Prof. P. Thiemann, S. Wehr Summer Term 2009

Concepts of Programming Languages

http://proglang.informatik.uni-freiburg.de/teaching/konzepte/2009ss/

Exercise Sheet 8

2009-06-19

Exercise 1 (6 points)
Modify the THREAD language to include *thread identifiers*. Each new thread is associated with a fresh thread identifier, represented as a natural number. Spawning a child thread returns the thread identifier of the newly created thread. Moreover, the child thread receives a two-element list containing its own thread identifier as well as the identifier of its parent thread. Here is a short example:

```
let thread = proc(tids) begin
               print(car(tids);      % thread id of this thread
               print(car(cdr(tids))  % thread id of the parent thread
             end
in let child-tid = spawn(thread)
in print(child-tid)                  % thread id of the newly created thread
```

Exercise 2 (3 points)
Implement mutexes in Java.[1] A mutex should be represented by a Java class `Mutex`, supporting the following methods:

`Mutex(boolean isOpen)` The constructor initializing the state of the mutex.

`void mutexWait()` The wait function of the mutex, as explained in the lecture.

`void mutexSignal()` The signal function of the mutex, as explained in the lecture.

Test your mutex class using the implementation of the dining philosophers problem available from the lecture's homepage.

Exercise 3 (3 points)
An `MVar<T>` (mutable variable) is a "box" which is either empty or contains a value of type `T`. It supports the following operations:

`T take()` Returns the value stored in the box, blocks until the box becomes full.

`void put(T value)` Stores a new value in the box, blocks until the box becomes empty.

[1] Use only Java concurrency primitives such as `synchronized`, `wait`, `notify` etc. External libraries are not allowed.

Initially, the box is empty. Use mutexes to implement the `MVar<T>` class in Java.[2]

Exercise 4 (4 points)
Implement channels in Java based on `MVar`s.[3] A channel `Channel<T>` should permit multiple threads to write values of type `T` to it, and read values of type `T` from it, safely. It supports the following operations:

`void put(T value)` Puts a value into the channel. This operation always succeeds and never blocks.

`T get()` Reads a value from the channel. This operation blocks if the channel is empty. Otherwise, it returns the value that was least recently put into the channel. In other words, the values of the channel are subject to a "last in, first out" policy.

Submission
Via email to `wehr@informatik.uni-freiburg.de`. The strict submission deadline is **2009-06-25, 1 pm.**

[2] Java concurrency primitives and external libraries are not allowed.
[3] Java concurrency primitives and external libraries are not allowed.