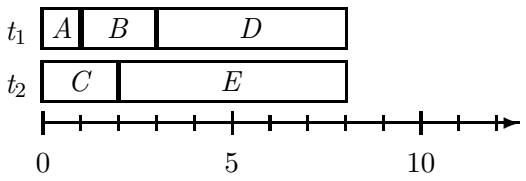
Solutions for Exercises, Week 11

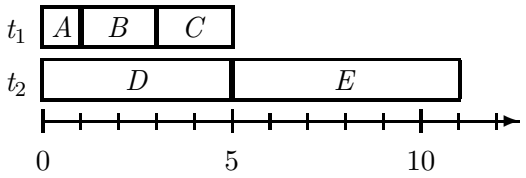
1. Solution for ParComp.1

(a) With the given task execution times it is possible to make a perfect fit:



This gives an execution time of 8 seconds and a speedup of $\frac{16}{8} = 2$.

(b) If task E is postponed, the following scenario is possible:



with an execution time of 11 seconds and a speedup of $\frac{16}{11} \approx 1.45$.

2. Solution for Concurrent Systems Exam December 2002, Problem 3

Question 3.1

(a) $I \triangleq open \Rightarrow waiting(Queue) = 0$

(b) I is a monitor invariant since processes only wait at *Queue* if *open* is false and *Queue* is emptied by *signal_all* whenever *open* becomes true.

(c) If there are currrently less than k processes waiting when $Go(k)$ is called, all of these are woken, but new calls of $Pass()$ still have to wait (if closed). As a special case, if the gate is open, a call of $Go(k)$ has no effect.

Question 3.2

```
module Gate
  op Pass();
  op Set(boolean);
  op Go(integer);
body
  process Keeper;
    var open : boolean := false;
    repeat
      in Pass() and open → skip
      [] Set(b : boolean)   → open := b
      [] Go(k : integer)    → for j in 1..min(k, ?Pass) do
                                in Pass() → skip ni
      ni;
    forever;
end Gate;
```

3. Solution for Concurrent Systems Exam December 2003, Problem 3

Question 3.1

The first three calls of *put*() will enable one of the calls of *unload*() to succeed. The two remaing calls of *put*() will then both succeed leaving the server with $count = 2$. The second call of *unload*() will remain blocked waiting for acceptance by the server.

Question 3.2

```
monitor Batch

  var count : integer := 0;
      NonFull : condition;
      Full : condition;

  procedure unload() {
    while count < N do wait(Full);
    count := 0;
    signal(NonFull);
  }

  procedure put() {
    while count = N do wait(NonFull);
    count := count + 1;
    if count < N then signal(NonFull) else signal(Full);
  }

end
```

In this solution, care has been taken to wake up only the mininum number of waiting *put*-calls using a cascade wakeup on *NonFull*. A solution in which *unload* signals to all on *NonFull* is also acceptable.

4. Solution for Concurrent Systems Exam December 2004, Problem 3

Question 3.1

```
monitor Latch

  var count : integer := 0;
      IsZero : condition;

  procedure set(k : integer) {
    if k ≥ 0 then count := k;
    if count = 0 then signal_all(IsZero)
  }

  procedure down() {
    if count > 0 then count := count − 1;
    if count = 0 then signal_all(IsZero)
  }

  procedure await() {
    while count ≠ 0 do wait(IsZero)
  }

end
```

[The signalling in *set* is necessary since *count* may be set to 0 (!). The **while**-loop in *await* might be replaced by an **if**-statement although in the server-based solution, not all waiting processes are guaranteed to get through before a *set* is called and hence the solution shown here is closer to this semantics.]

Question 3.2

The given SYNCHRONIZE code indicates a solution with two simple (i.e. one-time) barriers in a row. Care must be taken in resetting the simple barriers properly.

```
process Q;
  repeat
    latch1.await();
    latch3.set(n);  latch4.set(1);          — Prepare for second stop
    latch2.set(0);
    latch3.await();
    latch1.set(n);  latch2.set(1);          — Prepare for next round
    latch4.set(0);
  forever;
```

[*set*(0) may be replaced by *down*(). Resetting of $latch_1$ and $latch_3$ may be done earlier.]