Module utils

Data Types

server_address()

```
server_address() = {string(), non_neg_integer()}
```

Function Index

binary_convert_key/2	Toma un termino, lo convierte a binario agregandole el identificador como prefijo, y lo retorna junto con su longitud (considerando el prefijo).
binary_convert_value/1	Toma un termino, lo convierte a binario, y lo retorna junto con su longitud.
close_server_sockets/1	Cierra la conexion con cada uno de los sockets de la lista de sockets objetivo.
create_message/1	Dado un codigo de operacion genera el binario en el formato especificado por el protocolo que se enviara como mensaje al servidor.
create_message/3	Dado un codigo de operacion, la longitud de la clave, y la clave, genera el binario en el formato especificado por el protocolo que se enviara como mensaje al servidor.
create_message/5	Dado un codigo de operacion, la longitud de la clave, la clave, la longitud del valor, y el valor, genera el binario en el formato especificado por el protocolo que se enviara como mensaje al servidor.
create_servers_info/1	Dado un listado de sockets y su direccion y puerto, nos devuelve una lista con la inforamcion que iremos llevando de los sockets.
create_sockets/1	Toma una lista de pares IP y puerto, y crea un socket para cada uno de ellos.
create_stats_message/2	Crea un string con informacion las stats de los servidores a los que el cliente se encuentra conectado.
create_status_message/3	Crea un string con informacion de las stats que el cliente lleva localmente.
delete_info/2	Dado un socket de un servidor y uan lista de informacion de servidores, elimina de la lista aquella informacion de servidor que coincida con el socket pasado.
get_server/2	Dada una clave y un ServersTable con la informacion de los servidores, obtiene el bucket (servidor) asociado a la clave a traves de la funcion hash.
modify_key_counter/3	Dada una tabla de servidores, un socket y un valor K, intenta modificar el contador del socket pasado (de la lista de de servidores) por dicha cantidad.
rebalance_servers/2	Dado un ServersTable con la informacion de los servidores corriendo y la informacion de un servidor objetivo, reemplaza cada aparicion del socket del server objetivo en el ServersTable por un socket elegido aleatoriamente de entre los demas sockets de los servers disponibles, rebalanceando asi la carga entre todos los servidores.
recv_bytes/2	Lee N bytes del socket objetivo.
replace_sockets/3	Dada una lista de sockets, un socket objetivo, y una lista de la informacion de los servidores disponibles, reemplaza cada aparicion del socket objetivo en la lista por un socket de uno de los servers disponibles, elegido de manera aleatoria.

Function Details

binary_convert_key/2

```
binary_convert_key(Term::term(), Identifier::binary()) -> {binary(), non_neg_integer()}
```

`Term`: el termino a convertir.

returns: Una tupla {BinaryTerm, BinaryLength} con el binario equivalente al termino y su longitud.

Toma un termino, lo convierte a binario agregandole el identificador como prefijo, y lo retorna junto con su longitud (considerando el prefijo).

binary_convert_value/1

```
binary_convert_value(Term::term()) -> {binary(), non_neg_integer()}
```

`Term`: el termino a convertir.

returns: Una tupla {BinaryTerm, BinaryLength} con el binario equivalente al termino y su longitud.

Toma un termino, lo convierte a binario, y lo retorna junto con su longitud.

close_server_sockets/1

```
close_server_sockets(ServersInfo::[server_info()]) -> ok
```

`ServersInfo`: La lista con la informacion de los servidores con los cuales queremos terminar la conexion.

returns: El atom ok.

Cierra la conexion con cada uno de los sockets de la lista de sockets objetivo.

create_message/1

```
create_message(Operation::integer()) -> binary()
```

`Operation`: codigo de la operacion, un entero que debe caber en 8 bits.

returns: El binario en el formato correcto.

Dado un codigo de operacion genera el binario en el formato especificado por el protocolo que se enviara como mensaje al servidor.

create_message/3

```
create_message(Operation::integer(), KeyLength::integer(), Key::binary()) -> binary()
```

`Operation`: codigo de la operacion, un entero que debe caber en 8 bits.
`KeyLength`: la longitud de la clave del mensaje, un entero que debe caber en 32 bits.
`Key`: la clave en formato binario, con longitud KeyLength.

returns: El binario en el formato correcto.

Dado un codigo de operacion, la longitud de la clave, y la clave, genera el binario en el formato especificado por el protocolo que se enviara como mensaje al servidor.

create_message/5

```
create_message(Operation::integer(), KeyLength::integer(), Key::binary(), ValueLength::integer(), Value::binary()) ->
binary()
```

`Operation`: codigo de la operacion, un entero que debe caber en 8 bits.
`KeyLength`: la longitud de la clave del mensaje, un entero que debe caber en 32 bits.
`Key`: la clave en formato binario, con longitud KeyLength.
`ValueLength`: la longitud del valor del mensaje, un entero que debe caber en 32 bits.
`Value`: el valor en formato binario, con longitud ValueLength.

returns: El binario en el formato correcto.

Dado un codigo de operacion, la longitud de la clave, la clave, la longitud del valor, y el valor, genera el binario en el formato especificado por el protocolo que se enviara como mensaje al servidor.

create_servers_info/1

```
create_servers_info(Sockets::[{gen_tcp:socket(), server_address()}]) -> [server_info()]
```

`Sockets`: listado de sockets para los cuales queremos crear su lista de informacion.

returns: Una lista con la informacion asociada a los sockets.

Dado un listado de sockets y su direccion y puerto, nos devuelve una lista con la inforamcion que iremos llevando de los sockets.

create_sockets/1

```
create_sockets(ServerList::[server_address()]) -> allCreated | {notCreated, {string(), non_neg_integer()}}
```

`ServerList`: Una lista de pares (string(), non_neg_integer()) representando pares (IP, puerto) de servidores memcached.

returns: Una lista de gen_tcp:socket() si la creacion es exitosa, o {notCreated, (string(), non_neg_integer())} si se produjo un error, donde la segunda componente de la tupla es el par (IP, puerto) del servidor que no logro conectarse.

Toma una lista de pares IP y puerto, y crea un socket para cada uno de ellos. Si todos se crean correctamente, se devuelve la lista con los objetos gen_tcp:Socket asociados a cada uno.

create_stats_message/2

```
create_stats_message(ServersInfo::[server_info()], InitialNumServer::non_neg_integer()) -> string()
```

`ServersInfo`: Lista de servidores a los cuales el cliente se encuentra conectado actualmente.
`InitialNumServer`: cantidad de servidores que habia cuando el cliente se creo.

returns: String que representa el mensaje de stats solicitado.

Crea un string con informacion las stats de los servidores a los que el cliente se encuentra conectado. Si un servidor se desconecta en el momento que pide stats, no hace rebalanceo de cargas (solo lo hacen PUT, DEL, GET).

create_status_message/3

```
create_status_message(TotalKeys::non_neg_integer(), ServersInfo::[server_info()],
InitialNumServer::non_neg_integer()) -> string()
```

`TotalKeys`: numero total de claves.
`InitialNumServer`: cantidad de servidores que habia cuando el cliente se creo.

returns: String que representa el mensaje de status solicitado.

Crea un string con informacion de las stats que el cliente lleva localmente. La funcion sigue contando las claves de un server cuando este se desconecto y no se hizo un rebalanceo de carga.

delete_info/2

```
delete_info(ServerSocket::gen_tcp:socket(), ServersInfo::[server_info()]) -> [server_info()]
```

`ServerSocket`: El socket de una informacion de servidor que queremos eliminar.
`ServersInfo`: La lista de informacion de servidores de la cual queremos eliminar aquella que tenga el socket indicado.

returns: Una nueva lista de informacion de servidores sin las que tenian el socket pasado.

Dado un socket de un servidor y uan lista de informacion de servidores, elimina de la lista aquella informacion de servidor que coincida con el socket pasado.

get_server/2

```
get_server(Key::term(), ServersTable::servers_table()) -> gen_tcp:socket()
```

`Key`: La clave para la cual queremos determinar el servidor.
`ServersTable`: La estructura con la informacion de los servidores.

returns: El socket del servidor asociado a la clave.

Dada una clave y un ServersTable con la informacion de los servidores, obtiene el bucket (servidor) asociado a la clave a traves de la funcion hash.

modify_key_counter/3

```
modify_key_counter(ServersTable::servers_table(), Socket::gen_tcp:socket(), K::non_neg_integer()) -> servers_table()
```

`ServersTable`: tabla de servidores para la cual queremos modificar el contador de uno de sus sockets.
`Socket`: es aquel al cual le queremos modificar el contador.
`K`: es el valor por el cual queremos modificar el contador.

returns: Una nuevas tablas de servidores, con el contador asociado al socket indicado modificado.

Dada una tabla de servidores, un socket y un valor K, intenta modificar el contador del socket pasado (de la lista de de servidores) por dicha cantidad.

rebalance_servers/2

```
rebalance_servers(ServersTable::servers_table(), ServerSocket::gen_tcp:socket()) -> servers_table()
```

`ServersTable`: La estructura con la informacion de los servidores memcached en ejecucion.

returns: Un nuevo ServersTable con el rebalanceo de servidores aplicado.

Dado un ServersTable con la informacion de los servidores corriendo y la informacion de un servidor objetivo, reemplaza cada aparicion del socket del server objetivo en el ServersTable por un socket elegido aleatoriamente de entre los demas sockets de los servers disponibles, rebalanceando asi la carga entre todos los servidores.

recv_bytes/2

```
recv_bytes(Socket::gen_tcp:socket(), N::non_neg_integer()) -> {ok, binary()} | {error, term()}
```

`Socket`: objeto Socket de gen_tcp del cual queremos leer.
`N`: entero no negativo que representa la cantidad de bytes a leer.

returns: {ok, binary()} si se leyo correctamente, {error, _} si no.

Lee N bytes del socket objetivo.

replace_sockets/3

```
replace_sockets(Sockets::[gen_tcp:socket()], ServerSocket::gen_tcp:socket(), ServersInfo::[server_info()]) ->
[gen_tcp:socket()]
```

`Sockets`: Lista de sockets sobre la que se hara el reemplazo.
`ServerSocket`: El servidor a reemplazar en cada aparicion.
`ServersInfo`: Lista con informacion de todos los servidores disponibles.

returns: Una lista donde cada aparicion del socket objetivo fue reemplazada por un socket aleatorio de los servers disponibles.

Dada una lista de sockets, un socket objetivo, y una lista de la informacion de los servidores disponibles, reemplaza cada aparicion del socket objetivo en la lista por un socket de uno de los servers disponibles, elegido de manera aleatoria.
