# Technical Documentation

## Motivation:

This project is inspired from the masters thesis [1] of Erik van der Kouwe. As part of his project, he had made changes in qemu to run deterministically. However the changes done are in a much older version of qemu (in 2009) and much of Qemu's architecture has changed ever since. This project therefore focused on porting many of his changes that are still relevant and finding and eliminating other sources of indeterminism that are present in the more recent qemu code that I worked on.

## Change Highlights:

The major changes in the qemu code for making it deterministic are as follows:
1. Implementation of the notion of virtual time in QEMU based on the number of guest instructions executed.
2. Elimination of dependency on host for timers (by avoiding the use of SIGALRM), thereby making guest time completely independent from the host.
3. Synchronization between the main IO thread and the cpu thread.

## Design:

1. Qemu follows the dynamic binary translation approach for translating guest instructions into host code. As part of this, the function **disas_insn** in **target-i386/translate.c** converts guest instructions one by one. A set of such translated instructions is then aggregated to form a basic block. Such a basic block of instructions is fetched and executed every time the qemu cpu thread needs to execute the corresponding guest instructions.
So to keep track of the number of instructions executed, we need a counter and a mechanism to ensure that this counter is incremented after every instruction is executed. Thus we have a variable named **ticks_virtual_machine** in **qemu-timer.c** to serve as a such a counter. We also have a *helper function* in **target-i386/op_helper.c** which increments this counter every time it is invoked.
Now to ensure that this helper function is invoked on every instruction execution, we generate a translation for it invocation in **disas_insn** along with the translation for each guest instruction. This helper function thus becomes part of the basic block which contains the translated instruction and is executed after every guest instruction execution. This is how virtual time is maintained.

2. In the original qemu architecture, there are timer queues associated with each type of clock that qemu maintains. These queues contain the time at which the cpu thread needs to be interrupted and related events be processed by the IO thread. Based on the earliest timer deadline across all such queues, a host timer is set up to send a SIGALRM signal once the earliest deadline is reached.
To eliminate this dependency on host, we introduce a variable named

**ticks_virtual_machine_next_timer** which keeps track of the next earliest deadline when an interrupt is expected. Whenever any of the timer queue changes (a new deadline is added or an old one deleted) this variable is updated accordingly. Since we have no host timer now, we manually check after every instruction execution (in the helper function) if **ticks_virtual_machine** has exceeded **ticks_virtual_machine_next_timer.** If it has then we send an interrupt and the cpu thread exits from the guest execution loop.

3. Synchronization between the IO thread the CPU thread is established by the use of additional condition variables which ensure that after the cpu thread stops executing the guest, it always stops for the IO thread to proceed before going back to executing guest instructions again.

## References:

1. Erik van der Kouwe. Porting the QEMU virtualization software to MINIX 3. From the website http://www.few.vu.nl/~vdkouwe/doc/msc-thesis-cs-erik-van-der-kouwe.pdf as of December 1, 2013.