CSE521 Project 1 design report

I individually finish the project:

For learning more about synchronization, we had developed most of the code individually and it overlaps in most of the areas. So as we have two complete set of code, I submit one set of code and the others are ready to submit other set of code.

Email address:qiantao@buffalo.edu

# Thread datapath:

## Listener thread

The listener thread wait new connections from clients: if a new client is setup, listener thread parse HTTP request and retrieve file length which is the criteria of scheduling in this project, then form a HTTP session, send it to “Queue A”(in real code, it is “*static struct queuescheduler\_t queuescheduler*”).

## Scheduler thread

Scheduler thread get http session from “Queue A” according to different scheduler policy.

The relationship between Listener thread and scheduler thread is a producer-consumer problem. One mutex(*queuescheduler\_t.lock*) is used to protect the “shared” Queue A . One condition variable(*queuescheduler\_t.notify*) is used to syncronize Listener thread and scheduler thread.

Queue B

Queue A

exec thread1

Scheduler thread

Listener thread

Fifo

FFIFO

exec thread2

exec thead3

## How to implement SJF

When scheduler thread retreive HTTP session from “Queue A”, it compare the file length of all queue entries, and select the shortest session.

## Execution thread

According to project requirment, there are multiple exectuion threads, which are created once application startup. Every exectution thread run one HTTP request and send content back to client.

I reuse online code called threadpool to implement thread pool.

The relationship between Scheduler thread and execution threads is a one produer-multiple consumer problem. Same to the above, one mutex(*threadpool\_t.lock*) to protect the “shared” Queue B and use one condition variable(*threadpool\_t.notify*) to syncronize scheduler thread and execution threads.

## Context switches

When application startup, all threads are created. But scheduler thread enter sleep.

Before scheduler thread wakeup, listenter thread put all HTTP requests it receive to “queue A”. When scheduler thread wakeup, it get HTTP requests from “queue A” , and send these requests to   
“queue B”,and notify the execution threads. Once the execution threads are notified, they can process HTTP request and send file back to client.

## Race conditions Avoidiance

All shared resources between threads already are protected by mutex, so race conditions are avoided.

# Soucre Files

## Server.c

The main entry, parse command options,create threads

## queueschedule.c

Listener thread; scheduler thread; exectuion threads’ worker function

## minilog.c

support debugging output and log when the application is daemonized.

## protocol.c

parse HTTP request, encapuslate HTTP response

## poollib/threadpool.c

third party online code to implment thread pool

# Some advanced Design

I resolve some little but important issues in this design.

## Flush logging to harddisk file

In minilog(), when call vfprintf() try to print out to the log file, there are no any content wrote to the log file. I add fflush(),which flush from glibc buffer to OS;then add fsync(), which flush from OS to disk. The issue is resloved

## Reinvent daemon()

Posix daemon() is not sutiable for the requirement because it close all file descriptors of parent , even including logging file descriptor. So after daemonized, the logging file descripter is useless. On the other hand, it is a bad solution that open logging file descriptor after daemonizing, because how can print out an error message to standout when open logging file unsuccessfully in “daemonized” mode.

**thread-safe API**

Need to pick thread-safe version in multithread programming such as localtime\_r() in this project.

# Design flaw

## Only one condition variable in consumer-producer problem

For “queue A”, if listener thread find the queue is full, listener thread just return fail, this design is flawed because the current design only provide one condition variable. The correct solution is that two condition variables should be provided, one for “empty”, one for “full”. The “queue B” has the same flaw.

# Some issues need to finished

There are some issues I have not enough time to resolve it, they all are related to socket programming.

## Source Ip address of client

In the requirement of logging, the beginning should be the ip address of client,now my desgin doesn’t support it. I have tried to resolve it, but it bring some fatal side effect.

## TIME\_WAIT of Server socket

When kill server and restart server in TIME\_WAIT interval, client can’t connect server

## Reuse of Server port

Now the current design permits to start two server which use the same port, this case should be avoided.

# Third party code

## scandir

It help to get an alphanumeric sorted file lists under one directory.

<http://www.c.happycodings.com/Gnu-Linux/code7.html>

## threadpool

Actually the oneline code is flawed, and it bring flaw to my design.

Attachment 1---README

Notice:

this version has a flaw:When kill server and restart server again in TIME\_WAIT, client can’t connect server

1.Compile:

go to linux host

there are two makefiles in the project, one is under poollib/

#cd poollib/

#run make

#cd ..

#run make

2.start server:

#./myhttpd -d -p 22222

(suppose myhttpd run on host metallica.cse.buffalo.edu)

3.test plan

please use Firefox as your browser

3.1 verify normal GET function

test step: input "http://metallica.cse.buffalo.edu:22222/Server.c" to your browser address

expect:wait about one minuate, you can get Server.c in browser

3.2 parser "~", can translate to "~/myhttpd/"

test step:input "http://metallica.cse.buffalo.edu:22222/~minilog.c" to your browser address

expect:your can get ~/myhttpd/minilog.c in browser

3.3 verify:when GET a directory, and file "index.html" exist under the directory, "index.html" will be send to client

test step:

3.3.1 under metallica

#cd ~/myhttpd/poolib

#cp makefile index.html

3.3.2

input "http://metallica.cse.buffalo.edu:22222/poollib" to your browser

expect:you can get ~/myhttpd/poollib/index.html in browser

3.4verify: when GET a diretory which doesn't include "index.html"

test step:

3.4.1 under metallica

#cd ~/myhttpd/poolib

#rm index.html

3.4.2

input "http://metallica.cse.buffalo.edu:22222/poollib" to your browser

expect:you can get a file list under the direcotry,which are sort alphanumeric

3.5verify FCFS scheduler(default scheduler)

test step:

3.5.1 start server

#./myhttpd -d -p 22222 -t 120

3.5.2 first get a longer file

input "http://metallica.cse.buffalo.edu:22222/Server.c" to browser

3.5.3 first get a shorter file

input "http://metallica.cse.buffalo.edu:22222/minilog.h" to browser

expect:watch the debugging informaion in metallica's console,you will find "Server.c" schedule first, "minilog.h" schedule late

3.6 verify SJF scheduler

3.6.1 start server

#./myhttpd -d -p 22222 -t 120 -s SJF

3.5.2 first get a longer file

input "http://metallica.cse.buffalo.edu:22222/Server.c" to browser

3.5.3 first get a shorter file

input "http://metallica.cse.buffalo.edu:22222/minilog.h" to browser

expect:watch the debugging informaion in metallica's console,you will find "Server.c" schedule late, "minilog.h" schedule first

3.7 daemonize and logging file

3.6.1 start server

#./myhttpd -d -p 22222 -t 60 -l ~/log.txt

3.5.2 get a file

input "http://metallica.cse.buffalo.edu:22222/Server.c" to browser

3.5.3

under metallica

#cat ~/log.txt

expect:wait about for one miniute(default sleep time),you will find myhttpd log some scheduler information to log.txt

#ps -aef | grep myhttpd

expect:you will find the ppid of myhttpd is 1, and there are not any terminal attached to myhttpd, so it is already daemonized