**ASSIGNMENT - 4**

**Introduction:**

Welcome to the forth programming assignment of Fundamentals of Programming Languages.

In this homework you will be exploring CML and writing programs using a message passing style.

The homework is due on **Friday December 6th at 11:59 PM.**

Part-A: Simple Servers (25 Points)

Part-B: Fibonacci Servers (25 Points)

Part-C: Events and Mailboxes (50 Points)

Note: Please adhere to the **type signatures** that we request you to follow as you will be graded exactly based on them. If you do not follow the **type signatures** you will be assigned 0 points. You will also need to use **TextIO.print** instead of **print** in this assignment to see output in a CML program.

**Requirements and Submission:**

All files should have your name and UB email address in comments at the top of the file prior to any code you have written. Create one tarball which should be named with your UBIT id, underscore and then HW3 (example: UBITID\_HW4.tar), this tarball should contain your source files. The sources files should be called UBITID\_hw4a.sml, UBITID\_hw4b.sml, UBITID\_hw4c.sml, one fore each part of the assignment.

Submit using **submit\_cse505 <filename>**

For details on how to submit, refer to:

<https://wiki.cse.buffalo.edu/services/content/submit-script>

<https://wiki.cse.buffalo.edu/services/content/student-servers>

**Part A – Simple Servers: 25 Points**

Start your program with the following lines:

CM.make "$cml/cml.cm";

open CML;

val chan: int chan = channel();

…

fun main () = …

RunCML.doit(main, NONE);

Your task is to write two communication servers: one which generates the natural numbers, starting from 0 and going up to and including 100, and the other which prints them. This means your generating server will be sending to the printing server. The servers should communicate over the shared channel chan. The generating servers should be implemented as a recursive function called sender. The printing server should be implemented as a recursive function called receiver. The main function should spawn a thread for each server.

val sender = fn : int -> unit

val receiver = fn : unit -> 'a

val main = fn : unit -> unit

**Part B – Fibonacci Servers: 25 Points**

In this portion of the assignment you are tasked with creating a set of servers that computes the Fibonacci sequence.

CM.make "$cml/cml.cm";

open CML;

val chan1: int chan = channel();

val chan2: int chan = channel();

val chan3: int chan = channel();

You are tasked with writing the following functions with the following type signatures:

val sender1 = fn : int -> int -> unit

val sender2 = fn : int -> 'a

val receiver = fn : unit -> 'a

val main = fn : unit -> unit

There are three Fibonacci servers you will need to implement. Conceptually, two of the servers store two sequential Fibonacci numbers and the third is responsible for printing. Thus, when the program starts the first server (sender1) will start with the number 0, and the second server (sender2) will start with the number 1. The first server will send to the second its current number. The second server will calculate a new number by adding its current number and the one it received. It will send to the first server the number it previously stored and the number it received from server1 to the receiver to be printed. Notice that sender1 is curried and not only takes the first Fibonacci number but also how many Fibonacci numbers to be generated and printed by the network of servers. The main function should create threads for each of the servers.

**Part C – Events and Mailboxes: 50 Points**

In this portion of the assignment you will be tasked with creating a small mailbox abstraction. Your mailbox will be parameterized by two channels, an in channel and an out channel. The mailbox will receive on the in channel and buffer the values in a list the mailbox maintains. Other threads may consume values from the mailbox by receiving from the out channel. You will need to implement the mailbox using events. The mailbox will need to choose between sending values from its list buffer and receiving values to place on its out channel.

After implementing and testing your mailbox you should implement a stream of mailboxes, such that one mailbox receives from the out channel of the mailbox before it. You will also need to a generating function that will generate such a stream of mailboxes. The generating function will be parameterized by two channels, the in and out channels for the first mailbox, and integer dictating how many mailboxes to generate. This function should be recursive and allocate new channels. On the recursive invocation of generator the previous out channel should be the in channel for the next mailbox. A new channel should be allocated as the new out channel.

You will also write a sender and a receiver server, which will generate values to places on the mailbox stream and remove + print values taken out of the mailbox stream. The sender should be parameterized by the channel on which it should send (conceptually the in channel for the first mailbox in the stream) and a number. The sender should send this integer counting down to 0.

val mailbox = fn : 'a chan -> 'a chan -> 'a list -> 'b

val receiver = fn : int chan -> 'a

val sender = fn : int chan -> int -> unit

val generator = fn : 'a chan -> 'a chan -> int -> 'a chan

val main = fn : unit -> unit

You can test your mailbox stream with the following code:

fun main () =

let val chStart = channel()

val chEnd = generator chStart (channel()) 100

val \_ = spawn (fn () => receiver chEnd)

val \_ = spawn (fn () => ignore (sender chStart 50))

val \_ = spawn (fn () => ignore (sender chStart 50))

in ()

end;