Ketan Deshmukh
UB # 50097066
ketandes@buffalo.edu

DESIGN DOCUMENT : MULTI-THREADED WEB SERVER IN C

PART I

Queuing and Scheduling

I have implemented two different queues - Waiting Queue and Ready Queue. These two queues are linked lists. Waiting Queue holds all the requests listener thread listens and which are to be scheduled. Once scheduler thread starts executing, it dequeues a request at a time from waiting queue (according to the policy - FCFS or SJF) and puts it into the Ready Queue. Hence, Ready Queue holds all the requests that are ready for execution by worker threads.

The structure of a node of either queue looks like -

```
typedef struct node{
   int acceptfd;                // socket on which to communicate
   char file_name[20];
   char file_path[200];
   char request_type[5];        // GET or HEAD
   int file_size;
   char client_ip[20];
   char content_type[15];       // Text or HTML or IMAGE
   char arrival_time[30];
   char current_dir[200];
   struct node *next;
   struct node *previous;
} Node;
```

Multithreading

In this project, I have created a Listener thread, a Scheduler thread and a pool of worker threads. Data structure used for storing the information of pool of worker threads is an Array.
All the above mentioned threads are POSIX threads.

Synchronization

To maintain the consistency and integrity of the data in this multithreading scenario, I've used locks and condition variable provided by the *pthread.h* library. I've used following methods provided by it -

pthread_mutex_init (mutex,attr)
pthread_mutex_destroy (mutex)
pthread_mutex_lock (mutex)
pthread_mutex_unlock (mutex)
pthread_cond_init (condition, attr)
pthread_cond_destroy (condition)
pthread_cond_wait (condition,mutex)
pthread_cond_signal (condition)
pthread_cond_broadcast (condition)

PART II

How are context switches between threads implemented in your code?

Every request follows the following form of life cycle :

```
                    (1)            (2)
Start ----> Listener ----> Scheduler ----> Worker ---> End
```

As shown in the diagram above there are 2 main context switches. Context switches are handled using the queue data structures I explained in PART I. When a Listener thread appends a request into the Waiting Queue, its context is switched to Scheduler thread. Similarly when the Scheduler thread puts it into the Ready Queue, context is switched to the Worker thread.

PART III

How are your race conditions avoided in your code?

As the Waiting Queue and Ready Queue are the shared data structures between the Listener-Scheduler and Scheduler-Worker threads respectively, race conditions are bound to occur. Hence to prevent such situations, I have used Locks and Condition variables in the following manner.

- Adding a new node into or removing a node from either of the queues is my critical section. To perform above operations on the queue, a thread has to obtain the lock first. Once the operation is performed, that thread releases the lock.

- If there are no elements present to consume in the queue, the consumers(i.e. Scheduler for Waiting Queue and Worker for Ready Queue) wait till they receive a signal from corresponding producers.
- Whenever a producer adds an element into the empty queue, it signals the waiting consumers about the presence of an element it/they can consume.

PART IV

Briefly critique your design, pointing out advantages and disadvantages in your design choices.

Advantages :

- Using linked list as data storage gives flexibility and dynamicity. You can allocate as many requests as your memory allows.
- Usage of locks instead of Busy Waiting helps CPU utilise its time efficiently.

Disadvantages :

- To reduce complexity of code I have used to separate queues to hold the requests. This results in some memory wastage. In the ideal implementation this could have been done with just a simple flag variable indicating the status of a request (i.e. at a particular instance of time where does the request belong to - Waiting Queue or Ready Queue).

PART V

Please cite any online or offline resources you consulted while preparing your project, other than the course materials.

I have used several man pages to understand the functionalities of several string and file related functions. Apart from these I have used, following link to understand the fundamentals of the POSIX threads -

https://computing.llnl.gov/tutorials/pthreads/