Coroutines

Coroutines are supported by Simula and Modula-2. They are similar to Java’s threads, except the programmer has to explicitly transfer control from one execution context to another. Thus, like threads several coroutines can exist simultaneously but unlike threads there is no central scheduler that decides which coroutine should run next. A coroutine is represented by a closure. A special operation transfer(C) shifts control to the coroutine C, at the location where C last left off.

Coroutines...

The next slide shows an example from Scott where two coroutines execute “concurrently”, by explicitly transferring control between each other. In the example one coroutine displays a moving screen-saver, the other walks the file-system to check for corrupt files.

```plaintext
var us, cfs: coroutine;
coroutine update_screen() {
    ... detachloop{
        ... transfer(cfs) ...
    }
}
coroutine check_file_system() { ... }
main () { ... }
```
Coroutines in Modula-2

Modula-2's system module provides two functions to create and transfer between coroutines:

```plaintext
PROCEDURE NEWPROCESS(
    proc: PROC; (* The procedure *)
    addr: ADDRESS; (* The stack *)
    size: CARDINAL; (* The stack size *)
    VAR new: ADDRESS); (* The coroutine *)

PROCEDURE TRANSFER(
    VAR source: ADDRESS; (* Current coroutine *)
    VAR destination: ADDRESS); (* New coroutine *)
```

The first time `TRANSFER` is called, `source` will be instantiated to the main (outermost) coroutine.

---

Readings and References

- Read Scott, pp. 474–479
- [http://www.mathematik.uni-ulm.de/oberon/0.5/articles/coroutines.html](http://www.mathematik.uni-ulm.de/oberon/0.5/articles/coroutines.html)