Assignment 2 Design Document

Alan Caro
CruzID: alcaro

CSE130, Fall 2019

1 Goal

The goal of this program is to make our previous single threaded HTTP server into a multithreaded server and add a logging feature to it.

2 Assumptions

I am assuming the user will only use the curl command. No directories will be sent by the client or requested. When no Content-Length is passed the client will have to exit the process by himself/herself. Also, I am assuming the user will send request with this format:

PUT:
curl -T localfile http://localhost:8080 --request-target filename_27_character

GET:
curl http://localhost:8080 --request-target filename_27_character

3 Design

The general approach I am taking is to check what request and file name the user passed. Then, if the request and file name is valid, I open a socket then process the GET or PUT command accordingly. If there is any errors like an invalid file or request I send the appropriate error code.

To handle multithreading, I followed the producer and a consumer problem presented in class. My producer is my main function and the consumer is the start function. Producer accepts the connection and consumer uses that connection to process a request. The shared variables are MYQUEUE and GLOBAL_OFFSET, MYQUEUE is a queue from the C++ STD that holds the sockets awaiting to be processed. GLOBAL_OFFSET saves where on the log file I should start writing. MYQUEUE is modified in the producer code, when a connection is accepted, the socket is added to the queue. Later consumer or start function grabs the front of the queue, pops it then processes the request. GLOBAL_OFFSET is modified every time I need to write to the log file. I add the length of the string that I am writing to the offset. The critical regions are where I modify MYQUEUE or GLOBAL_OFFSET. Logging each request contiguously works since I use a global offset that “reserves” space for a single request to start writing. In other words, my offset tells the request where in the file to start writing.

4 Pseudocode

This is the core pseudocode for the program. Note that it’s pseudocode, not C (or Java or Python) code.

```
procedure main
    Declare string hostname
    Declare string port
    Declare N
    Declare option

    loop
        switch(option)
            case ‘N’:
                N = atoi(optarg)
                break;
            case ‘l’:
                LOGFILE = optarg
                break
            case ‘:’:
                fprintf(“option needs a value\n”)
                break
            case ‘?’:
                fprintf(“unknown option: %c\n”,optopt)
                break

    hostname = argv[argc - 2]
    strcpy(port, argv[argc - 1]

    if(LOGFILE)
        FD_LOG = open(LOGFILE, O_CREAT | O_RDWR | O_TRUNC, 0644)

    threads = new pthread_t[N]

    sem_init(&logMutex, 0, 1)
    sem_init(&full, 0, 1)
    sem_init(&empty, 0, N)

    loop
```

```
            pthread_create(&threads[i], NULL, start, NULL)


        Declare struct addrinfo addrs, hints
        hints.ai_family = AF_INET
        hints.ai_socktype = SOCK_STREAM
        getaddrinfo(hostname, port, &hints, &addrs)
        main_socket = socket(addrs.ai_family, addrs.ai_socktype,addrs.ai_protocol)
        enable ← 1
        setsockopt(main_socket, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable))
        bind(main_socket, addr.ai.addr, addr.ai_addrlen)
        listen(main_socket, 16)


        loop
            Declare client_socket = accept(main_socket, NULL, NULL);

            sem_wait(&empty)

            MYQUEUE.push( client_socket)

            sem_post(&full)

          return 0

    procedure start
        loop
            sem_wait(&full)

            Declare client socket = MYQUEUE.front()
            MYQUEUE.pop()

            sem_post(&empty)

            processOneRequest(client_socket)

    procedure processOnerequest
        Declare request
        Declare fileName
        Declare buffer

        memset(buffer, 0, 1024)
        recv(socket, buffer, 1024, 0)
        sscanf(buffer, “%s %s”, request, fileName)

        if fileName[0] == ‘/’
                memove(fileName, fileName + 1, strlen(fileName))
        if isValidName(fileName) == -1

                if(LOGFILE)
                        sem_wait(&logMutex)

                        localOffset ← GLOBAL_OFFSET
                        Declare message
                        sprintf(message, "FAIL: %s %s HTTP --- response 400\n=========\n",
request, fileName)
                        GLOBAL_OFFSET+= strlen(message)

                        sem_post(&logMutex)
                        localOffset+= printLog(message, localOffset)


                send(socket, “HTTP/1.1 400 Bad Request”, strlen(“HTTP/1.1 400 Bad
Request”),0)
                return
        if strcmp(request, “GET”) == 0
                processGet(fileName, socket)
        else if strcmp(request, “PUT”) == 0
                line ← strtok(buffer, “\r\n”)
                array[7]
                word[20]
                i ← 0
                j ← 0

                loop
                        array[i++] = line
                        line = strtok(NULL, “\r\n”)
                loop
                        if strstr(array[j], “Content-Length: ”) != NULL
                                break

                if array[j] != NULL
                        sscanf(array[j], “%*s, %s”, word)
                        i = atoi(word)
                else
                        i = -1
        processPut(fileName, socket, i)
    else
        if (LOGFILE)
            sem_wait(&logMutex)

            localOffset ← GLOBAL_OFFSET
            Declare message
            sprintf(message, "FAIL: %s %s HTTP --- response 400\n=========\n",
request, fileName)
            GLOBAL_OFFSET+= strlen(message)

            sem_post(&logMutex)
            localOffset+= printLog(message, localOffset)


        send(socket, “HTTP/1.1 400 Bad Request”, strlen(“HTTP/1.1 400 Bad

procedure isValidName
    Declare variable j
    Declare struct path_stat
    loop
        j+= 1
    if j != 27
        return -1
    loop
        c ← fileName[i]
        if isalpha(c)
            continue
        if isdigit(c)
            continue
        if c == ‘-’
            continue
        if c == ‘_’
            continue
        return -1

    return 0

procedure processGet
    fd ← open(fileName, O_RDONLY)
    buffer[32]

    if fd == -1
```

```
        if access(fileName, F_OK) == -1
            if (LOGFILE)
                sem_wait(&logMutex)

                localOffset ← GLOBAL_OFFSET
                Declare message
                sprintf(message, "FAIL: %s %s HTTP --- response
404\n=========\n", request, fileName)
            GLOBAL_OFFSET+= strlen(message)

                sem_post(&logMutex)
                localOffset+= printLog(message, localOffset)

            send(socket, "404 Not Found" strlen("404 Not Found"),0)
        else
            if (LOGFILE)
sem_wait(&logMutex)

                localOffset ← GLOBAL_OFFSET
                Declare message
                sprintf(message, "FAIL: %s %s HTTP --- response
403\n=========\n", request, fileName)
            GLOBAL_OFFSET+= strlen(message)

                sem_post(&logMutex)
                localOffset+= printLog(message, localOffset)


            send(socket, "403 Forbidden" strlen("403 Forbidden"),0)
        return

    Declare struct of type stat st

    if stat(fileName, &st) == 1)
        send(socket, "500 Internal Server Error" strlen("500 Internal Server Error"),0)

    size ← st.st_size

    if(LOGFILE)
        sem_wait(&logMutex)

        localOffset ← GLOBAL_OFFSET
        Declare message
        sprintf(message,”GET %s length 0\n=========\n", fileName)
            GLOBAL_OFFSET+= strlen(message)
            sem_post(&logMutex)

            localOffset+= printLog(message, localOffset)

    Declare char array str[1024]
    sprintf(str, "HTTP/1.1 200 OK \r\nContent-Length: %d\r\n\r\n", size);
    send(socket, str, strlen(str), 0);

    loop read(fd,buffer,1)
        send(socket, buffer, 1, 0)

    close(fd)

procedure processPut
    fd ← open(fileName, O_CREAT | O_RDWR | O_TRUNC, 0644)

    if fd == -1
        if (LOGFILE)
            sem_wait(&logMutex)

            localOffset ← GLOBAL_OFFSET
            Declare message
            sprintf(message, "FAIL: PUT %s HTTP --- response 403\n=========\n",
request, fileName)
            GLOBAL_OFFSET+= strlen(message)

            sem_post(&logMutex)

            localOffset+= printLog(message, localOffset)

        send(socket, “403 Forbidden” strlen(“403 Forbidden”),0)
        return
    i ← 0

    loop
        read (socket, buffer, 1)
        write(fd, buffer, 1)
        i+=1

    close(fd);
    if (LOGFILE)
        fd ← open(fileName, O_RDWR)
        Declare buffer_log
        Declare address
        Declare target
        Declare length

        length = sprintf(target + length, “PUT %s length %d\n”, fileName, size)

        sem_wait(&logMutex)

        localOffset = GLOBAL_OFFSET
        numLines ← size / 20 + (size % 20 != 0)
        GLOBAL_OFFSET += length + (9 * numLines) + (size * 3) + 9

        sem_post(&logMutex)

        loop
            length+= sprintf(target+length, “%08d “, address)

            loop
                length+= sprintf(target + length, “%02x “, buffer_log[j])

            length+= sprintf(target + length, “\n”)
            localOffset+= printLog(target, localOffset)
            length = 0
            address+=20

        close(fd)
        Declare buff
        strcpy(buff, “========\n”)
        localOffset+= printLog(buff, localOffset)

    send(socket, "HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n",
  strlen("HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n"), 0);
procedure printLog
    lineLength ← strlen(message)
    pwrite(FD_LOG, message, lineLength, localOffset)
    return lineLength
```