System Programming

Proxy lab: Writing a Caching Web Proxy

2017-18538 황선영

1. Goal of proxy lab

Web proxy란 web browser와 end server 사이에서 middleman으로 행동하는 program이다. 이번 lab에서는 직접 web object들을 cache하는 simple HTTP proxy 구현해본다.

first part에서는 accept incoming connection, read and parse requests, forward requests to web servers, read the servers' responses, forward those responses to the corresponding clients를 위한 proxy를 setup한다. second part에서는 multiple concurrent connections을 다루도록 proxy를 upgrade한다. third part에서는 구현한 proxy에 cache를 추가하여 web proxy를 완성한다.

2. Implementation

Thread-safe getbyhostname function을 구현하였다. writen, readn, readlineb, readnb 함수를 변형하였다. 그렇게 함으로서 caching하는 concurrent proxy program을 구현하였다.

① proxy.c

```c
int main(int argc, char *argv [])
{
        int cache_on = 1;
        int port;
        struct sockaddr_in clientaddr;
        int clientlen = sizeof(clientaddr);
        int listenfd;

        if(argc < 2) usage(argv[0]);

        if(argc >= 3) cache_on = (strcmp(argv[2],"off"));

        port = atoi(argv[1]);
        if(port <= 0 || port >= MAX_PORT_NUMBER)
        {
                printf("invalid port number: %d. Please input a port number from 1 to 65535.\n", port);
                exit(1);
        }

        cache_list = NULL;

        if(cache_on)
        {
                printf("cache is on.\n");
                cache_list = init_cache_list();
        }
        else printf("cache is off.\n");

        // ignore SIGPIPE
        Signal(SIGPIPE, SIG_IGN);

        listenfd = Open_listenfd(port);
        while(1)
        {
                pthread_t tid;
                int *connfdp = Malloc(sizeof(int));
                *connfdp = -1;
                *connfdp = Accept(listenfd, (SA *) &clientaddr, (socklen_t *)&clientlen);
                Pthread_create(&tid, NULL, (void *)proxy_process, (void *)connfdp);
        }

        delete_list(cache_list);
        return 0;
}
```

메인 함수이다. Port number가 valid한지 아닌지 살펴본다. invalid하다면 exit한다. 그리고 caching을 하여 caching proxy를 가능케 한다. 그 후 concurrent하게 실행하기 위하여 thread를 생성한다. 먼저 port번호에 해당하는 socket을 열어 bind해서 기다린다. 그리고 connfdp에 공간을 할당하여 listen하다가 client request를 받아온다. 그 후 thread를 생성한다.

```
void usage(char *str)
{
        printf("usage: %s <port> [cache policy]\n", str);
        printf("cache policy: 'off' to turn off caching\n");
        printf("cache policy: 'on' or other string to turn on caching\n");
        printf("cache policy: the default policy is to turn on caching\n");
        exit(1);
}
```

단순히 여러 정보를 출력하는 함수이다.

```
void proxy_process(int *arg)
{
        printf("pid: %u\n",(unsigned int)Pthread_self());
        Pthread_detach(Pthread_self());

        int client_fd = (int)(*arg);
        int server_fd = -1;
        char tmp_str[MAXBUF];
        char request_str[MAXBUF];
        char host[MAXBUF], port[MAXBUF], resource[MAXBUF];
        char cache_index[MAXBUF], content[MAX_OBJECT_SIZE];
        unsigned int len;
        int r_value = read_request(request_str, client_fd, host, port, cache_index, resource);

        printf("Read data from %s:%s\n",host,port);
        fflush(stdout);

        if(!r_value)
        {
                if(!read_node_content(cache_list, cache_index, content, &len))
                {
                        printf("cache hit!\n");
                        if(forward_content_to_client(client_fd, content, len) == -1)
                                fprintf(stderr, "forward content to client error.\n");
                        Close(client_fd);
                        return;
                }
                else
                {
                        int server_value = forward_to_server(host, port, &server_fd, request_str);
                        if(server_value == -1)
                        {
                                fprintf(stderr, "forward content to server error.\n");
                                strcpy(tmp_str, connect_fail_str);
                                Rio_writen(client_fd, tmp_str, strlen(connect_fail_str));
                        }
                        else if(server_value == -2)
                        {
                                fprintf(stderr, "forward content to server error(dns look up fail).\n");
                                strcpy(tmp_str, dns_fail_str);
                                Rio_writen(client_fd, tmp_str, strlen(dns_fail_str));
                        }
                        else
                        {
                                int f_value = read_and_forward_response(server_fd, client_fd, cache_index, content);
                                if(f_value == -1) fprintf(stderr, "forward content to client error.\n");
                                else if(f_value == -2 && cache_list) fprintf(stderr, "save content to cache error.\n");
                        }
                }
                close_fd(&client_fd, &server_fd);
        }
        return;
}
```

Proxy process가 동작하도록 하는 함수이다. 이곳에서 server value에 따라 connect fail과 dns fail을 처리해 준 후 read and forward response를 수행하도록 한다.

```
int read_request(char *str, int client_fd, char *host, char *port, char *cache_index, char *resource)
{
        char tmpstr[MAXBUF];
        char method[MAXBUF], protocol[MAXBUF], host_port[MAXBUF];
        char version[MAXBUF];

        rio_t rio_client;

        Rio_readinitb(&rio_client, client_fd);
        if(Rio_readlineb(&rio_client, tmpstr, MAXBUF) == -1) return -1;

        if(parse_request(tmpstr, method, protocol, host_port, resource, version) == -1) return -1;

        get_host_and_port(host_port, host, port);

        if(strstr(method, "GET"))
        {
                strcpy(str, method);
                strcat(str, " ");
                strcat(str, resource);
                strcat(str, " ");
                strcat(str, init_version);

                if(strlen(host))
                {
                        strcpy(tmpstr, "Host: ");
                        strcat(tmpstr, host);
                        strcat(tmpstr, ":");
                        strcat(tmpstr, port);
                        strcat(tmpstr, "\r\n");
                        strcat(str, tmpstr);
                }

                strcat(str, user_agent_hdr);
                strcat(str, accept_str);
                strcat(str, accept_encoding);
                strcat(str, connection);
                strcat(str, proxy_connection);

                while(Rio_readlineb(&rio_client, tmpstr, MAXBUF) > 0)
                {
                        if(!strcmp(tmpstr, "\r\n"))
                        {
                                strcat(str,"\r\n");
                                break;
                        }
                        else if(strstr(tmpstr, "User-Agent:") || strstr(tmpstr, "Accept:") ||
                                strstr(tmpstr, "Accept-Encoding:") || strstr(tmpstr, "Connection:") ||
                                strstr(tmpstr, "Proxy Connection:") || strstr(tmpstr, "Cookie:"))
                                continue;
                        else if(strstr(tmpstr, "Host:"))
                        {
                                if(!strlen(host))
                                {
                                        sscanf(tmpstr, "Host: %s", host_port);
                                        get_host_and_port(host_port, host, port);
                                        strcpy(tmpstr, "Host: ");
                                        strcat(tmpstr, host);
                                        strcat(tmpstr, ":");
                                        strcat(tmpstr, port);
                                        strcat(tmpstr, "\r\n");
                                        strcat(str, tmpstr);
                                }
                        }
                        else strcat(str, tmpstr);
                }

                strcpy(cache_index, host);
                strcat(cache_index, ":");
                strcat(cache_index, port);
                strcat(cache_index, resource);

                return 0;
        }

        return 1;
}
```

이 함수는 client로부터 오는 request를 읽는 함수이다. strcat은 문자열을 합치는 함수로, request를 parse하여 나의 str에 append한다. 그 다음 line을 하나 읽어 와서 여러 조건을 따져준 후 알맞은 형태로 읽어낸다. 최종적으로 cache_index에 host와 port, resource 정보를 담는다.

```c
int forward_to_server(char *host, char *port, int *server_fd, char *request_str)
{
        *server_fd = Open_clientfd(host, atoi(port));

        if(*server_fd < 0)
        {
                if(*server_fd == -1) return -1;
                else return -2;
        }
        if(Rio_writen(*server_fd, request_str, strlen(request_str)) == -1) return -1;

        return 0;
}

int forward_content_to_client(int client_fd, char *content, unsigned int len)
{
        if(Rio_writen(client_fd, content, len) == -1) return -1;
        return 0;
}
```

Forward to server 함수는 clientfd를 open하여 connection을 준비한다.

Forward content to client는 content를 쓴다. 이때 이상이 있는지 확인하는 작업까지 수행한다.

```c
int read_and_forward_response(int server_fd, int client_fd, char *cache_index, char *content)
{
        rio_t rio_server;
        char tmp_str[MAXBUF];
        unsigned int size = 0, len = 0, cache_size = 0;
        int valid_size = 1;

        content[0] = '\0';

        Rio_readinitb(&rio_server, server_fd);

        do{
                if(Rio_readlineb(&rio_server, tmp_str, MAXBUF) == -1) return -1;
                if(valid_size) valid_size = append(content, tmp_str, strlen(tmp_str), &cache_size);
                if (strstr(tmp_str, "Content-length")) sscanf(tmp_str, "Content-length: %d", &size);
                if (strstr(tmp_str, "Content-Length")) sscanf(tmp_str, "Content-Length: %d", &size);
                if (Rio_writen(client_fd, tmp_str, strlen(tmp_str)) == -1) return -1;

        }while(strcmp(tmp_str, "\r\n") != 0 && strlen(tmp_str));

        if(size)
        {
                while(size > MAXBUF)
                {
                        if((len = Rio_readnb(&rio_server, tmp_str, MAXBUF)) == -1) return -1;
                        if(valid_size) valid_size = append(content, tmp_str, MAXBUF, &cache_size);
                        if(Rio_writen(client_fd, tmp_str, len) == -1) return -1;
                        size -= MAXBUF;
                }

                if(size)
                {
                        if((len = Rio_readnb(&rio_server, tmp_str, size)) == -1) return -1;
                        if(valid_size) valid_size = append(content, tmp_str, size, &cache_size);
                        if (Rio_writen(client_fd, tmp_str, len) == -1) return -1;
                }
        }
        else
        {
                while((len = Rio_readnb(&rio_server, tmp_str, MAXLINE)) > 0)
                {
                        if(valid_size) valid_size = append(content, tmp_str, len, &cache_size);
                        if (Rio_writen(client_fd, tmp_str, len) == -1) return -1;
                }
        }

        if(valid_size)
        {
                if (insert_content_node(cache_list, cache_index, content, cache_size) == -1) return -2;
                printf("insert correct!\n");
        }

        return 0;
}
```

Read and forward response 함수는 말 그대로 response를 읽고 forward 해주는 함수이다. 먼저 server_fd로부터 read한 do 내의 operation을 수행한다. 이것은 client_fd 내에 정보를 기입하는 작업이다. 그 다음 buf의 사이즈에 따라 조금씩 달리해가며 정보를 읽은 후 쓰는 것을 반복한다. 만약 적당한 size가 들어갔다면 insert correct를 출력한다.

```c
int append(char *content, char *str, unsigned int len1, unsigned int *len2)
{
        if(len1 + (*len2) > MAX_OBJECT_SIZE) return 0;
        memcpy(content + (*len2), str, len1);
        *len2 += len1;
        return 1;
}

int parse_request(char *str, char *method, char *protocol, char *host_port, char *resource, char *version)
{
        char url[MAXBUF];

        if((!strstr(str, "/")) || !strlen(str)) return -1;
        strcpy(resource, "/");
        sscanf(str,"%s %s %s", method, url, version);

        if(strstr(url, "://")) sscanf(url, "%[^:]://%[^/]%s", protocol, host_port, resource);
        else sscanf(url, "%[^/]%s", host_port, resource);

        return 0;
}

void get_host_and_port(char *host_port, char *host, char *port)
{
        char *tmpstr = strstr(host_port,":");
        if(tmpstr)
        {
                *tmpstr = '\0';
                strcpy(port, tmpstr + 1);
        }
        else strcpy(port, "80");

        strcpy(host, host_port);
}

void close_fd(int *client_fd, int *server_fd)
{
        if(client_fd && *client_fd >=0) Close(*client_fd);

        if(server_fd && *server_fd >=0) Close(*server_fd);
}
```

Append 함수는 content에 정보를 추가해주는 함수이다.

Parse request는 들어온 request를 parse하는 함수이다. 여기엔 method, url, version, host port, resource 등의 정보가 포함되어 있다.

Get host and port는 host의 port number를 받는 함수이다.

Close_fd는 client file descriptor와 server file descriptor를 close하는 함수이다.

② cache.c

```
c_list *init_cache_list()
{
        c_list *list = Malloc(sizeof(*list));
        list->head = list->tail = NULL;
        list->lock = Malloc(sizeof(*(list->lock)));
        pthread_rwlock_init((list->lock), NULL);
        list->bytes_left = MAX_CACHE_SIZE;

        return list;
}

void init_node(c_node *node)
{
        if(node)
        {
                node->next = NULL;
                node->prev = NULL;
                node->length = 0;
        }
}

void set_node(c_node *node, char *index, unsigned int len)
{
        if(node)
        {
                node->index = Malloc(sizeof(char)*len);
                memcpy(node->index, index,len);
                node->length = len;
        }
}

void delete_node(c_node *node)
{
        if(node)
        {
                if(node->index) free(node->index);
                if(node->content) free(node->content);
                free(node);
        }
}
```

Init_cache_list는 cache list를 초기화하는 함수이다. c_list의 pointer를 반환한다.

Init_node는 node를 초기화하는 함수이다.

Set_node는 node를 set하는 함수이다. 이때 parameter를 받아 index와 길이를 직접 설정한다.

Delete_node는 node를 제거하는 함수이다. 이때 주의해야 할 것은 index, content, node 모두 free 해야 한다는 것이다.

```
void delete_list(c_list *list)
{
        if(list)
        {
                c_node *tmp = list->head;

                while(tmp)
                {
                        c_node *tmp2 = tmp;
                        tmp = tmp2->next;
                        delete_node(tmp2);
                }

                pthread_rwlock_destroy((list->lock));
                free(list->lock);
                free(list);
        }
}

c_node *search_node(c_list *list, char *index)
{
        if(list)
        {
                c_node *tmp = list->head;

                while(tmp)
                {
                        if(!strcmp(tmp->index, index)) return tmp;
                        tmp = tmp->next;
                }
        }
        return NULL;
}
```

Delete_list는 list를 제거하는 함수이다. 이 때 일반적인 list 제거와 다른 점은 pthread_rwlock_destroy 함수이다. 이 함수는 read-write lock object를 destroy함으로써 lock을 제거하여 list의 제거를 가능하게 한다.

Search_node함수는 index를 parameter로 받아 이를 토대로 search하는 함수이다.

```
void add_node(c_node *node, c_list *list)
{
        if(list)
        {
                if(node)
                {
                        while(list->bytes_left < node->length)
                        {
                                c_node *tmp_node = evict_list(list);
                                delete_node(tmp_node);
                        }
                        if(!list->tail)
                        {
                                list->head = list->tail = node;
                                list->bytes_left -= node->length;
                        }
                        else
                        {
                                list->tail->next = node;
                                node->prev = list->tail;
                                list->tail = node;
                                list->bytes_left -= node->length;
                        }
                }
        }
}

c_node *remove_node(char *index, c_list *list)
{
        if(list)
        {
                c_node *tmp = search_node(list, index);
                if(tmp)
                {
                        if(tmp == list->head) tmp = evict_list(list);
                        else
                        {
                                if(tmp->prev) tmp->prev->next = tmp->next;
                                if(tmp->next) tmp->next->prev = tmp->prev;
                                else list->tail = tmp->prev;
                                list->bytes_left += tmp->length;
                        }
                        tmp->prev = NULL;
                        tmp->next = NULL;
                }
                return tmp;
        }

        return NULL;
}
```

Add_node함수는 list에 node를 추가하는 함수이다. 삽입할 곳의 앞뒤 노드의 pointer를 수정한다.

Remove_node 함수는 index에 해당하는 node를 제거한 후 그 노드를 return하는 함수이다.

```
c_node *evict_list(c_list *list)
{
        if(list)
        {
                c_node *tmp = list->head;
                if(tmp)
                {
                        if(tmp->next) tmp->next->prev = NULL;
                        list->bytes_left += tmp->length;
                        if(list->head == list->tail) list->tail = NULL;
                        list->head = tmp->next;
                        return tmp;
                }
        }
        return NULL;
}

int read_node_content(c_list *list, char *index, char *content, unsigned int *len)
{
        if(!list) return -1;

        pthread_rwlock_rdlock((list->lock));

        c_node *tmp = search_node(list, index);

        if(!tmp)
        {
                pthread_rwlock_unlock((list->lock));
                return -1;
        }

        *len = tmp->length;
        memcpy(content, tmp->content,*len);

        pthread_rwlock_unlock((list->lock));

        pthread_rwlock_wrlock((list->lock));
        add_node(remove_node(index, list), list);
        pthread_rwlock_unlock((list->lock));

        return 0;
}
```

Evict_list는 node를 list로부터 쫓아내는 함수이다.

Read_node_content는 node의 content들을 읽어오는 함수이다. 먼저 node에 접근하기 위해서 pthread의 read-write lock을 풀어주어야 한다. 그 후 write lock을 건 다음 add하고 다시 lock을 풀어준다.

```
int insert_content_node(c_list *list, char *index, char *content, unsigned int len)
{
        if(!list) return -1;

        c_node *tmp = Malloc(sizeof(*tmp));
        init_node(tmp);
        set_node(tmp, index, len);

        if(!tmp) return -1;

        tmp->content = Malloc(sizeof(char)*len);
        memcpy(tmp->content, content,len);

        pthread_rwlock_wrlock((list->lock));
        add_node(tmp, list);
        pthread_rwlock_unlock((list->lock));
        return 0;
}
```

Insert_content_node 함수는 node에 content를 삽입하는 함수이다. 먼저 write lock을 걸어준 후 node를 더한 다음 다시 lock을 풀어준다.

3. Conclusion

이번 lab을 통해 직접 web proxy를 구현하여 그 특성을 알게 되었다. 이론으로만 보았던 HTTP에 대해 basic HTTP operation을 익히고 적용해 봄으로써 좀 더 깊게 이해하였다. cache하는 부분이

까다로웠지만, 구현하고 나니 뿌듯했다. 직접 웹 페이지에 접속해 내가 구현한 proxy를 시험해보면서 debugging을 했는데, 그래서인지 다른 lab보다 조금 더 debugging이 수월했다.