電機所碩二

張光耀

R04921005

SP MP4

Report

(1) Thread pool implementation

我的 thread pool implementation 主要是先由main thread產生對應於 server.cfg 內定義之 thread 數目(透過parse_arg此函式得到config檔的資訊)，main thread 主要的工作為接收 request 與建立新連線(new client)，work thread 主要負責處理request。

為了實作此mechanism，需定義request 的資料型態：

```
struct request{
    csiebox_server* server;
    int conn_fd;
    struct request* next;
};
```

用於建立request的link list，讓work thread在這個link list去取得request，值得注意的是，為了避免multithread取用request link list的混亂情形，在有需要修改request link list時就必須使用mutex來避免race condition，當一個request成功被某一個thread取用完畢時，再利用condition variable 通知其他thread來取用list。在request的設計上，我是採用 one thread 對應一client channel，也就是request的處理單位是以connection file descriptor來做區分，所以必須維護哪些已連上線的socket是正在被處理的，降低設計的複雜度。

Main thread 處於一無窮迴圈，利用select 監控新連線的request 與 已連線client 的I/O需求，根據相對應的request 去增加 request link的內容，並且根據MP4的需求會判斷有無 free thread來處理 request，若沒有則不會將其request納入request link list，並通知client 端等候三秒再次傳送需求。work thread 也處於一無窮迴圈，主要工作分為三部份，第一部分為等待request (利用condition variable 做通知)，第二部分為取用 link list 及 update link list ，第三部份即處理request。根據MP4 的要求，在處理request時會判斷該更新、上傳的檔案是否已經有別的process擁有其exclusive lock，若有則傳送訊息給client端，並要求其等候三秒在傳送一次request。

work thread 的生成以及handle request 函式如下：

```
    // multithread programming
    pthread_t *threads = (pthread_t*)malloc(tmp->arg.thread_num * sizeof(pthread_t));
  int *thr_id = (int*)malloc(tmp->arg.thread_num * sizeof(int));

    for (int i = 0; i < tmp->arg.thread_num; i++){
        thr_id[i] = i;
      pthread_create(&threads[i], NULL,handle_req_loop, (void*)&thr_id[i]);
    }
    sleep(2);           //wait for thread creation
}
```

```
void* handle_req_loop(void* data)
{
    int thr_num = *((int*)data);
    pthread_mutex_lock(&req_mutex);
    fprintf(stderr, "Thread%d starts\n", thr_num);
    while(1){
        if ( req_num > 0)
        {
            fprintf(stderr, "BEFORE: the req num = %d\n", req_num);
            struct request *a_request;
            thr_state[thr_num] = 0;                                     //set the thread state busy
            a_request = get_req(req_queue, last_req, &req_num);
            int rc = pthread_mutex_unlock(&req_mutex);

            if ( a_request)
            {
                fprintf(stderr, "thread%d start to handle request, conn_fd is %d\n", thr_num, a_request->conn_fd);
                handle_request(a_request->server, a_request->conn_fd);
                client_state[a_request->conn_fd] = 0;
                free(a_request);
                thr_state[thr_num] = 1;                                 //set the thread state idle
                fprintf(stderr, "thread%d finish handling request, conn_fd is %d\n", thr_num, a_request->conn_fd);
              fprintf(stderr, "AFTER: the req num = %d\n", req_num);
            }
        }
        else
        {
            fprintf(stderr, "thread%d waits for the request\n", thr_num);
            fflush(stderr);
            pthread_cond_wait(&got_req, &req_mutex);
            fprintf(stderr, "thread%d gets the request\n", thr_num);
            fflush(stderr);
        }
    }
}
```

(2) Discuss how to use process instead of thread to handle multiple clients and compare throughput of these two approaches

若要使用multi process 來handle MP4的要求，應該是利用 parent process 來等待 request ，每當有新的I/O 需求或連線需求，就fork()出一個child process來專門處理該 channel 的request，而parent process 則負責繼續等待新的request。

以memory的角度來分析這兩個mechanism的話，我認為thread pool 擁有較好的效率，因為 fork() 出新的process，會有較多冗餘的memory一同被複製。而以delay time 來看的話，或許multi process反應會較快，因為不需maintain request link list。