Design Document
Billy Kwong
Bqkwong
CSE 130 Fall19

ASGN3 Design

Goal

The goal for Assignment 3 is to modify the HTTP server that you already implemented to have one additional feature: caching.

Assumptions

I'm assuming that server will only support GET and PUT methods of HTTP protocol, therefore it will generate an error that server does not support this HTTP header if it will receive anything else. In addition, it will only run on Linux/Unix environment and use executable file path as the root directory of server.

Design

My strategy of cache implementation is to wait for client GET request. When server receive request it queries cached whether resources exist in its cache or not. If entry found, it sends the page from cache. In case of query fail, server load the page from disk, after sending page to client server also save it to in cache for later use.

Pseudocode

STARTUP:

```
// Declaring and initializing variables at startup of main function
int server_fd, client_socket;
struct sockaddr_in address;
int addrlen = sizeof(address);
int port = DEFULT_PORT;

// Setting Root path of web server
char *ROOT = getenv("PWD");

// Declaring an arroy of type struct cache_item and a variable int _pointer to implement cache functionality
cache_item_t _cache_item[4];
int _pointer;

// Parsing and processing command line arguments
for (size_t idx = 1; idx < argc && argv[idx][0] == '-'; idx++)
{
        if(argv[idx][0] == '-')
        {
```

```
                switch (argv[idx][1])
                {
                        case 'c':
                        logging = 1;
                        Break;
                        case 'p':
                        port = atoi(argv[idx+1]);
                        Break;
                        Default:
                        cout << "Usage: ./httpserver -p <port> or ./httpserver <port> -c\n" << endl;
                        exit(EXIT_FAILURE);
                }
        }
}
```

CACHE IMPLEMENTATION:

```
// Structure that will hold the entry of chache
typedef struct cache_item
{
        int init;
        string resource_key;
        string resource_data;
} cache_item_t;
```

```
// This is a function that takes an argument of resource key if resource found it means resource exists in cache therefore no need to load from disk.
string* cache_find(string resource_key)
{
        for(int i=0; i<4 ; i++)
        {
                if( _cache_item[i].init == 1 )
                {
                        if( _cache_item[i].resource_key.compare(resource_key) == 0 )
                        {
                                return &_cache_item[i].resource_data;
                        }
                }
        }
return NULL;
}
```

// This is the main piece of code that will implement the actual logic of cache for simplicity and efficiency I have chosen FIFO page replacement policy.

```
void cache_add(string resource_key, string resource_data)
{
	for(int i=0; i<4 ; i++)
	{
		if( _cache_item[i].init == 0 )
		{
			_cache_item[i].init = 1;
			_cache_item[i].resource_key.clear();
			_cache_item[i].resource_data.clear();
			_cache_item[i].resource_key.append(resource_key);
			_cache_item[i].resource_data.append(resource_data);
			return;
		}
	}

	if(_pointer==4)
	_pointer=0;

	_cache_item[_pointer].resource_key.clear();
	_cache_item[_pointer].resource_data.clear();
	_cache_item[_pointer].resource_key.append(resource_key);
	_cache_item[_pointer].resource_data.append(resource_data);

	_pointer++;
}
```

// An array _cache_item declared at the startup of main()

```
cache_item_t _cache_item[4];
```

// If page is not found in cache then it should be cached here is the logic and code to adding page into cache

```
string data_to_cached;
send(client_socket, "HTTP/1.0 200 OK\n\n", 17, 0);
while ((bytes_read = read(fd, data_to_send, BUFFER_SIZE)) > 0)
{
	write(client_socket, data_to_send, bytes_read);
	data_to_cached.append(data_to_send);
}
cache_add(path, data_to_cached);
```

```
// If page found in cache here is the logic and code to sending page from cache without loading it from disk
if(cache_find(path) != NULL)
{
	log("GET Request, Sending page %s\n", path );

	string* cached = cache_find(path);
	send(client_socket, "HTTP/1.0 200 OK\n\n", 17, 0);
	write(client_socket, cached->c_str(), cached->length());
}
```

LOGGING:

```
// A variable is declared at global scope
int logging = 0;

// Logging can be enable by providing -c argument on command line this flag will change the value of logging variable to 1
for (size_t idx = 1; idx < argc && argv[idx][0] == '-'; idx++)
{
	switch (argv[idx][1])
	{
		case 'c':
		logging = 1;
		Break;
		//…
	}
}
// log() will print the logging out on the screen
void log(const char *format, ...)
{
	if(logging)
	{
		va_list args;

		va_start(args, format);
		vprintf(format, args);
		va_end(args);
	}
}
```

```
// Usage of log function
log( "GET Request for home page.\n" );
log( "GET %s length %d [was in cache]\n", path, cached->length());
log( "GET Request, Sending page %s\n", path );
```