Computer Engineering & Informatics Department
University of Patras

Network Lab

Academic Year 2016-2017

E-mail: marinou@ceid.upatras.gr

1 Key - Value Store

This query was asked to store <key, value> pairs and then be able to retrieve them based on the key, key. So we created two functions, get and put that implement this exactly, with get to retrieve data as there are and put it to store them in a similar position - if the key exists, update to the corresponding cell and store it there, and if there is not then stored in a new location. The table created for storing these two values is in a store structure and is the keyvalue, two-dimensional with the first position of the cell being the key and the second the value. The functions are as follows

```
41
42  //-----------------------------------------------------------------------------------------------------------------------
43  char *get(char *key)
44  {
45    /** Returns the value associated with the key, or NULL
46    * if the key is not in the store. **/
47
48    int i=0;        //counter
49    int flag=0;     //if not exists the key
50    char p[50];     // value whitch returns if we didn't find the key
51    int pos=0;      //variable to store the position
52
53
54     for( i = 0; i<=netlab.next ; i++)     //while we have positions to check, we continue
55     {
56        if ((strcmp(netlab.keyvalue[i][0], key)) == 0)   //if the comparations return 0 then we have found the key
57        {
58            return netlab.keyvalue[i][1];              //return the value of the key, the second value from the cell
59        }
60     }
61
62     return NULL;   //return nothing
63  }
64
65  //-----------------------------------------------------------------------------------------------------------------------
66
```

```
65  //-----------------------------------------------------------------------------------------------------------------------
66
67  void put(char *key, char *value)
68  {
69    /* Puts the <key,value> pair into the store. If the key already existed, its previous
70    * value is discarded and replaced by the new one. */
71
72    int x;
73    int flag=0;
74
75        for (x=0;x<netlab.next ;x++) // check all the positions
76        {
77            if (strcmp(netlab.keyvalue[x][0], key)== 0) //if you find the key update
78            {
79
80                strcpy(netlab.keyvalue[x][1], value);
81                strcpy(netlab.keyvalue[x][0], key);
82                flag=1;
83                return;
84            }
85        }
86
87       if (flag == 0) //if the value doesn't exist, then we store it in the array
88       {
89            netlab.next++;   //put it in the next empty cell
90            strcpy(netlab.keyvalue[netlab.next][1], value);
91            strcpy(netlab.keyvalue[netlab.next][0], key);
92            return;
93       }
94  }
95  //-----------------------------------------------------------------------------------------------------------------------
96
```

2 Application Protocol (APPLICATION PROTOCOL)

The communication between server and client must be set based on the protocol entered in the speech when when one client connects to the server, it sends one or more commands. A command starts with a byte which defines the type of function: get value means 103 and Put 112, the words initials. Any other code will be considered as error and then this connection terminates the connection. The implementation of the above is thus:

From the client side

```
61
62      char prot;
63      for( i = 3; i < argc; i++ )                                 //Here we send the right input arguments
64      {
65          strcpy(str1,"get");
66          strcpy(str2,"put");
67          if ( strcmp(argv[i],"get" ) == 0)                       //if argv[i]=get then send g
68          {
69              prot = 103;
70              writen(sockfd,&prot,1);                             //send g
71
72              if (argv[i+1]==NULL) break;                         //if there isn't key, then stop the connection
73              writen(sockfd,argv[i+1],strlen(argv[i+1])+1);       //else, write it to the socket, send it to the server
74              recv(sockfd, buffer, 1, 0);                         //take the message from the server
75
76              if (buffer[0] == 'n'){                               //if the answer from server is n->110 ascii (didn't found)
77                  printf("\n");
78                  i +=1;
79                  continue;                                        //keep searching untill you find it
80              } else if (buffer[0] == 'f'){                           //if we found  it, then we recieve f->102 ascii
81                      while((er = read(sockfd, &buffer, 1))>0 && buffer[0]!='\0') //and we take it from the server
82                      printf("%c", buffer[0]);
83              }
84
85              printf("\n");
86              i++;
87              continue;
88          }
89
90          if ( strcmp(argv[i],"put" ) == 0) {                     //if argv[i]=put
91              prot = 112;
92              writen(sockfd,&prot,1);                             //send p
93
94              if (argv[i+1]==NULL) break;                         //if we haven't found key or value stop the connection
95              if (argv[i+2]==NULL) break;
96
97              writen(sockfd,argv[i+1],strlen(argv[i+1])+1);       //else, take both of them
98              writen(sockfd,argv[i+2],strlen(argv[i+2])+1);
99              i+=2;
100             continue;
101         }
102     }
```

From the server side

```
134     while(1) {
135        len= sizeof(struct sockaddr_in);
136        newsockfd = accept(sockfd, (struct sockaddr *)&serv_addr, &len);
137        pid=fork();
138
139
140       while(1) {
141          recv(newsockfd, bfr, 1,0);
142
143        if (bfr[0] == 'g')      //if the inptu is then
144        {
145            i=0;
146            while( recv(newsockfd, &ch, 1,0)>0 && (ch!='\0')){ //while we recieve one by one character from client
147                bfr[i++] = ch;                //store them
148            }
149            bfr[i] = '\0';                    //store 0 before the arg
150
151          bzero(my_buffer, 0);
152          my_buffer = get(bfr);               //store the result from get function
153
154          if ( my_buffer == NULL )            //if we took null
155            writen(newsockfd, &n_found, 1);   //return n
156          else
157          {
158            writen(newsockfd, &found, 1);     //else f for found
159            writen(newsockfd, my_buffer, strlen(my_buffer)+1); //return the result from get
160          }
161
```

```
162       //-------------------------------------------------------------------------------------------------
163        } else if (bfr[0] == 'p')           //if the inptu is then
164        {
165           i=0;
166           while( recv(newsockfd, &ch, 1,0)>0 && (ch!='\0')){   //while we recieve arg from client
167              bfr[i++] = ch;        //store them
168           }
169          bfr[i] = '\0';            //store 0 before the arg
170
171          i=0;
172          while( recv(newsockfd, &ch, 1,0)>0 && (ch!='\0')){  //while we recieve arg from client
173            bfr2[i++] = ch;         //store them
174          }
175          bfr2[i] = '\0';           //store 0 before the arg
176          put(bfr, bfr2);           //send key,value to put function
177
178        } else
179          break;                    //if we recieved no p or g then stop the connection
180     }
```

3 Client Implementation - 4 Server Implementation

The implementation is contained within the client.c and server {1.2.3.4} .c files in which the writen function that has been delivered to the lessons has been used.

The client and server implementation has been implemented according to the following logic: