CSEE 3827: Fundamentals of Computer Systems

Project #1

Tree Map Reduce

Due 11/21/17 at 11:59 PM

1 Background on MapReduce

MapReduce is one way to structure a large computation. The data to be processed is partitioned, and a *mapper* is applied to each partition. The results of those many mapper invocations are then reduced by a *reducer* function that boils the many independent results down to a single value. For example, a large wordcount query across many documents might apply wordcount to each individual document in the map phase, and then sum the results in the reduction phase. While some computations do not lend themselves well to this particular structure, many do. The benefit, first realized by Google in 2004, is that a non-expert programmer could provide relatively simple mappers and reducers that could then be applied at scale to process large datasets. For more background, Wikipedia's page is a reasonable introduction: https://en.wikipedia.org/wiki/MapReduce.

2 Function

For this project we will perform a map-reduce computation over a *balanced binary tree.* Every node in the tree either a leaf or an internal node with exactly two children. At each leaf there is a pointer to a string.

You should implement a function, called `mapreduce`, that takes three arguments:

- a pointer to the root of the tree
- the address of the mapper function
- the address of the reducer function

It should return the single integer result of applying the mapper, then reduction function, to the given tree. Your implementation should adhere to MIPS calling conventions. It should not modify the contents of the tree in memory.

Our mappers and reducers. For the purposes of this project, mappers will be functions that take a pointer to a null-terminated ASCII string and return an integer. We have provided two mappers:

- `unit` always returns 1
- `strlen` returns the length of the given string

Our reduction functions take two integer arguments and return a single integer. We have provided three reducers:

- `sum` returns the sum of the two arguments
- `min` returns the smaller of the two arguments
- `max` returns the larger of the two arguments

Indirect function calls. In order to apply the mapper and reducer, you will need to invoke the functions by address rather than label. You will find the addresses passed to mapreduce as arguments. Calling a function whose address is in a register is a simple matter of using the "jump and link register" instruction, `jalr $t0`, $t0 being an arbitrary register chosen for this example only.

Our tree format. Tree nodes are encoded in memory as a sequence of two or three words as follows:

non-leaf:	0	<address of left child>	<address of right child>
leaf:	1	<address of string>	

3 Test Harness

In the provided `main`, you will find twelve invocations of `mapreduce` on various combinations of the provided mappers and reducers applied two two test trees (`tiny` has 2 levels, `lorem` has 6).

4 Rules and Regulations

- Place all of your code above the separator in the scaffolding. We will test your program by combining the top half of your submission with a range of test harnesses.
- Do not remove the separator from your submission. We will be using it to separate the top and bottom halves of your submission.
- Submissions must be made via courseworks.
- The uploaded .s filename should include only alphanumeric characters, hyphens, or underscores. Apart from these restrictions, any filename is fine.

5 Scoring Rubric

To be graded, your submission must adhere to all rules and regulations listed above. Submissions that loop infinitely will receive credit only for the test cases passed before the infinite loop was encountered.

We will first test your code using the mappers and reducers found in the scaffolding, but in a different order. We will then test your adherence to calling conventions by invoking your function from a *hidden caller* and, in a third test, using *hidden callees* (i.e., alternate implementations of the provided map and reduction functions).

	Total	Formula
Function	80 pts	If #Pass == 0, no tests work, 0
		If #Pass == 1, only one test passed, 20
		If #Pass >1, $20 + \lceil 60 \times \frac{\#Pass}{12} \rceil$
HiddenCaller	10 pts	Checking for clobbered registers, $\lceil 10 \times \frac{\#Pass}{\#Checked} \rceil$
HiddenCallee	10 pts	Checking for correct mapreduce results, $\lceil 10 \times \frac{\#Pass}{12} \rceil$
Style	10 pts	If functional, extra credit to ten (or more) most elegant implementations.

6 Hints

- Spim is available here: http://spimsimulator.sourceforge.net/
- You will almost certainly prefer a recursive solution.
- We strongly advising implementing and testing incrementally, for example
  - Comment out all but one mapreduce invocation from main.
  - Consider testing mapreduce on subtrees if you want something even smaller (e.g., tinyRR is a leaf node of tiny, tinyL is a tree with two leaves).
  - Implement the tree traversal only to start,
  - then actually invoke a mapper,
  - then invoke a reducer.
- You may assume the reduction function is commutative.