# PERSONAL PORTFOLIO 2 – N8947791

## Code Snippets for authentication

This code is written in C.

The code reads from the file the usernames and passwords and authenticates the user.

The rest of the code is in the folder.

### Open a file and read

// Fill the array of structs with users.

void fill\_array\_of\_registered\_users( void ) {

// Declare file pointer.

FILE \* fp;

// Open file for reading.

fp = fopen( "Authentication.txt", "r" );

if( fp == NULL )

{

perror( "Error opening file" );

exit( 1 );

}

// Store the username and passwords of the users.

read\_username\_and\_password( fp );

// Close the file after use.

fclose( fp );

}

### Authenticate user

// Authenticates the user. Returns true if the user is registered and false if the user is not registered.

bool authenticate\_user( char \* given\_username, long given\_password ) {

for ( int k = 0; k < NUM\_OF\_USERS; k++ ) {

if ( strcmp( registered\_users[ k ].username, given\_username ) == 0 ) {

if ( registered\_users[ k ].password == given\_password ) {

return true;

}

}

}

return false;

}

### Structure used for storing username and passwords

// Declare users struct.

typedef struct users {

// The username of the client.

char username[ 100 ];

// The password of the client.

unsigned long password;

} users\_t;

## Code Snippets for Client – Server connection

This code is written in C. Referred from CAB403 Systems Programming.

The code establishes a connection between the server and client.

The rest of the code is in the folder.

### SERVER SIDE CONNECTION USING SOCKETS AND THREADPOOL

int main( int argc, char \*\* argv )

{

// Signal handler for Ctrl+C

signal( SIGINT, signal\_handler );

// Thread id's, each of type int.

int thr\_id[ NUM\_HANDLER\_THREADS ];

// Array of thread structures.

pthread\_t p\_threads[ NUM\_HANDLER\_THREADS ];

// Used for wasting time.

struct timespec delay;

// Assigning the address to a global variable.

pt = &p\_threads[ 0 ];

// Create the request-handling threads.

for ( int i=0; i < NUM\_HANDLER\_THREADS; i++ ) {

thr\_id[ i ] = i;

// Creating 10 threads with thread structure, function that starts with each thread is handle\_requests\_loop and we are passing the thread\_id as parameter.

pthread\_create( &p\_threads[ i ], NULL, handle\_requests\_loop, (void\*)&thr\_id[ i ] );

user\_fd[ i ] = -1;

}

// SERVER - MULTITHREADING USING THREADPOOL

// 1. Create a threadpool of 10 threads and each thread calling handle\_requests\_loop method.

// 2. Set up the server socket, listening and wait for a client.

// 3. After client is connected to the server, we call the add\_request function with the client number.

// Listen on socket\_fd.

int socket\_fd;

// My address ( Server's ) information

struct sockaddr\_in my\_addr;

socklen\_t size;

int yes = 1;

// Port number

int port\_number;

// Get the port number if specified

if ( argc != 2 ) {

port\_number = DEFAULT\_PORT\_NUMBER;

} else if ( argc == 2 ) {

// Port number is given to the server.

port\_number = atoi( argv[ 1 ] );

}

// Generate the socket

if ( ( socket\_fd = socket( AF\_INET, SOCK\_STREAM, 0 ) ) == -1 ) {

// Error occured while generating a socket

perror( "Socket generation failure.\n" );

exit( 1 );

}

if ( setsockopt( socket\_fd, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof( int ) ) == -1 ) {

perror( "Error in setsockopt.\n" );

exit( 1 );

}

memset( &my\_addr, 0, sizeof( my\_addr ) );

// Generate the end point

// Host byte order

my\_addr.sin\_family = AF\_INET;

// Short network byte order

my\_addr.sin\_port = htons( port\_number );

// Auto-fill with my IP

my\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket to the end point

if ( ( bind( socket\_fd, (struct sockaddr \*)&my\_addr, sizeof( struct sockaddr ) ) ) == -1 ) {

perror( "Binding failure\n" );

exit( 1 );

}

printf( "\nServer: Filling array of clients..." );

// Initialise array list

fill\_array\_of\_registered\_users( );

// Initialise mutex lock.

if ( pthread\_mutex\_init( &leaderboard\_lock, NULL ) != 0 ) {

printf("\nMutex initialisation failed.\n");

close( socket\_fd );

exit( 1 );

}

printf( "\nServer: Mutex lock initialized..." );

// Start listening

if ( ( listen( socket\_fd, BACKLOG ) ) == -1 ){

perror( "Listening failure\n" );

exit( 1 );

}

// Listening successful.

printf( "\nServer: Server starts listening..." );

// Assign value to global server socket.

server\_fd = socket\_fd;

// This variable represents the request number.

static int counter = 0;

// This represents the thread number.

// Repeat: Accept, send and close the connection

// For every accepted connection, use a seperate process or thread to

// serve the client.

// Main accept( ) loop

while( 1 ) {

// Connector's address ( Client's ) information

struct sockaddr\_in their\_addr;

// New connection on client\_fd.

int client\_fd;

memset( &their\_addr,0,sizeof( their\_addr ) );

size = sizeof( struct sockaddr\_in );

if ( ( client\_fd = accept( socket\_fd, (struct sockaddr \*)&their\_addr, &size ) ) ==-1 ) {

perror( "Accept failure.\n" );

exit( 1 );

}

int connections = 0;

for ( int i = 0; i < NUM\_HANDLER\_THREADS; i++ ) {

if ( user\_fd[ i ] != -1 ) {

connections++;

}

}

if ( connections < NUM\_HANDLER\_THREADS ) {

// Got connection.

printf( "\nServer : Got connection from client %s\n", inet\_ntoa( their\_addr.sin\_addr ) );

add\_request( counter, client\_fd, &request\_mutex, &got\_request );

// Increase the number of requests.

counter++;

} else {

close( client\_fd );

}

} // end of while loop.

// Close the server socket.

close( socket\_fd );

return 0;

} //End of main

### CLIENT SIDE USING SOCKETS

int main( int argc, char \*\* argv )

{

// Connector's address information.

struct sockaddr\_in their\_addr;

struct hostent \*he;

int socket\_fd, numbytes;

char buff[ MAXSIZE ];

if ( argc != 3 ) {

fprintf(stderr, "Please enter the client hostname and port number\n" );

exit(1);

}

// Get the host information.

if ( ( he = gethostbyname( argv[ 1 ] ) ) == NULL ) {

herror("Cannot get hostname.");

exit(1);

}

if ( ( socket\_fd = socket( AF\_INET, SOCK\_STREAM, 0 ) ) == -1 ) {

perror( "Socket Failure.\n" );

exit(1);

}

memset( &their\_addr, 0, sizeof( their\_addr ) );

their\_addr.sin\_family = AF\_INET;

their\_addr.sin\_port = htons( atoi( argv[ 2 ] ) ) ;

their\_addr.sin\_addr = \*( (struct in\_addr \*)he->h\_addr) ;

if ( connect( socket\_fd, (struct sockaddr \*)&their\_addr, sizeof( struct sockaddr ) ) < 0 ) {

perror( "Connection Failure.\n" );

close( socket\_fd );

exit( 1 );

}

}

}

// Close the socket.

close(socket\_fd);

return 0;

}//End of main

## USER STORIES, REVIEW LETTER

I contributed to user stories regarding the encryption of data and storing the hash-value of passwords with salt rather than just plain-text passwords. Also, prioritized the user stories along with the other team members. The introduction, conclusion and some questions on the second business review letter were done by me. I also have suggested the development team that it would be better if there were message boxes or dialog boxes to confirm the removal of a property from the property list. I have took part in discussions regarding the environment for coding and the optimal method to store the files that are uploaded by the user. Also took part in discussions involving negotiations with the client team about the release plans, sprints and user stories.

## PROJECT ANALYSIS

I have used the Waterfall model (Requirements, Design, Implementation and Verification) to break down the project development process and have used a tree diagram to illustrate the different components essential for the application to function. I have included the tree diagram in the folder. This model was also used in the first release.

## PROJECT IMPLEMENTATION AND TESTING

The application was tested using Selenium by me and is attached with this document.