Thuy Uyen My Tran
CSCI 117
07/01/2022

Part 1 – Understanding Threads

1.1)

S1 T1 S2 S3 S3.1 T2	– Display True
S1 S2 T2 S3 S3.1 T2	– Display True
S1 S2 T2 S3 S3.1 T1	-- Display False
S1 T1 S2 T2 S3	-- Unification Error
S1 S2 T1 T2 S3	-- Unification Error
S1 S2 T2 T1 S3	-- Unification Error
S1 T1 S2 S3 T2	-- Unification Error
S1 S2 T1 S3 T2	-- Unification Error
S1 S2 T2 S3 T1	-- Display northing

1.2)

Display at Quantum Infinity

Y : Unbound
T2 : Unbound
T1 : Unbound

Display at Quantum 1

Y:3
T2 : 3
T1 : Unbound.

Explanation

- For Infinity: the main thread is being executed without anu consultation of the thread initiation statements inside; moreover, the threads are executed after Browse.

- For Quantum 1: Y is unified with X and then X is given value of 3. This transitively gives Y a value of 3. T2 is unified with value 3 with the command inside its thread. However, T1 stays unbound because (4+3) is expanded into many different statements in kernel syntax, which means that it is not finished in quantum 1. It will finish (at the expense of the others) if we expand the quantum to 4, allowing (4+3) to execute and unify.

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
```

1.4)

Because of Kernel Translation the line B = thread true end is expanded into multiple statements, so a higher quantum is needed to execute in the main thread before we run into **if**. It is expanded to probably around three statements since the minimum quantum needed to hit a suspension is 5.

Thuy Uyen My Tran
CSCI 117
07/01/2022

1.5a)

n	Fib1_sugar (seconds)	Fib1_thread (seconds)	Fib2_sugar (seconds)
10	0.48	4.51	0.02
11	1.18	8.20	0.02
12	2.86	16.90	0.02
13	7.2	32.77	0.02
14	18.54	67.22	0.02

Fib1_sugar and fib1_thread are both recursive and therefore show a common pattern of doubling every time we increase x by 1. However, while recursion in both cases creates overhead, more overhead is created with the addition of threads.

Fib2_sugar however maintains a constant time due to its iterative nature. Since it is iterative, it does not need to work its way back up.

1.5b)

n	Threads created
0	No threads created
1	No threads created
2	2
3	4
4	8
5	14
6	24
7	40
8	66

F(n) = F(n-1) + F(n-2) + 2

Part 2 – Streams

2.1)

```
local Producer OddFilter Consumer N L P F S in
  Producer = proc {$ N Limit Out}
  if (N<Limit) then T N1 in      //condition checking
    Out = (N|T)                  //set return
    N1 = (N + 1)                 //counter incrementor
    {Producer N1 Limit T}
  else Out = nil                 //complete the list with null
  end
  end

  OddFilter = proc {$ P Out}
    case P
      of nil then Out = nil      //base case
      [] '|'(1:X 2:Xr) then
      if({Mod X 2} == 0) then T in
        Out = (X|T)               //ressult
        {OddFilter Xr T}
      else
        {OddFilter Xr Out}        //get rest of list if odd case
      end
    end
  end

  // Example Testing
  N = 0                        //begin of the list
  L = 101                      //set the last element to 101 (we can only reach 100)
  {Producer N L P}             // [0 1 2 .. 100]
  {OddFilter P F}              // [0 2 4 .. 100]
  {Consumer P S}
  skip Browse F
  skip Browse S
end
```

2.2)

```
  Consumer = fun {$ P} in
    case P
      of nil then 0
      [] '|'(1:X 2:Xr) then
      (X + {Consumer Xr})
    end
  end
```

2.3)

```
local Producer OddFilter Consumer N L P F S in
  thread
  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
  end
  thread
  OddFilter = proc {$ P Out}
    case P
      of nil then Out = nil
```

```
         [] '|'(1:X 2:Xr) then
         if({Mod X 2} == 0) then T in
            Out = (X|T)
            {OddFilter Xr T}
         else
            {OddFilter Xr Out}
         end
      end
   end
   end
   thread
   Consumer = fun {$ P} in
      case P
         of nil then 0
         [] '|'(1:X 2:Xr) then
         (X + {Consumer Xr})
      end
   end
   end

   // Example Testing
   N = 0
   L = 101
   {Producer N L P} // [0 1 2 .. 100]
   {OddFilter P F}  // [0 2 4 .. 100]
   {Consumer P S}
   skip Browse F
   skip Browse S
end
```