Saishnu Ramesh Kumar (300758706)

CSCI 117 – Lab 6

Part 1 – Understanding Threads:

1) **Possible Thread Sequences:**

   S1 T1 S2 S3 S3.1 T2 – Displays True

   S1 T1 S2 T2 – Displays Error

   S1 S2 T1 T2 – Displays Error

   S1 S2 T2 S3 T1 – Displays Error

   S1 S2 T1 S3 S3.1 T2 – Displays True

   S1 S2 T2 T1 – Displays Error

2) **Quantum of Infinity:**

   *Hoz> runFullT (Infinity) "declarative threaded" "part1b.txt" "part1b.out"
   Y : Unbound

   T2 : Unbound

   T1 : Unbound

   Quantum of Finite 1:

   *Hoz> runFullT (Finite 1) "declarative threaded" "part1b.txt" "part1b.out"
   Y : 3

   T2 : 3

   T1 : Unbound

Explanation:

The difference between the infinity and finite 1 output is that in the infinity case, Y, T2, and T1 are unbounded. This is because when the main thread completes, the execution doesn't happen, and the values revert back to the main thread to get binded again. As for the finite 1 quantum, when it was executed, the values are Y:3, T2:3, and T1: Unbound. This happens because it was converted from the sugar syntax into kernel syntax. In this case, T1 could not be computed due to the kernel syntax of (4+3) and it got reverted back to the main thread of the program hence as to why it is unbounded once again and the output is such.

3) **Updated code with "skip Basic" statement:**

```
local Z in
  Z = 3
  thread local X in
  X = 1
  skip Browse X
  skip Browse X
        skip Basic
  skip Browse X
  skip Browse X
        skip Basic
  skip Browse X
    end
  end
  thread local Y in
  Y = 2
```

```
skip Browse Y
        skip Basic
skip Browse
skip Browse Y
        skip Basic
skip Browse Y
skip Browse Y
  end
end
skip Browse Z
skip Browse Z
skip Browse Z
        skip Basic
skip Browse Z
skip Browse Z
end
```

4) **Updated Interp.hs:**

The minimum quantum that causes a suspension to occur in the program is 5, this is more than the assumption of 3 that was stated in the lab description. The suspension happened because of the syntax sugar from number 4 Is converted into the kernel syntax which is why I was it was not the result that I expected.

5) Tracking memory usage and execution time:

a) Fib1_sugar.txt:

X	Result	Time(s)	Bytes
8	34	0.09	24,655,720
9	55	0.20	54,919,560
10	89	0.46	129,532,952
11	144	1.03	316,950,000
12	233	2.52	794,275,392
13	377	7.070	2,021,311,520
14	610	18.54	5,195,647,160
15	987	46.68	13,442,004,352
16	1597	168.39	34,936,282,448

Fib2_sugar.txt:

X	Result	Time(s)	Bytes
12	223	0.01	4,753,232
13	377	0.01	5,042,424
14	610	0.01	5,344,720
15	987	0.02	5,658,496
16	1597	0.02	5,985,888
17	2584	0.03	6,325,480
18	4181	0.02	6,676,552

For the fib1_sugar.txt, the fastest time recorded on my system was 0.09s when X=8 and the slowest would be when X=16 where the time recorded was 168.39s. As for fib2_sugar.txt, the fastest time was 0.01 and the slowest was 0.03 seconds on my system. It can overall be said that fib2_sugar.txt runs much faster than fib1_sugar.txt. The reason as to why fib1 runs slower is because it has to keep switching back to the main thread to complete the program before the binding occurs whereas fib2 does not need to do that. As for fib1_thread.txt, it just ran continuously and had to be manually stopped through the command prompt menu.

b) The pattern for fib1_thread.txt would be the Fibonacci sequence as seen in both of the tables below. It would add the two previous results together to get the next following result. As seen below to the get 2nd number, it would be the 0th result + the 1st result = 2, etc. This would keep repeating itself recursively. The first table took more longer to execute over time, as the numbers increased as it is going into another separate thread, whereas the second table was much quicker as it does not require to do that, called recursively within the main thread. The formula for this form of sequence would be (n-1) + (n-2) where n > 1.

When using (thread {Fib (In-1)} end + thread {Fib (In - 2)} end), slower execution time:

Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Result	1	1	2	3	5	8	13	21	34

When using (thread {Fib (In-1)} end + {Fib (In - 2)}), faster execution time:

Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Result	1	1	2	3	5	8	13	21	34

Part 2 – Streams:

1) local Producer Consumer OddFilter Filter N L P F Add in

```
   Producer = proc {$ N Limit Out}
   if (N<Limit) then T N1 in
      Out = (N|T)
      N1 = (N + 1)
      {Producer N1 Limit T}
   else Out = nil
   end
end
```

```
OddFilter = proc {$ P Out}
  Filter = fun {$ 01 T1}
          case 01 of nil then T1
    []'|' (1:h 2:T) then S in
    if((H mod 2) == 1) then
      S = {Filter T T1}
         S
         else
            S = {Filter T T1}
            (H|S)
      end
    end
    end
    Out = {Filter P nil}
end

// Example Testing
N = 0
L = 100
{Producer N L P} // [0 1 2 .. 100]
{OddFilter P F} // [0 2 4 .. 100]
```

2) **Consumer = fun {$ P} in**

case P of nil then 0

```
      []'|' (1:H 2:T) then
         (H + {Consumer T})
      end
end
```

3) **N = 0**

```
L = 100
thread {Producer N L P}
   skip Browse P
end
thread {OddFilter P F}
   skip Browse F
end
thread Add = {Consumer F}
   skip Browse Add
end
   skip Browse P
   skip Browse F
   skip Browse Add
end
```