Eric Smrkovsky
10/5/2019
Csci 117
Lab 7

Part One

1.1) S1 T1 S2 T2 (error)
S1 S2 T1 T2 (error)
S1 S2 T1 S3 T2 (error)
S1 S2 T1 S3 S3.1 (true) T2 (error)
S1 T1 S2 S3 S3.1 (true) T2 (error)

1.2) When the quantum is set at infinity the program does not have a chance to execute the contents of the threads because they don't get a chance to run. Since the skip statements are at the same level, they all run while the variables Y T1 and T2 all have unbound store locations. When the quantum is set to one for the program everything executes properly except the quantum causes the last skip statement to run before the thread assigned to T! finishes executing, so it is unbound when the final skip browse executes.

1.3)

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

Eric Smrkovsky
10/5/2019
Csci 117
Lab 7

1.4) The minimum quantum that causes a suspension is 5.

1.5a)

fib1_sugar.txt	**fib2_sugar.txt**	**fib1_thread.txt**
X = 13	X = 690	X= 13
41.18 secs	59.60 secs	41.37 secs
2,046,290,616 bytes	3,083,242,528 bytes	14,192,019,400 bytes
R = 377	R = -1697342244203252328	R = 377

The first sugar file is expensive, involving recursion during two recursive calls in the else part of the program. The second sugar file involves iteration passing an accumulator with each function call, when the base case is reached the program simply returns the accumulator this is a way faster and cheaper program! Also, the result for the second sugar is actually the result from an integer overflow. The thread program has similar results because it is essentially the same program except the recursive calls are within thread statements.

1.5b)

The equation I found that would calculate the amount of threads created for the nth Fibonacci number is [(fib n)*2]-2 and matches the pattern 0 0 2 4 8 14 24 40 66..

End part one.

Eric Smrkovsky
10/5/2019
Csci 117
Lab 7

Part 2.1

```
local OddFilter Producer N L P F in

  Producer = proc {$ N Limit Out}
    if (N<Limit) then T N1 in //check condition
      Out = (N|T)             //set return (this is all we need)
      N1 = (N + 1)            //increment a counter (the recursion)
      {Producer N1 Limit T}   //produce next (will tag along with T)
    else
      Out = nil               //complete list with a nil
    end
  end

  OddFilter = proc {$ P Out}
    case P of nil then        //base case
      Out = nil
    [] '|'(1:H 2:T) then M Two in
      Two = 2                 //Mod only works with variables
      {Mod H Two M}           //check if even
      if (M==0) then Rest in  //if even then
        {OddFilter T Rest}    //get the rest of the list
        Out = (H|Rest)        //result
      else
        {OddFilter T Out}     //odd case, get rest of list
      end
    end
  end

// Example Testing
  N = 0            //beginning of producer, 0 is to be included
  L = 101          //set to 101 so we reach 100
  {Producer N L P} // [0 1 2 .. 100]
  skip Browse P    //output list created by producer
  {OddFilter P F} // [0 2 4 .. 100]
  skip Browse F    //output list filtered by consumer
end
```

Eric Smrkovsky
10/5/2019
Csci 117
Lab 7

Part 2.2

```
local Producer N L Input Consumer Result in

  Producer = proc {$ N Limit Out}
    if (N<Limit) then T N1 in //check condition
      Out = (N|T)             //set return (this is all we need)
      N1 = (N + 1)            //increment a counter (the recursion)
      {Producer N1 Limit T}   //produce next (will tag along with T)
    else
      Out = nil               //complete list with a nil
    end
  end

// Consumer (returns sum of input stream)
  Consumer = fun {$ P}
    case P of nil then        //if it's empty we're done
      nil
    [] '|'(1:H 2:T) then O A in //we have a list
      if (T==nil) then        //if the tail is empty return head
        H                     //start of accumulator
      else
        O = {Consumer T}      //if there is a tail chase it
        A = (O+H)             //add head to values returned
        A                     //return accumulator
      end
    end
  end

// Example Testing
  N = 0
  L = 11                      //necessary for list of 1-10
  {Producer N L Input}        //produce a list
  skip Browse Input           //Input : [ 0  1  2  3  4  5  6  7  8  9  10 ]
  Result = {Consumer Input}   //sum it
  skip Browse Result   //Result : 55
end
```

Eric Smrkovsky
10/5/2019
Csci 117
Lab 7

Part 2.3

```
// Part 2 question 3 uses above functions (OddFilter, Consumer, Producer)
  N = 0                          //necessary for list of 0-100
  L = 101                       //necessary for list of 0-100
  thread
    {Producer N L Input}            //produce a list
    end
  thread
    {OddFilter Input OutOdd}      //remove odd numbers
    end
  thread
    Result = {Consumer OutOdd}   //sum it
    skip Browse Result
    end
 end
```