CSc 520

Principles of Programming Languages

36: Procedures — Coroutines

Christian Collberg

collberg@cs.arizona.edu

Department of Computer Science

University of Arizona

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.

Coroutines. . .

```
var us, cfs: coroutine;

coroutine update_screen() {
   ...
   detach
   loop {
      ... transfer(cfs) ...
   }
}

coroutine check_file_system() {  ... }

main () { ... }
```

Coroutines. . .

```
coroutine check_file_system() {
   ...
   detach
   for all files do {
      ... transfer(cfs)
      ... transfer(cfs)
         ... transfer(cfs) ...
   }
}

main () {
   us := new update_screen();
   cfs := new check_file_system();
   transfer(us);
}
```

Coroutines in Modula-2

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

```
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.

Coroutines in Modula-2. . .

```
VAR crparams: CoroutineParameters;
    source: ADDRESS; (* current coroutine is called by this *)
    newcr: ADDRESS; (* coroutine just created by NEWPROCESS *)

PROCEDURE Coroutine;
   VAR myparams: CoroutineParameters;
BEGIN
   myparams := crparams;
   TRANSFER(newcr, source); (* return to calling coroutine *)
   (* rest of coroutine *)
END Coroutine;

PROCEDURE Setup(params: CoroutineParameters; proc: PROC);
BEGIN
   NEWPROCESS(proc, addr, size, newcr);
   crparams := params; TRANSFER(source, newcr);
END Setup;
```

Readings and References

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