Chart Software Documentation

Created by: Péter Tibor Madai

Source compilation

The compiler to use is `gcc` with the `-fopenmp` switch for the multi threading feature.

The executable has to be named `chart` to be able to run, so the `-o chart` switch should also be used for specifying the output filename.

Example compilation command: `gcc chart.c -fopenmp -o chart`

Compilation tested with `gcc 11.3.0` — if you have problems with compilation, you should try with this exact version.

System Requirements

Minimum:

- CPU: Intel Pentium 1GHz
- RAM: 128MB DDR1
- GPU: Doesn't matter
- Network: 128kb/s
- Operating System: UNIX based

Recommended:

- CPU: Multi thread CPU
- RAM: 1GB DDR3
- GPU: Still doesn't matter

- Network: 512 kb/s
- Operating System: Ubuntu 22.04 LTS.

User Manual

What is `chart`?

`chart` is a simple program that simulates taking measurements, and is able to send this simulated data to a server instance of `chart` via File or UDP Socket. When new measurement data is passed to the server, it will create a visualization of the measurements in the form of a monochrome bitmap image.

Arguments for running `chart` after compilation.

Command: `./chart ...`

- `-help` — display help message. duh.
- `-version` — display version information.
- `-send` — default | set communication mode to send. Can be used with `-file` or `-socket`
- `-receive` — set communication mode to receive. Can be used with `-file` or `-socket`
- `-file` — default | set communication mode to file. Can be used with `-send` or `-receive`
- `-socket` — set communication mode to socket. Can be used with `-send` or `-receive`

Note that any wrong use-case of the arguments will result in an error message stating `ERROR: Invalid arguments`

`./chart -send -file` (file client)

In the send via file mode, the program expects another instance of the program running in `-receive -file` mode on the same computer. The data will be passed to the receiving end through a file on the hard drive.

`./chart -receive -file` (file server)

In receive via file mode, the program is waiting for another instance of the program running in `-send -file` mode on the same computer. The data will be passed to the receiving end through a file on the hard drive. After receiving the data, the server will proceed to wait for new data transactions.

`./chart -send -socket` (network client)

In send via socket mode, the program will send data through the network using UDP protocol to the receiving computer (server), which must be running in `-receive -socket` mode.

`./chart -receive -file` (network server)

In receive via socket mode, the program is waiting for data to be received from the computer that takes the measurements (client). The client must be running in `-send -socket` mode. After receiving the data from the client, the server will proceed to wait for other connections from a client.

Return values of `chart`

- `0` — Everything went A OK.
- `1` — An error in using the parameters of the program.
- `2` — Error in creating the socket for UDP transmission.
- `3` — Sending or Binding error when using socket transmission.

- `4` — Receiving error when using socket transmission.
- `5` — Handshake or Size check missmatch error when using socket transmission.
- `6` — Error that is raised when using the program in Send via File mode and there is no receiving program running.
- `7` — Error that is raised when using Send via Socket mode and the server is not responding to the initial handshake within one second.

Function descriptions

- `check_file_name` — checks if the file name of the executable is `“chart"`. If it is not chart it exits with `return code 1`.
- `array_contains` — searches for the `word` parameter in the `*arr[]` array parameter which is of the size of the `size` parameter.
- `print_version_exit` — takes an `exit code` as an argument, then displays version information and finally `exits` with the `provided error code`
- `print_help_exit` — takes an `exit code` as an argument, then displays help information and finally `exits` with the `provided error code`
- `get_char_array_size` — takes a `char array` as argument and counts the length of the array based on the `\0` character and finally returns the length value.
- `array_has_duplicate` — searches for a duplicate in the provided array. If there is a duplicate it returns with 1, otherwise 0.
- `check_args` — A procedure that handles the checking of arguments. If an argument is incorrect, it will generate an error message.
- `get_working_state` — This function processes arguments and decides whether the program will run in sending, receiving, file or socket mode. The function returns this information in a struct.
- `max` — returns the bigger integer out of the two arguments.
- `print_int_arr` — Prints out the contents of given integer array.

- `Measurement` — Generates simulated measurement data and writes this data in the location of the pointer provided in the arguments, finally returns the size of the generated array.
- `write_int` — Takes a char pointer and an integer as argument, then writes the integer to the location of the char pointer and also the 3 remaining bytes.
- `int_pow` — A basic pow function which works with integers.
- `BMPcreator` — Creates a bitmap image of a chart representing the values of the input array.
- `FindPID` — Searches for another running instance of the program. If it finds one it will return the PID of that program, otherwise it will return with value -1.
- `SendViaFile` — Sends the provided Values through a file.
- `ReceiveViaFile` — Processes data that is sent through a file.
- `SignalHandler` — Handles signals. :D
- `SendViaSocket` — Sends provided array of values through UDP protocol to the receiving server.
- `ReceiveViaSocket` — Receives arrays of values through UDP protocol from the sending client and processes these values.