1 Threads Are Lightweight

Try the following example:

```oz
fun {Fib N}
  case N
  of 0 then 0
  [] 1 then 1
  else
    thread {Fib N-1} end + thread {Fib N-2} end
  end
end
```

Use the Oz Panel to see how many threads are created. How many threads are simultaneously runnable?

Also translate the above function into kernel language!

2 Hamming Numbers

The Hamming problem (named after Richard Hamming) is to generate the first $n$ integers of the form $2^i \times 3^j \times 5^k$ for $i,j,k \geq 0$ in **increasing order**.

We proceed as follows:

1. Develop a lazy function `{Times Xs N}` that multiplies the integer elements of stream $Xs$ by the integer $N$.

2. Develop a lazy function `{Merge Xs Ys}` that merges the two ordered streams of integers $Xs$ and $Ys$. Merging is as described in the eighth tutorial.

3. Bind the variable $Hs$ to the stream of Hamming numbers. Use the following idea:
   - $Hs$ starts with the element 1.
   - Other elements of $Hs$ are obtained by merging streams containing elements of $Hs$ multiplied with 2, 3, and 5.

4. In order to request element of the stream, develop a function `{Request Xs N}` that requests the first $N$ elements of the stream $Xs$. 


Why is it important to use lazy streams?
You might want to try to come up with a program computing the Hamming numbers without lazy streams. You will find that this is very challenging!

Solution.

1. We express times by using a lazy map:

   fun lazy {Map Xs F}
   case Xs
     of nil then nil
     [] X|Xr then {F X}|{Map Xr F}
   end
   end

   fun lazy {Times Xs N}
   {Map Xs fun {$ X} X*N end}
   end

2. Merging is exactly as before, just add lazy after the fun keyword.

3. Hs = 1|{Merge {Times Hs 2}
     {Merge {Times Hs 3} {Times Hs 5}}}  

4. proc {Request Xs N}
   if N>0 then {Request Xs.2 N-1} end
   end

5. Without lazy streams, the program would generate all Hamming-numbers eagerly and thus run out of memory rather soon.