Vasabilab System Programming Notes:

(Jan 8, 2012 -kasidit chanchio)

Note 1. The Compilation and Installation of libmemcached.

I have just made this library work on my notebook last night. My system is an HP pavilion g4 with Intel core i-3 2330 with 4GB Ram running ubuntu 11.10. The following describes how I did it. Hope they are useful for you.

In order to use C client SDK to store and retrieve data from memcached or membase servers (see couchbase.org for server installation), first you need to download **libmemcached-1.0.2.tar.gz** from Brian Aker's http://libmemcached.org.

Next, extract and untar the tarball and change directory to the libmemcached-1.0.2 directory to compile and install the library.

```
$ gzip -d libmemcached-1.0.2.tar.gz
$ tar xvf libmemcached-1.0.2.tar
$ cd libmemcached-1.0.2
```

Run ./configure to create an appropriate Makefile.

```
$ ./configure
```

You need to apt-get g++ if you don't already have it.

```
$ apt-get install g++
```

Then, issues the make commands

```
$ make
$ make test
$ sudo make install
```

Since libmemcached is the shared library, make sure the LD_LIBRARY_PATH environment variable is set. I append the following line to the .bashrc file in my home directory.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

(In case you run into compilation and linking problems, you may try to add compilation flags like -fPIC to the CXXFLAG variable in the Makefile. Also, when you performed "make test", it is okay that some tests failed. You may focus on the testing on features that fit your work.)

Note 2. Using libmemcached

I found a couple of example programs from mysql website

http://dev.mysql.com/doc/mysql-ha-scalability/en/ha-memcached-interfaces-libmemcached.html

and used them to test the library. The first program reads
Example 1:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *key= "keystring";
  char *value= "keyvalue";
  //memcached_server_st *memcached_servers_parse (char *server_strings);
  memc= memcached_create(NULL);
  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);
  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));
  rc= memcached_set(memc, key, strlen(key), value, strlen(value), (time_t)0, (uint32_t)0);
  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Key stored successfully\n");
  else
    fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));
  return 0;
}
```

The program was saved on a "mypgm.c" file. It was compiled with a command

```
$ gcc -o mypgm mypgm.c -lmemcached
```

Then, we run

```
$ ./mypgm
```

and get

```
Added server successfully
Key stored successfully
```

Example 2:
Another program from the above website read

```
#include <stdio.h>
#include <sstring.h>
#include <unistd.h>
#include <libmemcached/memcached.h>
int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *keys[]= {"huey", "dewey", "louie"};
  size_t key_length[3];
  char *values[]= {"red", "blue", "green"};
  size_t value_length[3];
  unsigned int x;
  uint32_t flags;
  char return_key[MEMCACHED_MAX_KEY];
  size_t return_key_length;
  char *return_value;
  size_t return_value_length;
  memc= memcached_create(NULL);
  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);
  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));
  for(x= 0; x < 3; x++)
    {
      key_length[x] = strlen(keys[x]);
      value_length[x] = strlen(values[x]);
    rc= memcached_set(memc, keys[x], key_length[x], values[x],
                      value_length[x], (time_t)0, (uint32_t)0);
    if (rc == MEMCACHED_SUCCESS)
      fprintf(stderr,"Key %s stored successfully\n",keys[x]);
    else
      fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));
   }
  rc= memcached_mget(memc, keys, key_length, 3);
  if (rc == MEMCACHED_SUCCESS)
   {
    while ((return_value= memcached_fetch(memc, return_key, &return_key_length,
                                          &return_value_length, &flags, &rc)) != NULL)
     {
      if (rc == MEMCACHED_SUCCESS)
       {
        fprintf(stderr,"Key %s returned %s\n",return_key, return_value);
       }
     }
   }
  return 0;
}
```

After compile and run the program, we get

```
Added server successfully
Key huey stored successfully
Key dewey stored successfully
Key louie stored successfully
Key huey returned red
Key dewey returned blue
Key louie returned green
```

Now, we can study and use these functions to develop applications.

Note 3. An example program using TCP and BINARY protocol

This is another example program I wrote. Since the default behavior of libmemcache uses UDP protocol and take limited data size (around 1600 characters). We have to reconfigure its behavior to take binary data and expand send and receive buffers.

Example 3:

```
/*
 This is an example C client program using libmemcached to store data
 on membase server. The data in this program is binary and the connections
 mde to membase servers are TCP. This program is adapted from an example
 program on MySQL website.

 Author: Kasidit Chanchio
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

#define  NUM_KEYS  10        // number of keys used in this example

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;

  uint64_t *keys[NUM_KEYS];
  size_t key_length[NUM_KEYS];

  uint8_t *values[NUM_KEYS];
  size_t value_length[NUM_KEYS];
  unsigned int x;
  uint32_t flags;

  uint64_t return_key;
  size_t return_key_length;

  uint8_t *return_value;
  size_t return_value_length;

  int i, j;

  const char *config_string= "--BINARY-PROTOCOL --TCP-NODELAY --SOCKET-SEND-SIZE=8092";
  memc= memcached(config_string, strlen(config_string));

  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);

  if (rc == MEMCACHED_SUCCESS)
          printf("Added server successfully\n");
  else
          printf("Couldn't add server: %s\n",memcached_strerror(memc, rc));

  for(i = 0; i< 3; i++){
    keys[i] = (uint64_t *) malloc(sizeof(uint64_t));
    key_length[i] = sizeof(uint64_t);
    *keys[i] = (uint64_t) (1000 + i);
    printf(" key = %lu len %d\n", *keys[i], key_length[i]);
    values[i] = (uint8_t *) malloc(sizeof(uint8_t)*4096);
    for (j = 0; j < 4096; j++){
          *(*(values+i)+j) = i;
    }
    for (j = 0; j < 10; j++){
        printf("v = %u\n", *(values[i]+j));
    }
    value_length[i] = sizeof(uint8_t) * 4096;

    rc= memcached_set(memc, (const char *) keys[i], key_length[i],
         (const char *) values[i], value_length[i], (time_t)0, (uint32_t)0);
    if (rc == MEMCACHED_SUCCESS)
        printf("Key %lu stored successfully\n",*keys[i]);
    else
        printf("Couldn't store key: %s\n",memcached_strerror(memc, rc));
  }

  rc= memcached_mget(memc, (const char * const *) keys, key_length, 3);

  if (rc == MEMCACHED_SUCCESS) {
      while ((return_value= (uint8_t *) memcached_fetch(memc, &return_key, &return_key_length,
                                      &return_value_length, &flags, &rc)) != NULL) {
         if (rc == MEMCACHED_SUCCESS) {
              printf(" key %d len %d ret v len %d value: \n", return_key, return_key_length,
return_value_length);
                     for(i = 0; i < 10; i++){
                              printf(" %d ", *(return_value+i));
                     }
                     printf("\n");
         }
      }
  }
  return 0;

}
```