buffer, fd);
        }

    if (!strncmp(&buffer[0], "GET /\0", 6) ||
        !strncmp(&buffer[0], "GET /\0", 6)) /* convert no filename to
index file */
        (void)strcpy(buffer, "GET /index.html");
    /* work out the file type and check we support it */
    buflen = strlen(buffer);
    fstr = (char *)0;
    for (i = 0; extensions[i].ext != 0; i++) {
        len = strlen(extensions[i].ext);
        if (!strncmp(&buffer[buflen - len], extensions[i].ext, len)) {
            fstr = extensions[i].filetype;
            break;
        }
    }
    if (fstr == 0)
        logger(FORBIDDEN, "file extension type not supported", buffer, fd);
    if ((file_fd = open(&buffer[5], O_RDONLY)) ==
        -1) { /* open the file for reading */
        logger(NOTFOUND, "failed to open file", &buffer[5], fd);
    }

    logger(LOG, "send", &buffer[5], hit);
    len = (long)lseek(file_fd, (off_t)0,
        SEEK_END); /* 使用 lseek 来获得文件长度,比较低效*/
    (void)lseek(file_fd, (off_t)0, SEEK_SET);
    /* 想想还有什么方法来获取*/
    gettimeofday(&start, NULL);
    (void)sprintf(buffer,
        "http/1.1 200 ok\nserver: nweb/%d.0\ncontent-length: "
        "%ld\nconnection: close\ncontent-type: %s\n\n",
        VERSION, len, fstr); /* header + a blank line */
    logger(LOG, "header", buffer, hit);
    gettimeofday(&end, NULL);
    timeuse = end.tv_sec - start.tv_sec +
        (double)(end.tv_usec - start.tv_usec) / 1000000.0;
    /*printf("平均每个客户端完成写日志数据时间为 %fms\n", timeuse * 1000);*/
    sem_wait(psem);
    *((double *)memPtr + 4) += timeuse * 1000;
    sem_post(psem);

```

```

(void)write(fd, buffer, strlen(buffer)); // 往 fd 中写?

gettimeofday(&start, NULL);
/* send file in 8kb block - last block may be smaller */
while ((ret = read(file_fd, buffer, BUFSIZE)) > 0) {
    (void)write(fd, buffer, ret);
}
gettimeofday(&end, NULL);
timeuse = end.tv_sec - start.tv_sec +
    (double)(end.tv_usec - start.tv_usec) / 1000000.0;
/*printf("平均每个客户端完成读网页数据时间为 %fms\n", timeuse * 1000);*/
/*printf("平均每个客户端完成写 socket 的时间为 %fms\n", timeuse * 1000);*/
sem_wait(psem);
*((double *)memPtr + 2) += timeuse * 1000;
*((double *)memPtr + 3) += timeuse * 1000;
sem_post(psem);

gettimeofday(&end, NULL);
timeuse = (end.tv_sec - start_web.tv_sec) +
    (double)(end.tv_usec - start_web.tv_usec) / 1000000.0;
/*printf("平均每个客户端完成请求处理时间为 %fms, hit: %d\n", timeuse *
    * 1000, */
/*hit);*/

sem_wait(psem);
*((double *)memPtr += timeuse * 1000;
*((double *)memPtr + 5)--;
sem_post(psem);
if (!(*((double *)memPtr + 5)) && hit > 200000) {
    /*if (!(*((double *)memPtr + 5))) {*/
    gettimeofday(&end_totol, NULL);
    time_totol =
        (end_totol.tv_sec - start_totol.tv_sec) +
        (double)(end_totol.tv_usec - start_totol.tv_usec) / 1000000.0;
    char buffer[BUFSIZE + 1]; /* static so zero filled */
    (void)sprintf(buffer,
        "共用 %fms 成功处理 %d 个客户端请求, 其中\n "
        "平均每个客户端完成请求处理时间为 %fms\n "
        "平均每个客户端完成读 socket "
        "时间为 %fms\n 平均每个客户端完成写 socket 时间为 "
        "%fms\n "
        "平均每个客户端完成读网页数据时间为 %fms\n "
        "平均每个客户端完成写日志数据时间为 %fms\n",
        time_totol * 1000, hit, *((double *)memPtr / hit,
        *((double *)memPtr + 1) / hit,
        *((double *)memPtr + 2) / hit,
        *((double *)memPtr + 3) / hit,
        *((double *)memPtr + 4) / hit); /* header + a blank line
        */

    logger(LOG, "time", buffer, hit);
}

usleep(10000); /*在 socket 通道关闭前, 留出一段信息发送的时间*/
close(file_fd);
}

close(fd);

```

```

//释放内存
free(param);
// pthread_exit(0);
}

int main(int argc, char **argv) {

    int i, port, pid, listenfd, socketfd, hit;
    socklen_t length;
    static struct sockaddr_in cli_addr; /* static = initialised to zeros */
    static struct sockaddr_in serv_addr; /* static = initialised to zeros */
    if (argc < 3 || argc > 3 || !strcmp(argv[1], "-?")) {
        (void)printf(
            "hint: nweb Port-Number Top-Directory\t\tversion %d\n\n"
            "\tnweb is a small and very safe mini web server\n"
            "\tnweb only servers out file/web pages with extensions named below\n"
            "\t and only from the named directory or its sub-directories.\n"
            "\tThere is no fancy features = safe and secure.\n\n"
            "\tExample: nweb 8181 /home/nwebdir &\n\n"
            "\tOnly Supports:",
            VERSION);
        for (i = 0; extensions[i].ext != 0; i++)
            (void)printf(" %s", extensions[i].ext);
        (void)printf(
            "\n\tNot Supported: URLs including \"..\", Java, Javascript, CGI\n"
            "\tNot Supported: directories / /etc /bin /lib /tmp /usr /dev /sbin\n"
            "\tNo warranty given or implied\n\tNigel Griffiths nag@uk.ibm.com\n");
        exit(0);
    }
    if (!strcmp(argv[2], "/") || !strcmp(argv[2], "/etc", 5) ||
        !strcmp(argv[2], "/bin", 5) || !strcmp(argv[2], "/lib", 5) ||
        !strcmp(argv[2], "/tmp", 5) || !strcmp(argv[2], "/usr", 5) ||
        !strcmp(argv[2], "/dev", 5) || !strcmp(argv[2], "/sbin", 6)) {
        (void)printf("ERROR: Bad top directory %s, see nweb -?\n", argv[2]);
        exit(3);
    }
    if (chdir(argv[2]) == -1) {
        (void)printf("ERROR: Can't Change to directory %s\n", argv[2]);
        exit(4);
    }

    /* Become deamon + unstopable and no zombies children (= no wait()) */
    /*if (fork() != 0)*/
    /*return 0; [> parent returns OK to shell <]*/

    /*(void)signal(SIGCLD, SIG_IGN); [> ignore child death <]*/
    /*(void)signal(SIGHUP, SIG_IGN); [> ignore terminal hangups <]*/
    /*for (i = 0; i < 32; i++)          // what meaning?*/
    /*(void)close(i);*/
    /*close open files*/

    // 设置组的 pid 为 点前进程的 pid
    /*(void)setpgrp(); [> break away from process group <]*/

    logger(LOG, "nweb starting", argv[1], getpid());
    /* setup the network socket */
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        logger(ERROR, "system call", "socket", 0);

```

```

port = atoi(argv[1]);
if (port < 0 || port > 60000)
    logger(ERROR, "Invalid port number (try 1->60000)", argv[1], 0);

//初始化线程属性,为分离状态
/*pthread_attr_t attr;*/
/*pthread_attr_init(&attr);*/
/*pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);*/

/*pthread_t pth;*/
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(port);
if (bind(listenfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    logger(ERROR, "system call", "bind", 0);
if (listen(listenfd, 64) < 0)
    logger(ERROR, "system call", "listen", 0);

if ((psem = sem_open(SEM_NAME, O_CREAT, 0777, 1)) ==
    SEM_FAILED) { // 信号量是否为全局变量, fork 会咋样?
    perror("create semaphore error");
    exit(1);
}

if ((shm_fd = shm_open(SHM_NAME, O_RDWR | O_CREAT, 0777)) < 0) {
    perror("create shared memory object error");
    exit(1);
}
ftruncate(shm_fd, 6 * sizeof(double));
memPtr = mmap(NULL, 6 * sizeof(double), PROT_READ | PROT_WRITE, MAP_SHARED,
    shm_fd, 0);
if (memPtr == MAP_FAILED) {
    perror("create mmap error");
    exit(1);
}

*(double *)memPtr = 0;
*((double *)memPtr + 1) = 0;
*((double *)memPtr + 2) = 0;
*((double *)memPtr + 3) = 0;
*((double *)memPtr + 4) = 0;
*((double *)memPtr + 5) = 0;

gettimeofday(&start_totol, NULL); // 统计总时间

threadpool *pool = initTheadPool(80);
for (hit = 1;; hit++) { // accept and create pthread
    /*printf("hello\n"); // 在这不能输出??? 到哪去了*/
    length = sizeof(cli_addr);
    if ((socketfd = accept(listenfd, (struct sockaddr *)&cli_addr, &length)) <
        0)
        logger(ERROR, "system call", "accept", 0);

    webparam *param = (webparam *)malloc(sizeof(webparam));
    param->hit = hit;
    param->fd = socketfd;
    /*if (pthread_create(&pth, &attr, &web, (void *)param) < 0) {*/
    /*logger(ERROR, "system call", "pthread_create", 0);*/
}

```

```
/*}*/  
task *web_task = (task *)malloc(sizeof(task));  
web_task->arg = (void *)param;  
web_task->function = web;  
web_task->next = NULL; // next need init?  
  
addTask2ThreadPool(pool, web_task);  
}  
}
```