TP7 - KEY VALUE STORE
(4)

"signals+asynchronous I/O+sockets"

Server

At this point your application will not be a simple main but a server that handles incoming connections for the known operations you already implemented in the previous TPs. Specifically the server will support listening of **NON BLOCKING** connections that will provide the known commands and will reply with the known replies or your key value store (as usual).

Preliminaries

Get acquainted with

- File descriptors and sets: Read the wikipedia entry https://en.wikipedia.org/wiki/File_descriptor and remember some important things.
    - **IT IS AN INDEX**,
    - **FD_SETSIZE** preprocessor define in "select.h"
    - How to use select(...) - very important, check : https://www.gnu.org/software/libc/manual/html_node/Waiting-for-I_002fO.html

- Signals: In this TP you will have to handle the signal SIGINT (ctrl+c) in order to safely shut down your application-server that could possibly have dynamic allocated structure instances and active file descriptors that need to close. Things to look into are:
    - sigaction struct
    - function pointers for callbacks (sigaction rationale)

  MAKE A TEST APPLICATION THAT HANDLES KNOWN SIGNALS AND PRINTS OUT THEIR TYPE before you start tinkering your server with this.

- Sockets: This has been stressed out enough in the previous TP. You will be provided with a simple python script for the client functionality.

In Detail

1. Prepare your server socket - wrap around **fcntl(..)** to update the file descriptor of the socket with **O_NONBLOCK**.
2. Check the provided source for the FDState struct and its members.
    a. **buffer** is used for data in/out
    b. **end** and **cur**
    c. **status** can be set from the predefined values (READ/WRITE/RESULT etc)
3. You will maintain an array of pointers that point to the FDState datatype. The array will be static with the length predefined as FD_SETSIZE. The catch here is that a file descriptor returned from incoming connections with **accept(..)** is a simple integer value and it will signify the i-th **INDEX** (that is the i-th pointer in the array) that will be used to manually allocate space for a FDState instance for a specific connection status and input/output data.

inside your run_server function...

4. Initialize your file descriptor sets for read/write (hint: **FD_ZERO** macro)
5. Maintain an infinite “while” loop that blocks with **select(..)** and accepts incoming connections
6. After select:
    a. **accept(...)** the incoming connection,
    b. allocate state entry for read,
    c. serve the key value request,
    d. change the state entry for write
    e. reply back to the sender

Things to clean up with signal callback function

1. Any allocated state instances.
2. Close the leftover file descriptors.
3. Close the store.