Bag of Tasks

Homework #5, CSCI 322, Winter 2016

Due: Monday, March 7, at midnight.

Partial sums: Recall the parallel prefix calculations of section 3.5.1 in Andrews's book, and illustrated in code in Figure 3.17. We use n processes to compute the partial sums of an array of size n in $\log(n)$ time. In that context, we needed a barrier to make sure each phase completed before the next phase began.

Message passing: Rewrite this program to use message passing, so that instead of reading from a shared array, each process passes its current sum to the next with a message. Implement this in Racket using channels.

Implement a procedure of a single parameter n, which will create an array of n threads and an array of n channels. Each thread i will initialize its local number to i.

In order for process i to read the sum from process j, process j has to send the message to process i's channel, and process i has to read it. All threads should use only local variables and the array of channels.

Output: When all partial sums have been completed, print them out in order. Use message passing to coordinate this task so the sums come out in order.

Synchronous and asynchronous: Implement this both with synchronous and with asynchronous message passing. Call the two procedures `psum-synch` and `psum-asynch`.

With synchronous message passing we don't need a barrier, because no process proceeds until after all messages at each phase are complete. But you have to be careful that no phase deadlocks.

What about asynchronous message passing? Reread the text's discussion of why we need the barrier, and analyse your algorithm to see if it is a problem here. If it is, come up with a solution, and explain why your solution works. If not, explain why.

Turn in: Include a pdf document with your explanations of the barrier problem and your solution. Both the synchronous and asynchronous programs should be in a single file.