The occam-π “Better Bar” Simulation

and how it all works

Fred Barnes, Systems Research Group
Computing Laboratory, University of Kent
F.R.M.Barnes@kent.ac.uk
The Communicating Processes Paradigm

- Systems are built from layered networks of communicating parallel processes

- Synchronous point-to-point communication via 'channels'

- The three sub-components here are 'plus', 'prefix' and 'delta'
  - that could also be parallel process networks

- Implementation in occam-$\pi$; semantics from Hoare’s CSP and Milner’s $\pi$-calculus — mathematics
The Communicating Processes Paradigm

- Individual processes are completely isolated;
  - interacting with the environment only through channels visible in their interfaces

- The synchronous nature of channel communication means that different implementations may behave differently with respect to their environment.

- Formal methods (CSP) permit reasoning about these behaviours.

\[
\text{DELTA} = \text{in}?x \rightarrow (\text{out}.0!x \parallel \text{out}.1!x) \delta \text{DELTA}
\]

- See books by Tony Hoare and Bill Roscoe for more..
The occam-$\pi$ Programming Language

- A highly concurrent programming language, derived from classical occam
  - can build systems containing millions of communicating parallel processes
  - context-switch time (to switch the CPU from one process to another) in the order of 50 nanoseconds (2.4 GHz P4)
  - mobile channel types, mobile processes, process priority, shared channels, extended synchronisations, dynamic process creation, ...
  - guaranteed freedom from aliasing and race-hazard error

- On-going development of the language and its applications at Kent
  - to educational ends, breaking the view that concurrency is hard and providing a robust software engineering methodology
  - building internet servers (e.g. the occam-$\pi$ web-server)
  - building highly-concurrent and scalable operating-systems

http://www.cs.kent.ac.uk/projects/ofa/kroc/
The “Better Bar” Simulation

- Originally used to demonstrate fairness (or rather the lack of it – starvation)
  - when there are two or more people waiting to be served, whom does the bartender choose?

- Implementation consists of around 100 parallel processes (some come and go)
  - uses a negligible amount of CPU – mostly waiting for timeouts

- Featuring: 14 drinking philosophers, 2 bar-tenders, beer-cellar and supplier, de-tox machine, ambulance service, juke-box, pool-table, teleporter and 2-player ‘shoot-em-up’ philosopher paintball
The “Better Bar” Simulation

Most central component is the ‘display’ process:

(from philosophers)  (from bar)  (from cellar)  (from beer-corp)

(from infoline)  (from game)  (from projectiles)  (from nhs)

 Processes compete for access to the shared channel, sending tagged-protocol messages to draw on the screen:

CLAIM to.dpy!
SEQ
  to.dpy ! colour; ANSI.FG.BLUE
  to.dpy ! string.x.y; 4; 1; "4 pints"

Adding additional features is trivial :-(
The “Better Bar” Simulation

- Majority of the complexity lies in ensuring objects on the screen avoid each other
  - central ‘collision.detector’ process responsible for this:
    - (from philosophers)
    - (from bar)
    - (from game)
    - (from projectiles)

- ‘collision.detector’ maintains a representation of the screen, noting what is where:

- Sprites have ‘buddy’ processes to help them move around
The “Better Bar” Simulation

- ‘philosopher’ processes are wired up as follows:

- Moving around avoiding obstacles is handled by ‘common.ifcode’; ‘phil.maincode’ simply does:

  ```
  out ! move.x.y; cx; cy; tx; ty
  ```

- Being hit by paintballs is handled in ‘common.ifcode’

- Similar process networks used for the bar-tenders
The “Better Bar” Simulation

Game processes represent the interactive portion:

- keyboard.if
- game.if
- projectile.network
- score.keeper

- keyboard?
- (to philosophers)
- (to philosophers)
- (to philosophers)
- (to philosophers)
- (to collision-detector)
- (to collision-detector)
- (to display)
- (to display)
- (to display)
- (to display)

A relatively complex process network — some care required in programming to prevent deadlock

Fred Barnes, October 2005
The “Better Bar” Simulation

Player 1 keys:

/ (dot or slash to fire, arrow keys to move)

Player 2 keys:

q, r, t (space-bar to fire, q,a = up, down, r,t = left, right)

Use fire to start

• objective is to shoot philosophers and turn them your colour (green or red)
• game ends when nothing has been shot for a certain amount of time (shown at the bottom-right of the screen)
Acknowledgements

Concurrency research sub-group:

- Part of the Systems Research Group:
  
  http://www.cs.kent.ac.uk/research/groups/sys/

- Academic staff:
  - Professor Peter Welch
  - Dr. Fred Barnes
  - Mr. David Wood

- Research students:
  - Mario Schweigler, Christian Jacobsen, Damian Dimmich
  - Adam Sampson, Joseph Sheridan

More general information on communicating process architectures:

http://www.wotug.org/