```
  constraint val {
    how_many >= 10; how_many <= 256; }

  function new(string name = "n_packets");
    super.new(name);
  endfunction

  task create_called_N_times();
    packet t;

    for(int i = 0; i < how_many; i+=8) begin
      for (int j = 0; j < 8; j++) begin
        fork
          begin
            packet my_t;
            my_t = packet::type_id::create(
              "packet",,get_full_name());

            if (!my_t.randomize() )
              `uvm_fatal("SEQ", "Randomize failed")

            start_item(my_t, packet_priority);
            finish_item(my_t);
          end
        join_none
      end
      wait fork;

      #5;
    end
  endtask

  task create_called_once();
    packet t;

    t = packet::type_id::create(
      "packet",,get_full_name());
    for(int i = 0; i < how_many; i++) begin
      if (!t.randomize() )
        `uvm_fatal("SEQ", "Randomize failed")

      start_item(t, packet_priority);
      finish_item(t);
      #5;
    end
  endtask

  task body();
    create_called_N_times();
    // create_called_once();
  endtask

  function void do_record(uvm_recorder recorder);
    super.do_record(recorder);
    $add_attribute(recorder.tr_handle,
      how_many, "how_many");
  endfunction
endclass

// ----------------------------------------

class driver extends uvm_driver#(packet);
  `uvm_component_utils(driver)

  function new(string name = "driver",
      uvm_component parent = null);
    super.new(name, parent);
  endfunction

  task run_phase(uvm_phase phase);
    forever begin
      packet t;
      seq_item_port.get_next_item(t);
      `uvm_info("DRVR",
        {" Got ", t.convert2string()}, UVM_MEDIUM)
      foreach (t.payload[i]) // Consume time.
        #10;
      seq_item_port.item_done();
    end
  endtask
endclass
```

```
class env extends uvm_env;
  `uvm_component_utils(env)

  driver d;
  uvm_sequencer#(packet) sqr;

  function new(string name = "env",
      uvm_component parent = null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    d = driver::type_id::create("driver", this);
    sqr = uvm_sequencer#(packet)::type_id::create(
      "sequencer", this);

    sqr.set_arbitration(SEQ_ARB_STRICT_FIFO);
  endfunction

  function void connect_phase(uvm_phase phase);
    d.seq_item_port.connect(sqr.seq_item_export);
  endfunction
endclass

class test extends uvm_test;
  `uvm_component_utils(test)

  env e1;

  function new(string name = "test",
      uvm_component parent = null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    e1 = env::type_id::create("e1", this);
  endfunction

  task run_phase(uvm_phase phase);
    n_packets seq1, seq2;

    phase.raise_objection(this);

    packet::type_id::set_type_override(
      packet_with_randc_addr::get_type);

    packet::type_id::set_inst_override(
      big_packet::get_type(),
      "uvm_test_top.e1.sequencer.seq1.packet");

    seq1=n_packets::type_id::create(
      "seq1",,get_full_name());
    seq2=n_packets::type_id::create(
      "seq2",,get_full_name());

    seq1.packet_priority = 1000;

    factory.print();

    if (!seq1.randomize() with {how_many == 256;})
      `uvm_fatal("TEST",
        "Randomization failed for seq1")
    if (!seq2.randomize() with {how_many == 256;})
      `uvm_fatal("TEST",
        "Randomization failed for seq2")

    fork
      seq1.start(e1.sqr);
      seq2.start(e1.sqr);
    join

    phase.drop_objection(this);
  endtask
endclass

module top();
  initial begin
    uvm_config_db#(int)::set(null, "",
"recording_detail", 1);
    run_test("test");
  end
endmodule
```