GDB Cheatsheet

Basic Commands

- To run gdb on a binary, run `gdb <path/to/your/executable>`
- To pass arguments to your binary, run `run [arguments' list]` in the gdb shell
- To run gdb attached to XV6, run `make qemu-gdb` in terminal 1 and run `gdb-multiarch` in terminal 2 in the same directory
- To pass arguments to a binary in XV6 run it from the XV6 shell with the corresponding arguments
- `c` lets you continue the programs execution, either after attaching to the process or after a breakpoint
- `file <[user,kernel]/file>` lets you load the specific file you want to debug
- `b [line_number,function_name]` lets you set a breakpoint in the currently loaded file
- `p [symbol_name]` lets you peek at the current value of a symbol (e.g. what value a variable currently holds)
- `backtrace` lets you examine the call stack
- `Ctrl + c` lets you interrupt the current execution and brings you back into the GDB shell
- `q` is used to quit gdb
- `help` can be used to list information about all available commands

Breakpoints

- Conditional breakpoints: If you only want to break if a condition is met, you can do this with `b [location] if [condition]`
- `condition` can be simple comparisons (e.g. `var == 20`) or more complex, f. eks. `strcmp(var, "hello") == 0`
- `watch` lets you break on a read/write to a variable by using `watch var` (breaks on write) and `rwatch var` (breaks on read)
- `watch` breakpoints can also have conditions: `watch [var] if [condition]`
- `info b` lists all of the set breakpoints
- `disable [breakpoint_number]` lets you temporarily disable a breakopint
- `enable [breakpoint_number]` lets you re-enable a disabled breakpoint
- `delete` lets you remove all breakpoints, while `delete [breakpoint_number]` lets you remove a specific breakpoint
- `ignore <breakpoint_number> <number>` lets you ignore the breakpoint with the corresponding `breakpoint_number` for the next `<number>` of hits

Stepping Through Execution

- `n` lets you execute the next line in code
- `s` lets you step inside the execution of the current line of code (e.g. inside a function call)
- `finish` lets the current function complete and then breaks again (until it hits a return)
- `c` lets you continue execution

Pretty Printing

- `print var->attr` prints the value of attr of a struct `var`
- `p/format [symbol]` lets you format the output of your print
- The following formats exist:
  - `x` shows the value in hexadecimal
  - `t` shows the value in binary
  - `c` shows the value as an integer *and* its character representation (useful for `uint8`)
  - `f` tries to represent it as a floating point number
  - `s` tries to print it as a string
- `display [symbol]` lets you print a value of a symbol anytime the execution pauses on a breakpoint
- `undisplay` removes the display again

Checking Hardware State

- `info registers` shows all the registers current state
- `info registers [register_name]` shows the current value of the specified register

Checking Thread State

Why do we care about threads? Because XV6 runs on multiple cores and thus ends up having a kernel thread per core. Sometimes, when debugging, multiple threads hit a breakpoint, and you might want to only make progress on one of them.

- `info threads` lists all threads and their current frame
- `thread <thread-id>` lets you switch to a specific thread while debugging
- `thread apply <thread-id> <cmd>` lets you apply a specific command to a specific thread
- `thread apply all <cmd>` lets you apply a specific command to all threads
  e.g. `thread apply all bt` gives you the stack traces of all the threads
- `thread find <regex>` lets you find a thread matching the regular expression
- `thread name <name>` lets you name a thread for easier identification

Control Thread Stepping

Sometimes it can be useful to control which thread is allowed to make progress under certain circumstances. Fot that, GDB offers the feature called `scheduler-locking`.

- `set scheduler-locking [mode]` allows you to change the current scheduler locking mode of GDB. GDB supports the following modes:
  - `off`: No locking, which means a thread may preempt at any time and all threads make progress when stepping through execution or continuing execution
  - `on`: Full locking, which means only the current thread is allowed to progress, both when stepping and continuing.
  - `step`: Only locked for stepping commands (`next`, `step`), but not when continuing execution