Nikitha
4768

1] module semaphore_ex;
semaphore sem;
initial begin
sem = new(4);
fork
display("Process 1");
display("Process2");
display("Process3");
display("Process4");
join
end
task automatic display(string msg)
sem.get(2)
$display($time, "\t got key for %s", msg);
#30;
sem.put(2)
$display($time, "\tPutting back key by %s", msg);
endtask
endmodule

Output:
0 got the key for process1
0 got the key for process2
30 Putting back key by Process1
30 Got the key for Process3
30 Putting back key by Process2
30 Got the key for process4
60 putting back key by Process3
60 putting back key by Process4

Explanation:
- 4 processes are created (process 1, 2, 3 & 4) within the fork-join
- Each process will call the task display
- display task will have the sem.get & sem.put method where we are getting the 2 keys and after some time putting back the same key.

2) module semaphore_ex;
semaphore sem;
initial begin
sem = new(4);
fork
display("process 1");
display("process 2");
display("process 3");
display("process4");
join
end
task automatic display(string msg)
sem.get(3)
$display($time, "\t got key for %s", msg);
#30;
sem.put(3)
$display($time, "\t putting back key by %s", msg);
endtask
endmodule

output:
0 got the key for process 1
30 putting back the key by process 1
30 got the key for process 2
60 Putting back key by process 2
60 got the key for process 3
90 putting back key by process 3
90 got the key for Process4
120 putting back key by process4

3) my_sem = new(1);
task
forever begin
my_sem.get();
my_process1();
my_sem.put();
end
forever begin
my_sem.get();
my_process2();
my_sem.put();
end
join

- Op:
- 2 forever begin loop executes parallelly.

O/P:
0 process 1
10 process 2
20 process 1
30 process 2
⋮
- It will continues forever.

4) module test;
event e_a;
initial begin
#30;
-> e_a;
end
initial begin
wait(e_a);
end
task wait(e_a);
wait(e_a.triggered);
$display("event received");
endtask
endmodule.

4) module event_test;
event e_a, e_b, e_c, e_d;
initial begin
#30
-> e_a;
end
initial begin
wait(e_a);
fork
begin: P1;
-> eb;
end: P1
begin: P2;
-> e_c;
end: P2;
begin: P3
-> e_d;
end: P3
join

task wait(e_a);
wait(e_a.triggered);
$display("event e_a received")
endtask
endmodule.

5)
```
class generator;
rand byte packet;

endclass

class driver #(mailbox m);
  task A;
  m.get(msg);
  $display(" packet received %d", msg);
  endtask
  endclass
module test;
    mailbox mi_box = new();
    initial begin
    generator g= new();
    driver #(mb_new) d= new();
      repeat(10) begin
        g.randomize();
        mb_new.put(g.packet);
      #1 d.A();
      end
      endmodule
```

6)
```
Time: 1  calling Task LO-PRI-0  priority: 0
Time: 2  calling Task LO-PRI-1  priority: 0
Time: 3  calling Task LO-PRI-2  priority: 0
Time: 4  calling Task LO-PRI-3  priority: 0
Time: 5  calling Task HI-PRI0   priority: 1
Time: 5  calling Task LO-PRI4   priority: 0
Time: 51 completed Task LO-PRI-4 priority: 0
Time: 101 Completed Task HI-PRI-0 priority: 1
Time: 151 completed Task LO-PRI-4 priority: 0
Time: 201 completed Task LO-PRI-3 priority: 0
Time: 251 Completed Task LO-PRI-2 priority: 0
Time: 301 completed Task LO-PRI-1 priority: 0
```

7) abc-2 will execute in case 1. & it will execute parallely.
- Bcz, in case 1; we have fork-join
- But in case 2: it will execute in sequentially, in which abc-2 will not execute.

8] module test;
initial begin
$display ("process1");
end
initial begin
$display ("process2");
end
endmodule

9] @ has more priority than wait

10] @ #01 before trigger
@ #02 before trigger
@ #02 after trigger