مستند سازی پروژه مبانی برنامه سازی

پانیذ حلواچی

فاز اول پروژه (بخش کلاینت)

- تابع ()init ورودی نمیگیرد و خروجی آن نیز int اســت. درباره ی آنکه دقیقا به چه صـورت عمل میکند حقیقتا من هم نمیدانم و با توجه به آنکه ما دراینباره نخوانده ایم کدهایش را از زیپی که در اختیارمان قرار داده شـــده بود کپی کردم.به طور کلی درباره ی این تابع میتوان گفت تنظیمات اولیه برای اتصال به سرور را فراهم میکند.
- تابع ()client ورودی نمیگیرد و خروجی آن از نوع int اســـت و client_socket را return میکند. این تابع برای ارتباط کلاینت با سرور سوکتی میسازد و آن را return میکند.
- تابع ()firstpart دو ورودی struct sockaddr_in servaddr و char *token را میگیرد و خروجی آن void اســت. این تابع ابتدا منوی اولیه را که شـامل بخش های login و register اســت printf میکند و ســپس گزینه ی انتخابی کاربر را scanf میکند و از کاربر نام کاربری و رمز ورودش را دریافت میکند حال با توجه به گزینه ی انتخابی پیام مشـخص شـده ی آن گزینه(سـاختاری که در داک ذکر شـده بود) را به سـرور میفرسـتد و سـپس پاسـخ سـرور به درخواسـت کلاینت را دریافت میکند که تایپ آن یا successful هسـت یا error .اگر ارور بود در کنسـول پیام ارور را چاپ میکند (من در کدم نوشته ام که در کنسول تنها error چاپ شود ولی اکنون پس از فاز دو که سرورم علت خطا را نیز برای کلاینت میفرستد بهتر است در آن بخش مینوشـتم که بخش content پاسخ سرور را چاپ کند باتوجه به آنکه قبلا فاز اول را ارسـال کردم متاسـفانه امکان تغییر آن نیسـت)و دوباره به همین تابع return میکند. اگر عملیات سـرور برای register موفقیت آمیز بوده باشد اینبار کلاینت پیام درخواست برای login را با آن نام کاربری و رمز عبور ارسال میکند و برای بخش login پاسخ سـرور با دو تایپ error و authtoken است اگر error باشـد که مانند قبل عمل میکند ولی اگر authtoken بـاشـــد از بخش content آن رشـــتـه ی توکن را ذخیره میکند وســـپس بـه تـابع ()return secondpart میکند.
- تـابع ()secondpart دو ورودی struct sockaddr_in servaddr و char *token را میگیرد و خروجی آن void اسـت. این تابع منو ای شـامل گزینه های create channel و join channel و logout را بر روی کنسـول چـاپ مـیـکـنـد. اگـر کـاربـر گـزیـنـه ی create channel را انـتـخـاب کـنـد نـام کـانـال را از کـاربـر بـا scanf("%[^\n]",channelname) میگیرد.(درواقع تا زمانی میگیرد که کاربر enter بزند) ســـپس درخواســـت متناسب را با فرمت ذکر شده در داک به سرور ارسال میکند و پاسخ سرور را دریافت میکند در صورتی که عمل خواسته شـده از سـرور موفقیت آمیز بوده باشـد به ()return thirdpart میکند در غیر اینصـورت وجود خطا در عملیات را به کاربر نشـــان داده و به خود تابع return میکند. برای گزینه ی join channel نیز چنین فرایندی طی میشـــود. اگر گزینه ی انتخابی logout باشـد پیام متناسـب را به سـرور میفرسـتد و سـپس به ()return firstpart میکند.(در این قسمت پیام خطا از سرور فرض در نظر نگرفته ام زیرا در الگوریتمم میتوان از صحت توکن اطمینان داشت.
- تابع ()thirdpart دو ورودی struct sockaddr_in servaddr و char *token را میگیرد و خروجی آن void است. این تابع منو ای شامل گزینه های send message و refresh و channel members و leave را بر روی کنسـول چاپ میکند.با انتخاب گزینه ی send message از کاربر پیامش را میگیرد و سـپس پیام متناسـب را به سـرور فرسـتاده و به خود تابع return میکند. با انتخاب گزینه ی refresh و ارسـال پیام به سـرور در پاسـخ پیام های ارسـال شده در کانال به کلاینت فرستاده میشود سپس در این بخش از کد

```
cJSON* ms = cJSON_GetObjectItemCaseSensitive(serverans3, "content");
if(cJSON_IsArray(ms))
{
   cJSON *m = NULL;
   cJSON_ArrayForEach(m, ms)
   {
    strcpy(contentS, cJSON_GetObjectItem(m,"sender")->valuestring);
    strcpy(contentC, cJSON_GetObjectItem(m,"content")->valuestring);
       printf("%s:",contentS);
       printf("%s\n",contentC);
   }
}
```

بخش content پاسخ سرور را (به صورت جیسون است) دریافت میکنیم و در صورتی که آرایه باشد پوینتری به عضو آن در نظر گرفته شـــده و ســـپس با اســـتفاده از cJSON_GetObjectItem آیتمی را که نامش sender اســـت در contentS, کپی کرده و آن را بر کنسول print میکنیم و به همین ترتیب برای آیتمی به نام content نیز همین عمل صـورت میگیرد و این فرایند تا زمانی که پوینتر به تمام اعضـای آرایه اشـاره کند تکرار میشـود. برای بخش channel members نیز پاسخ سرور را بدین صورت print میکنیم با این تفاوت که در این بخش به جای دو آیتم یک آیتم در هر عضو است. در هر دو بخش channel members و refresh تابع به خودش بازگردانده میشود. برای گزینه ی آخر که leave است پیام متناسب را به سرور فرستاده سپس به return secondpart() میکنیم.

فاز دو پروژه (بخش سرور)

- تابع ()init تنظیمات اولیه مربوط به اتصال کلاینت و سرور را انجام میدهد.
- تابع ()serversetting ابتدا سوکتی برای سرور میسازد سپس سوکت را bind میکند (راستش خودم هم دقیقا نمیدانم این تابع ()bind چه کار میکند) سپس منتظر کلاینت میماند.
- تابع ()clientsetting سوکت کلاینت را به سرور متصل میکند و راه ارتباط سرور و کلاینت ساخته میشود و میتوانند پیام هایشان را بهم منتقل کنند.
- تابع های compare فایل مربوطه را باز میکنند و خط به خط میخوانند(fscanf برای بخش های مختلف هر خط) و آن را با رشته ی داده شده در ورودی این تابع مقایسه میکنند (()strcmp) خروجی این توابع اعدادی همچون 0 و 1 و -1 هستند که بدین صورت تعریف میشود--> در تابع compare tokens اگر چنین توکنی موجود باشد حاصل تابع 0 و در غیر اینصورت 1 خواهد بود. در تابع compare usernames اگر username موجود و password مربوط با آن نیز با password داده شده در ورودی مطابقت داشته باشد حاصل تابع 1 و در صورت که password مطابقت نداشته باشد حاصل تابع 0 و اگر چنین username ای موجود نباشد حاصل تابع -1 خواهد بود. تابع compare channelnames نیز همچون token خروجی اش تعریف میشود.
- تابع ()sendmessage دو رشته ورودی میگیرد که آنان را به ترتیب valuestring بخش با نام type و بخش با نام content در نظر میگیرد.سپس با استفاده از cJSON_Print cJSON حاصل را به رشته تبدیل میکند و آن را به کلاینت میفرستد. خروجی این تابع void است.
- تابع ()register user پیام ارسال شده از کلاینت را میگیرد و با توجه به ساختار پیام کلاینت که از قبل مشخص شده است بخش هایی همچون username و password را با sscanf ذخیره میکند و در صورتی‌که username تکراری نباشد آن را در فایل userlist اضافه میکند و پیام عملیات موفق به کلاینت میفرستد. خروجی این تابغ void است.
- تابع ()login پیام ارسال شده از کلاینت را میگیرد و با توجه به ساختار پیام کلاینت که از قبل مشخص شده است بخش هایی همچون username و password را با sscanf ذخیره میکند و در صورتی که username وجود داشته باشد و password داده شده نیز با password ثبت شده در فایل مطابقت داشته باشد توکن میسازد و آن را به کلاینت میفرستد.در غیراینصورت (هرکدام از شرایط ذکر شده در بالا برقرار نباشد) پیام خطا میفرستد.
- تابع ()set token با استفاده از تابع rand به 12 عضو آرایه ی توکن مقداردهی میکند(به شکلی که برابر عدد شود) و سپس به 12 عضو بعدی نیز مقداردهی میکند به شکلی که برابر حرف شود. خروجی این تابع void است.ورودی هم نمیگیرد.(آرایه ی توکن global در نظر گرفته شده است.
- تابع ()channel members پیام ارسال شده از کلاینت را میگیرد سپس صحت توکن داده شده در پیام کلاینت را بررسی میکند اگر اشتباه بود پیام خطا به کلاینت ارسال میکند حال اگر صحیح بود اینبار چک میکند که آیا struct clients با این توکن در کانالی عضو هست یا نه و مانند قبل عمل میکند . اگر در کانالی عضو بود در struct clients های ساخته شده بخش channelname را بررسی میکند و اگر با نام کانال توکن یکسان بود آن را به آرایه ی cjson

که نامش content است اضافه میکند سپس با استفاده از root cJSON_Print حاصل را به رشته تبدیل کرده و به کلاینت میفرستد. خروجی این تابع void است.

- تابع ()refresh پیام ارسال شده از کلاینت را میگیرد سپس صحت توکن داده شده در پیام کلاینت را بررسی میکند اگر اشتباه بود پیام خطا به کلاینت ارسال میکند حال اگر صحیح بود اینبار چک میکند که آیا struct clients با این توکن در کانالی عضو هست یا نه و مانند قبل عمل میکند. حال فایل messages را باز میکند و خط به خط میخواند درصورتی که نام کانال ابتدای هر خط با نام کانال توکن تطابق داشت پیامی شامل نام ارسال کننده پیام در گروه و پیام ارسال شده در گروه میفرستد(با توجه به آنکه قبلا در متن به جزییات چگونگی تشکیل و ارسال پیام به کلاینت پرداخته ام در اینجا دیگر نمیپردازم)
- تابع sendfromuser به عنوان ورودی بخش message موجود در پیام ارسال شده از کلاینت را میگیرد و مجددا مانند توابع بالا صحت توکن و عضو بودن آن در کانال را چک میکند اگر هر دو مورد تایید بودند پیام کاربر را با نام کانال و نام خود کاربر و پیامش در فایل messages ذخیره میکنیم.
- تابع remove first part of string پیام ارسال شده ی کاربر را دریافت میکند و سپس قسمت "send " و توکن را حذف میکند و در واقع درآرایه داده شده به عنوان ورودی تنها بخش message میماند. خروجی این تابع void است.
- تابع ()join channel به عنوان ورودی پیام کلاینت را دریافت میکند و بخش channelname و توکن آن را ذخیره میکند و وجود آنان را بررسی میکند سپس درصورتی که هر دو موجود بودند. Index آرایه struct clients ای که توکنش برابر با توکن ذخیره شده باشد را می یابدو channelname را در بخش channelname این عضو از آرایه ی استراکت کپی میکند و سپس پیام عملیات موفق برای کلاینت ارسال میکند. خروجی این تابع void است.
- تابع ()create channel به عنوان ورودی پیام کلاینت را دریافت میکند و بخش channelname و توکن آن را ذخیره میکند و وجود توکن را بررسی میکند سپس درصورتی که موجود بود فایل channelnames را باز کرده و channelname را به آن اضافه میکند. خروجی این تابع void است.
- تابع ()leave به عنوان ورودی پیام کلاینت را دریافت میکند و بخش توکن آن را ذخیره میکند و وجود توکن را بررسی میکند. سپس بخش channelname عضوی از آرایه struct clients را که توکنش با توکن ذخیره شده یکسان است میابیم اگر تهی باشد پیام خطا میدهیم که کاربر در کانالی عضو نیست در غیر اینصورت مقدار آن را تهی میکنیم.
- تابع ()logout به عنوان ورودی پیام کلاینت را دریافت میکند و بخش توکن آن را ذخیره میکند و وجود توکن را بررسی میکند. اگر وجود داشت فایل tokens را باز کرده و خطی را که شامل این توکن باشد را حذف میکنیم.(توضیح الگوریتم حذف خط در فایل را در صورت نیاز در تحویل حضوری توضیح خواهم داد.) خروجی این تابع void است.

توابع cJSON به کار برده شده در فاز دوم پروژه :

1. cJSON_AddItemToArray()

این تابع به عنوان ورودی پوینتری به یک CJSON از نوع آرایه اســـت و پوینتری به item ای که میخواهم به آرایه بیفزاییم میگیرد.خروجی این تابع void هست و حاصل آن افزودن آیتم داده شده به آرایه ی داده شده است.

2. cJSON_CreateObject()

این تابع ورودی نمیگیرد و خروجی آن * CJSON اســـت . با اســـتفاده از این تابع میتوانیم متغیری از CJSON (که یک struct است) با نوع object بسازیم.

3. cJSON_CreateArray()

این تابع ورودی نمیگیرد و خروجی آن * CJSON اســـت . با اســـتفاده از این تابع میتوانیم متغیری از CJSON (که یک struct است) با نوع object بسازیم.

4. cJSON_AddItemToObject()

این تابع به عنوان ورودی پوینتری به یک CJSON از نوع object اســـت و پوینتری به item ای که میخواهم به آرایه بیفزاییم میگیرد.خروجی این تابع void هست و حاصل آن افزودن آیتم داده شده به object داده شده است.

5. cJSON_CreateString()

این تابع ورودی رشـــته میگیرد و خروجی آن * CJSON اســـت . با اســـتفاده از این تابع میتوانیم متغیری از CJSON (که یک struct است) با نوع string بسازیم. که عضو مقدار رشته استراکت آن برابر با مقدار رشته ورودی میشود.

6. cJSON_Delete()

این تابع ورودی پوینتری به یک CJSON میگیرد و خروجی آن void اســـت. این تابع برای از بین بردن اســـتراکت cjson ای که پوینتر به آن اشاره دارد استفاده میشود.

7. cJSON_Print()

این تابع ورودی یک struct cJSON میگیرد و ســـپس آن را به رشـــته ای با فرمت تعریف شـــده برای cJSON تبدیل میکند و خروجی آن پوینتری است که به این رشته اشاره میکند.

توابع cJSON به کار برده شده در فاز اول پروژه :

8. cJSON_Parse()

دقیقا برعکس cJSON_Print است. به این صورت که یک پوینتر که به رشته (با فرمت مخصوص cJSON) اشاره دارد به عنوان ورودی میگیرد و یک struct cJSON درست میکند و با آن رشته بهش مقداردهی میکند سپس به عنوان خروجی پوینتری به استراکت ساخته شده برمیگرداند.

9. cJSON_GetObjectItem()

در ابتدا prototype آن به صورت زیر است:

```
(cJSON *) cJSON_GetObjectItem(const cJSON * const object, const char * const string);
```

ورودی اول در واقع به struct cJSON ای اشاره میکند که item در آن قرار دارد. String در واقع item->string که نام item هست میباشد. این تابع item با آن نام را در object می یابد سپس پوینتری که به item اشاره دارد به عنوان خروجی برمیگرداند.

10. cJSON_IsArray()

به عنوان ورودی پوینتری به cJSON دریافت میکند و اگر نوع آن آرایه باشد مقدار true و در غیر اینصورت مقدار false را برمیگرداند.(نوع خروجی آن bool است)

11. cJSON_GetStringValue()

به عنوان ورودی پوینتری به cJSON (از نوع string) دریافت میکند و رشته ی ذخیره شده در آن را به عنوان خروجی میدهد(*char).در واقع item->valuestring را به عنوان خروجی میدهد.