Solutions for CP Exercises, September 11

1. Solution for Share.1

The problem is to get two processes to make a true synchronization, i.e. the processes should "meet" inbetween they perform their operation.

Proposal 1

An obvious way to ensure the invariant is to use directly the variables that appear in the invariant, i.e. the number of times OP_A respectively OP_B have been executed. This results in the following program:

var a, b : *integer*;

$a := 0;\ \ b := 0;$

process A =	**process** B =
repeat	**repeat**
while $a > b$ **do** ;	**while** $b > a$ **do** ;
...	...
OP_A;	OP_B;
$a := a + 1$;	$b := b + 1$;
forever	**forever**

Notice that both tests and assignments are atomic according the the rule of critical references. Note also that the test cannot be changed to $a \neq b$ since deadlock may then occur (how?). Furthermore, it is important that both a and b are incremented *after* the execution of the respective operation.

This solution, however, has the drawback that it increments a and b indefinitely. An attempt to count **modulo** somthing is very difficult to do correctly. Another problem is that the synchronization code is spread over the process. This is readily solved by moving the increments up before the while-loops.

Proposal 2

An attempt to synchronize using flags could be to use a flag for each process according to the following strategy: When a process arrives at the "meeting place", it raises its flag and waits for the flag of the other process to come up. When this is the case, it lowers its own flag and continues.

This solution is not correct! If a process is suspended after having raised its flag, the other process may run without blocking.

The solution cannot be patched by having the processes wait for the other process to take its flag down, before lowering its own, since this solution is subject to deadlock.

A correct solutionis obtained by having the processes take down the flag of the *other* one and then wait for their own flag to be lowered. It is, however, sufficient with only "half" of this solution. See proposal 4, below.

Proposal 3

One may try to use a variable to alternate the execution of the two processes:

var $Turn : (A, B)$;

Initialization: $Turn := A$; – Arbitrary

$SYNC_A$:	$SYNC_B$:
$Turn := B$;	$Turn := A$;
while $Turn = B$ **do** ;	**while** $Turn = A$ **do** ;

This solution satisfies the invariant as no operation will get ahead of the other, but it will not utilize the concurrrency of the problem. A concurrent solution is presented below:

Proposal 4

Here we make an assymmetric solution with only one flag. The flag is used as follows: When a certain of the processes (say P_A) arrives at the meeting place, it raises the flag and waits for it to be lowered. The other process will start by waiting for the flag to be raised and will then lower it. In this way it is ascertained that both process will wait for the other one to arrive at the meeting place.

var $Flag : boolean$;

Initialization: $Flag := false$;

$SYNC_A$:	$SYNC_B$:
$Flag := true$;	**while** $\neg Flag$ **do** ;
while $Flag$ **do** ;	$Flag := false$;

In this solution there is no risk of overflow and the synchronization code has been gathered in the beginning of the processes.