

INFO1131 Practical Exercises

Lab 11: Monitors and Transactions

This lab session is also dedicated to shared-state concurrency. This time we will work on monitors and transactions.

1. **Mvar** is a box that can be full or empty. It comes with two procedures, **Put** and **Get**. If the box is empty, **{Put X}** puts **X** in the box, and the box become full. If the box is full, **Put** waits until it is empty. Doing **{Get X}** on a full box atomically binds **X** to the content of the box and empties the box. If the box is empty, **Get** waits until the box is full. Implement a procedure **{MakeMvar Put Get}** that creates a **MVar**, and that binds variables **Put** and **Get** to the procedures that provides the functionality previously described.

There are many possible implementations.

First, implement it using monitors (lock, wait, notify, notifyall). Use the implementation in file `monitor.oz` on Moodle.

Is it possible to implement that with locks, atomic cell exchange or ports? Pick one of the three and implement it.

2. **Breaking up big transactions.** Use the implementation of a transaction manager described in Figures 8.23 and 8.24 from the book of the course, and that can be found in `transaction.oz` on Moodle.

Consider this example from the book:

```
declare
Trans NewCellT
{NewTrans Trans NewCellT}

T={MakeTuple db 1000}
for I in 1..1000 do T.I={NewCellT I} end

fun {Rand} {OS.rand} mod 1000 + 1 end

proc {Mix}
  {Trans proc {$ Acc Ass Exc Abo _}
    I={Rand} J={Rand} K={Rand}
    if I==J orelse I==K orelse J==K then {Abo} end
    A={Acc T.I} B={Acc T.J} C={Acc T.K}
  in
    {Ass T.I A+B-C}
    {Ass T.J A-B+C}
    {Ass T.K ~A+B+C}
  end _ _}
```

```

end

S={NewCellT 0}
fun {Sum}
  {Trans fun {$ Acc Ass Exc Abo}
    {Ass S 0}
    for I in 1..1000 do
      {Ass S {Acc S}+{Acc T.I}} end
      {Acc S}
    end _}
  end
end

{Browse {Sum}} % Displays 500500
for I in 1..1000 do {Mix} end % Mixes up the elements
{Browse {Sum}} % Still displays 500500

```

The example above defines the transaction Sum that locks all the cells in the tuple while it is calculating their sum. While Sum is active, no other transaction can continue. How could you rewrite Sum as a series of small transactions? The goal is that each small transaction should only lock a few cells. You can define a representation for a partial sum, so that a small transaction can see what has already been done and determine how to continue. By doing this, you will allow your system to perform other transactions while a sum calculation is in progress.

- Does it really make sense to break such a big transaction into small ones?
 - Implement the solution and confirm your reasoning.
3. Read and write locks. The transaction manager locks a cell upon its first use. If transactions T1 and T2 both want to read the same cell's content, then they cannot both lock the cell simultaneously. We can relax this behavior by introducing two kinds of locks, read locks and write locks. A transaction that holds a read lock is only allowed to read the cell's content, not change it. A transaction that holds a write lock can do all cell operations. A cell can either be locked with exactly one write lock or with any number of read locks. For this exercise, extend the transaction manager to use read and write locks.