Module 6 and 7: Threads with Process Synchronization

In this group activity, we will continue to practice problem solving with the ADJ approach.

- Length: 50 minutes

Introduction

As previously discussed, we use the following steps for solving a design problem in SER334: Analysis, Design, and Justification.

- **Analysis:** a translation of the original ill-defined problem into concrete requirements that can be evaluated.
- **Design:** a solution to the problem being solved.
- **Justification:** an argument which supports your design as being superior to other potential designs which also satisfy the analysis requirements.

This time, we will look at a pair of problems involving threading from different perspectives. The first one features a problem that is expressed in the language of a systems developer, where we will focus on *analysis*. The is expressed as a purely algorithmic problem, and an analysis is already provided. For the second problem, you will be asked to produce a *design*: rough pseudocode for a parallel algorithm. This is much tricky than the previous OS structure examples - design here is effectively ad-hoc. This means we will need to have a definition of what a design looks, and metrics to judge designs against one another.

K

We now introduce the idea of K. K stands for **K**nowledge. K is a set of facts which we take to be the *ground truth*[1] for our world[2]. For now, take it to be everything you know... and that your group members all agree on. Defining K is an important tool in fighting ambiguity. K is a formal way of representing how the world works, and what is actually known/true. We might say things like "This problem is founded on K" to indicate the knowledge needed for a problem to be solved.

Task 0x0 (4 minutes): Getting Started

Start by forming a group of 6 ± 1 people, and reading over this document. One member of the group should start a GoogleDoc to submit your answers for for Tasks 0x1 and 0x3 below. It is not expected that answers are proof-read or well formatted, just recorded. Don't make any changes after the end of the class period. The link must be sent to the TA or instructor.

[1] *https://en.wikipedia.org/wiki/Ground_truth*
[2] If you were working at a company, what more specifically might K represent?

Task 0x1 (15 minutes): Analyzing a Problem

Web servers exist as a means to deliver content over the internet in response to HTTP requests. Given the nature of internet-provided content, anywhere from hundreds to millions of users across the world expect to be able to quickly receive said content at any time of day or night. This lends itself to multi-threading given the highly parallel nature of serving so many individuals at once. Given this background, consider the following situation:

A small startup company is attempting to determine why their web server hosting an online forum and web-IDE is not able to keep up with incoming requests. Both of these services require reading significant amounts of data from disk, and responding to potentially thousands of HTTP requests every minute. These HTTP requests produce responses whose size varies wildly (one response may just be mostly text for a forum post, while others might be large binary executables). The startup's resident system administrator claims that the issues being experienced are due to insufficient *balance* between threads on their web server, with many threads being overloaded with requests while others lie dormant.

Is the system administrator right in their conclusion? **Analyze** the problem, and from your analysis determine if the sysadmin is right or wrong. Also consider if there are parts of the provided scenario that don't matter, or if there is missing information to solve the problem and include this in your analysis.

Task 0x2 (8 minutes): Analysis Discussion

Several groups will be asked to share their answers to the questions in Task 0x1.

Task 0x3 (15 minutes) : Designing a solution

Consider the following problem statement: "*Consider parallelizing the insertion and selection sort algorithms. Which would be more amenable to parallelism?* ***Analyze*** *the problem,* ***design*** *a choice, and* ***justify*** *the choice.*" This problem is different than the previous design problem we saw (selecting an OS), because it involves creating an algorithm, rather than selecting a choice from a finite set. **See the appendix for an already completed analysis.** Given that someone has already completed the analysis portion of this problem, your group needs to propose a design solution and answer several questions (see below). Skip justification. **Groups on the east side of the room should do Q1 first, and then Q2 for selection sort. Groups on the west side of the room should do Q2 for insertion sort first, and then Q1.**

1. Questions on the *analysis*:
   (a) Why did the analysis define the two algorithms?
   (b) Why did the analysis define the word amenable?
   (c) Is there anything defined in the analysis that would be fair game to be ground truth defined by K?
   (d) Is there anything NOT defined in the analysis that would be fair game to be ground truth defined by K?
   (e) Is the assumption for integer size needed? Why or why not?
   (f) Is the assumption for known array length needed? Why or why not?
   (g) Why do the assumptions not discuss things like pthreads, mutexes, or even the number of threads?
2. What is the *design*?

Task 0x4 (8 minutes): Design Discussion

Several groups will be asked to share their answers to the questions in Task 0x3.

Appendix: Analysis for Sorting Parallelism Problem

- **Problem Definitions and Assumptions**

  To begin, we need to better define the design problem. We choose to define selection sort as shown in Algorithm 1, and insertion sort as shown in Algorithm 2. For abstraction purposes, we have chosen the algorithms to sort arrays of distinct integers with a known length. Distinct integers is not an immediately reasonable assumption, but holds without loss of generality by mapping each element in an arbitrary input to a distinct integer. Known length is a reasonable assumption, since without it, the array may be infinite, and no algorithm will terminate on it.

  Both algorithms are stated in the C programming language, and have the same basic form. Each has a parameter named a which indicates the pointer to the address of an array of integers, and a second parameter called N which stores the number of elements in the array. We assume that $N \geq 0$, that the data in a is contiguous, and that the algorithms are to be used on a platform with integers are 32-bits.

Algorithm 1 The selection sort algorithm. (Based off of code by Sedgewick.)

```
void selection_sort(int[] a, int N) {
    for (int i = 0; i < N; i++) {
        int min = i;
        for (int j = i+1; j < N; j++)
            if (a[j] < a[min]) min = j;
        exch(a, i, min);
    }
}
```

Algorithm 2 The insertion sort algorithm. (Based off of code by Sedgewick.)

```
void insertion_sort(int[] a, int N) {
    for (int i = 1; i < N; i++) {
        for (int j = i; j > 0 && a[j] < a[j-1]; j--)
            exch(a, j, j-1);
    }
}
```

We also define amenable to mean: provides a faster computation time with parallelism. The algorithm (selection sort or insertion sort) that provides the fast computation time may be said to be the one most amenable.

- **The Goal**

  A potential *design* for this problem would look like the pseudocode for a parallel sorting algorithm based on either selection sort of insertion sort. A particular design must be quantified in terms of computational speed. For a *metric*, we use the number of comparisons and exchanges as a measure of computational time needed for a particular design, expressed with tilde notation. A design is optimal (i.e., more amendable) if it of a lower order than the alternative design.

  For a particular design to *justified*, we also need to argue that each of the designs is respectively the fastest possible for that type of algorithm.

- **Observations**

  Fundamentally, data dependency is what prevents an algorithm from being parallelized. Hence, we need to look at the data access patterns as preparation for designing an algorithm.

  Selection Sort:

- The same fixed element i is compared to the set of indices which follows it ($j > i$).
  - The elements in the indices that follow i will change for every different i.
- Exchanges occur one at a time, and between the element i and some later element j ($j \geq i$).

Insertion Sort:

- Comparisons occur one at a time, and only between the element j and its neighbor $j - 1$.
- Exchanges occur one at a time, and only between the element j and its neighbor $j - 1$.
  - Exchanges are only dependent on both the values of j and its neighbor $j - 1$.