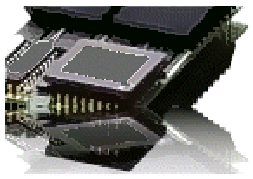

Brock University
Department of Computer Science

COSC 3P93 A2: Supercomputer Reset v.2

Instructor: **Vlad Wojcik**

DUE DATE / TIME: `Monday, 23 Nov 2015, 4 PM.`

INTRODUCTION:

We still want to reset a 3D torus supercomputer, but even faster. To that end we will equip the user program running on a host computer with a semaphore, to be signalled by the torus when reset is complete.

In operating systems (peruse your notes of COSC 2P13 or find a **refresher here**) we used the concept of a semaphore as a tool to synchronize processes and to enforce mutual exclusion. The semaphore S is normally seen as a non-negative integer variable initialized to a particular suitable value at system boot time. Two operations are defined on a semaphore:

> **`signal(S);`** -- increments the semaphore value **S** by one;
> **`wait(S);`** -- decrements the value of **S** by one as soon as it is possible to do so without yielding **S** negative.

The system call **`wait(S)`** does not exit if it finds S <= 0. With proper programming, sooner or later another process will perform **`signal(S)`** operation(s), allowing **`wait(S)`** to exit successfully.

In Ada, we can create semaphores at will. We can specify

```
task type SEMAPHORE (INITVALUE : INTEGER := <some sutable value>) is
    entry WAIT;
    entry SIGNAL;
end SEMAPHORE;
```

and implement it as

```
task body SEMAPHORE is
    COUNT : INTEGER := INITVALUE;
begin
    loop
        select
                                         accept SIGNAL; COUNT := COUNT + 1;
            or when COUNT > 0 => accept WAIT;   COUNT := COUNT - 1;
        end select;
    end loop;
end SEMAPHORE;
```

THE PROBLEM:

As before, a 3D torus supercomputer (3x4x5 processors) needs to be reset. To reset a processor means to bring it to a certain controlled state.

Every processor in the torus has exactly six immediate neighbours. We will assume that each processor runs an Ada task. Each task has a RESET entry and an ACK entry.

The user program is equipped with a semaphore DONE, declared as follows:

```
    DONE : SEMAPHORE (INITVALUE => -1);
```

Now, for expediency, a user program, running on the host computer, first calls the RESET entry of TWO tasks running on an arbitrarily chosen processors maximally distant from each other. (The distance on a torus is measured in the minimum number of hops along the torus). Then it waits on its DONE semaphore, waiting for the acknowledgement. When this acknowledgment is received, the user program is informed that the torus multiprocessor has been reset.

Each torus processor task waits first on the RESET entry call. Once the first RESET request is received, it then calls the RESET entry of all five of its immediate neighbours (with the exception of the neighbour that called its RESET entry), and then waits on its own ACK entry for ACKnowledgments from its immediate neigbours (which may come in any order). Subsequent calls to RESET the same processor are accepted, but ignored. Upon receipt of all due ACKnowledgments, the task calls the ACK entry call of the neighbour that first called its RESET entry.

Finally the two processors originally instructed to RESET the entire torus by the user program running on the host computer are in a position to instruct that program, that the RESET operation is complete. They do it, each by issuing the call:

```
        DONE.SIGNAL;
```

Write the most efficient Ada program to reflect this behaviour. While the torus is being reset, the program should produce a brief report, viz.:

```
        PNO    #ACKs
        ###     ####
        ###     ####
```

where PNO stands for processor number and #ACKs represents the number of acknowledgments issued.

NOTE: Beware of deadlock!

SUBMISSION FORMAT:

Both hardcopy (paper) and electronic submission are required.

Hardcopy submission: Your submission envelope with the standard Cover Page should contain all relevant printouts and supporting documentation, demonstrating your design and flawless behaviour of your program. The envelope should be dropped in the submission box on or before deadline date / time.

Electronic submission: Please create a directory on Sandcastle and place within it all the files (and only the files) to be submitted. To submit issue the command **submit3p93**, which is interactive in its nature. Obviously, you are allowed to submit your assignment only once. Should you encounter difficulties, please report them to your TA.

Similarly, the electronic submission should be performed on or before deadline date / time.

Instructor: Vlad Wojcik
Revised: 22 September, 2015 9:52 PM