SUNY AT BUFFALO 2012

MULTI-THREADED WEB SERVER

TEAM G7

Divya Chaudhary: divyacha@buffalo.edu

Si Chen: schen23@buffalo.edu

Suqiang Chen: suqiangc@buffalo.edu

CSE 421/521 – OPERATING SYSTEM PROJECT-1

Objective

The objective of this project is to implement a multithreaded web server “myhttpd” in C/C++ on a UNIX-based platform.

Responsibilities of each team member in the project

Divya Chaudhary	Multi-threading, queuing, scheduling, synchronization, logging, design document
Suqiang Chen	Protocol, logging, scheduling, design document
Si Chen	Socket Programming, options handling, multi-threading, queuing, scheduling, synchronization

Data Structures

1. **Multithreading:**

   We have created a threadpool containing threadnum execution threads and a queue containing all the threads.This is the structure for Thread and threadpool.

```
typedef struct Thread{

            pthread_t thread_id;       // thread ID
            long thread_count;         // # connections handled
} ;

typedef struct thpool_t{
            pthread_t* thread_id;              //< pointer to threads' ID
            int thread_count;                  //< amount of threads
            thpool_job_queue* jobqueue;  //< pointer to the job queue
}
```

2. **Queuing:**
   A structure thpool_job_queue is used for queuing all the requests. We store all the requests in a structure thpool_job_t and these jobs in the queue.

```
 typedef struct thpool_job_queue{
        thpool_job_t*    head;              // head of the queue
        thpool_job_t*    tail;              // tail of the queue
        int              jobN;              // number of jobs
        sem_t*           queueSem;        // x sem
 }
  typedef struct thpool_job_t{
```

```
            FUNC    function;
            void*    arg;     //function parameter
            int         socket_client_ID;
            long   filesize;
            char* request;
            struct thpool_job_t* prev;          // aim to the previous node
            struct thpool_job_t* next;           // aim to the next node
    }
```

3. **Scheduling:**
   Scheduling threads removes the requests (jobs) from the job queue in FCFS or SJF manner and assign them to the execution threads created in threadpool

4. **Synchronization:**
   We have used two mutex and two condition variable for synchronization amongst threads.

   Mutex:

   (a) clientId_mutex: This mutex is used for synchronizing the request for job queue.

   (b) client_enter_cond: This mutex is used for synchronizing the threads in threadpool ie. It is used by execution threads.

   Condition variables:

   (a) clientId_sche: This is used for synchronizing the listener and scheduler threads. Scheduler thread wait on this condition is there is no request in the job queue and listener thread signals on this condition when it has added job in the job queue.

   (b) clientId_req_cond: Execution thread waits on it if there is no job to be done and broadcast on this condition variable when the request has been executed.

Context switch

Context switch will happen when the scheduler thread is informed by listener thread to choose a thread from queue to manage the incoming request. When there is any incoming request, condition variable, “clientId_sche” will be triggered, and the scheduler will change its state from “wait” to “run”. Also, when there is no free thread to choose, scheduler thread just waits and change its state from “run” to “wait”.

Context switches will also happen on the execution threads in the thread pool. The execution thread changes its state from “wait” to “run” whenever it is chosen by scheduler and from “run” to “wait” when it has finished its job.

Race conditions avoided

Race conditions occur when different threads want to modify the shared resources. The various global shared resources, such as thread pool and job queue are used by the web server. We have prevented the race conditions on these by using mutex locks. For example, if both listener thread and scheduling thread wants to use the queue at the same time, they cannot use it, because every time a thread starts adding or removing jobs from queue, it acquires lock on the queue which prevents other thread from using the job queue at the same time. Also, each execution thread acquires a lock on thread pool so that the listener thread does not change the contents of thread pool at the same time.

Critique our design

Advantages: Threadpool is being used for creating the threads so that thread creation overhead is minimized and there is a limit on number of threads that could be executed. Also, synchronization prevents race condition.

Disadvantages: The option –r *dir* is not available, so the root directory is default.

Online References

[1] *Understanding Unix/Linux Programming,* Bruce Molay. 2004.

[2] http://see.xidian.edu.cn/cpp/u/hanshu/

[3] http://bbs.chinaunix.net/thread-2169150-1-1.html