Erlang Timers and Clocks

1. **Clock Processes**

   Please write an Erlang module `clock` that defines a process that manages a time (counted in ticks) as its state which increases in a configurable speed.

   a) `clock` processes shall handle these messages:

   - {`set, Value`}
     The internal time is adjusted to `Value`.
   - {`get, Pid`}
     The internal time is send in the form {`clock, Value`} to the process `Pid`.
   - `pause`
     Incrementing the internal time is paused until further notice. Subsequent `get` invocations will send identical values.
   - `resume`
     The process will continue incrementing the internal time where it left.
   - `stop`
     The `clock` process terminates.

   The `clock` processes shall increase the internal time with the configured frequency by means of the construct `receive ... after`.

   Additionally, it might be reasonably to equip the `clock` processes with a trace facility to visualize their inner working including messages to turn logging on and off.

   b) `start` function
   Please define a function `start` in the module `clock` that starts `clock` processes. Its single parameter shall determine the speed of the `clock`-process.

   c) `get` function
   Please encapsulate asking `clock` processes for their time in a `get` *function* that first sends an asynchronous `get` message then expects a `clock` reply message and finally extracts the time from this reply message.

   d) `tick` messages
   Using `receive ... after` directly in the clock process has a serious drawback. Can you spot what it is?

   Instead of using `receive ... after` directly in the clock process there should now be a connected ticker-subprocess that regularly generates tick events (of course by means of `receive ... after` or `timer:sleep`) and periodically sends `tick` messages to the original clock process. Please enhance the original clock process to handle these `tick` messages by incrementing its internal time (if not paused).

   Please adjust the creation of the clock process so that it also creates the ticker-subprocess.

   How can you assure that the clock process only handles `tick` messages by its own ticker-subprocess and not by anyone sending `tick` messages to it?

2. **Timer Processes**
   Similar to the above clock processes, *design* and *define* `timer` processes that allow to set a time span (in ticks) and invokes a given function when that time span has been elapsed.

Please be prepared for face-to-face technical discussions. These will happen during the on-site tutorial sessions.