Suggested Plan for Assessment 4 (Dining Philosophers Animation)

(0) Start with the **q7.occ** starter file from your exercises folder - also on raptor:

```
\courses\co538\exercises\q7.occ
```

(1) Add report channels to the philosopher, fork and security processes – initially, these can carry **INT** codes for the states being reported. What should these states and codes be? As soon as possible, change to the method in (2) below – this requires knowledge of variant protocols (see the *__protocol__* slides 64-71). *Meanwhile, skip to item (3).*

(2) Use a suitable **CASE** protocol (same one for all reports). The philosopher and fork messages are just *tags* (**thinking**, **hungry**, **sitting**, **eating**, ..., **on.table**, **held.right**, **held.left**) or similar names. The security message has a *tag* (e.g. **security**); then an **INT** (for how many are in the dining room).

(3) Wire all these from the college processes to an array of 11 external channels - say philosophers use elements 0-4, forks 5-9 and the security guard uses element 10. See *__applying__* slides 61 and 64. You could use the suggestion in slide 63 (which is *cleaner*) – but the slide 64 approach makes it easier to program the display process (slide 65). This display has to catch all the reports from the college and process them for animation on the screen – see item (6) below. With three separate groups of report channels (slide 63), providing a *fair service* to all processes in the college making reports is tricky. With one array (slide 64), this is easy.

(4) In the main process, set up a network connecting the college and a display process with an array of 11 report channels. See *__applying__* slide 65.

(5) In philosopher, program the thinking and eating periods initially as fixed length delays – say 5 and 8 seconds respectively. Give each philosopher its own id (as a **VAL INT** parameter), where the ids run from 0 through 4. To vary each philosopher's behaviour slightly, add its id to the chosen delay times above.

(6) Program the display process to (replicated) **ALT** over all its incoming report channels. Generate one line of text for each report received. This line should say which phil and/or fork is involved (deducible from the channel index of the **ALT** input guard). This may be tricky for fork messages: e.g. it should say the fork number and which phil is holding it.

Now you have a system that should compile and run! The next two items, (7) and (8), will simplify things – but only after reading the relevant bits of the *__shared-etc__* slides. *Initially, skip to item (9).*

(7) Suggest using a single "**SHARED! CHAN**" report channel, instead of the array of 11 individual channels. See slides 3-10 of *__shared-etc__* and slide 69 of *__applying__*. The college header now looks something like:

```
PROC reporting.college (SHARED CHAN REPORT report!)
```

and inside, all its (**PAR**) components can plug into the shared "**report!**" parameter (slide 71 of *__applying__*).

Now, the display just takes a single (unshared) report input channel (slide 70 of *__applying__*) and simply waits on that at the start of its loop - no **ALT**ing and no need to worry about *fairness*, since the reporters FIFO queue on their shared end of the channel. But now, the report channel needs enhancing to add identity numbers (**INT**s) to each philosopher and fork message (because display no longer has channel indices from which it can deduce which philosopher or fork sent the message).

So, each philosopher and fork process needs an extra "VAL INT id" parameter (which should be from 0-4 both for philosophers and for forks). Work out how to pass these into the philosopher and fork instances (it's trivial).

Does the system still work?

(8) Suggest using protocol inheritance (slides 28-37 of the same *__shared-etc__*). Have separate **CASE** protocols for philosopher, fork and security reports. Use these appropriately for the philosopher, fork and security processes. Define a **REPORT** protocol to extend those for philosopher, fork and security.

Does the system still work?

(9) Check out the main process (**q7**). The starter file has code there to set a random number seed. Find out how to use the **random** function:

```
INT, INT FUNCTION random (VAL INT n, seed)
```

which returns (first) a *pseudo-random* **INT** in the range 0..(n-1) and (second) an updated seed value, which must be used the next time **random** is called. See *__shared-etc__* slides 88-97 for occam-π functions. Be happy about *functions* returning two (or more) values - it's a useful mechanism (Java *methods* can only return a single result).

You can find documentation about **random** by following the link to the on-line *documentation* from the module page (scroll past the weekly notices, 6th. bullet in the Practical Resources box, click on "**Module course**", search for "**random**" and follow the link to "**Function random**"). Alternatively, go directly to:

http://occam-pi.org/occamdoc/course.html#name-random

(10) Each philosopher needs a distinct seed to use with **random**. Easy, pass all philosophers the seed set up in the main **q7** process and get each philosopher to add its **id** number. Now, they will all have different seeds ... but there is a problem with the simple (fast) random number logic used by the **random** function – it needs to *'warm up'* (see the anon Q&A year 2006, Q55).

You are strongly advised to make use of the anon Q&A resource. For this assessment, look in the keyword-index for things like "**q7**" (an old number for this exercise), "**coursework**" and "**random**".

(11) Improve the programming of the thinking and eating times of each philosopher – see (5) above – so that they wait for a random amount of time within the limits defined by **VAL INT**s in their starter file (**min.think.time**, **max.think.time**, etc.).

Does the system still work?

(12) Improve the programming of the display process so that we get some kind of animation, rather than scrolling of lines of text. Examples can be seen with the *Transterpreter* by loading any of the files from the raptor folder:

\courses\co538\answers\dining-phils-transterpreter*.occ

and running them from the occPlug. Don't try to compile! The *basic* solution is:

\courses\co538\answers\dining-phils-transterpreter\a7-basic.occ

which runs the *minimum standard of animation* that can achieve full marks on this assessment. If you do just this, you probably won't get full marks – because you will lose some somewhere. By achieving a better animation (e.g. where the phils/forks/etc. actually *move* in parallel – though not, of course, doing exactly the same things), you will get bonus marks that can bring you back to 100%. The other solutions there are to challenge your imagination (and programming, ☺).

You'll need to find out about:

```
PROC cursor.x.y (VAL BYTE x, y, CHAN BYTE out!)
```

Look up its documentation - it's in "**Module course**":

http://occam-pi.org/occamdoc/course.html#name-cursor.x.y

The other "**cursor**" (and "**erase**") **PROC**s there may also be helpful.

You should long ago have looked at:

examples\test-utils.occ

for more examples on the use of the screen utitlites.

Also, check out the final component of the **PAR** in your starter file (labelled*: "probably delete this process"*).

(13) Modify the fork process to *'fair alt'* over its pickup channels from its neighbouring philosophers. Same for display, if it's still receiving from an array of report channels.

(14) Make your animation *interactive* (see slide 70 of *__applying__*). The interaction could simply be to freeze and resume all action. Harder is to speed up and slow down the action. Maybe send a message to college security to change the maximum number of philosophers it allows in to the dining room ... setting this to 5 means deadlock is possible ... setting it to lower numbers, including zero, will change what you see happening. Maybe send message to individual philosophers that change its behaviour in some (interesting) way.

Note: the last two ideas above require multiple channels from the keyboard monitor to the reporting college - one for each target process in the college. Only one is shown in slide 70.

(15) A marking scheme for this assessment is in the answer to Q103 (of the anonymous Q&As for 2006). Go look. This year's scheme will be *similar* (but will have some marks for interaction). You will also need to provide some system diagrams. I will publish an actual marking scheme shortly.

Peter Welch (7th. November, 2012).