

Homework – Chapter 4

Name: _____

1. What is the difference between parallelism and concurrency?
2. You have two strategies for adding concurrency to your program (i.e. threads and fork), when should we favor fork() over threads, and when should we use threads over fork()?
3. Is it possible that multi-threading can make a program faster on a computer with only one processor? Explain.
4. Is it guaranteed that multi-threading will make a program faster on a computer with many processors? Explain.

Homework – Chapter 4

5. Give three example algorithms / types of problems for *task parallelism*, and three examples for *data parallelism*.
6. If we were migrating a program from `fork()` to threads, what replaces the `fork()` system call? What replaces the `waitpid()` system call?
7. What are the possible complications of using `fork()` in a multithreaded program?
8. Use Amdahl's law to calculate the speedup gain of an application that has 50% parallel computation, and 8 processors.

Programming Challenges

Consider the following code to read a regular expression and a list of file names from the command line (using `argc`, `argv`). It will read through each file and print out any lines that match the pattern.

Your job is to use the POSIX threads library to re-write it to make it run faster. You can do this using *task* or *data* parallelism. I don't care in what order the output appears – for example, you can read each file in its own thread. Since matching is “expensive” for complicated expressions, you could choose to do the matching in parallel. Its really up to you.

Re-Entrant Code

One challenge you will have is with code being “thread safe.” No matter how you do this, each thread will need its own regular expression (see text on thread local storage), so you will need to call `compile` in each thread or this won't work.

A Note On Complexity

My solution for this involved 21 changed lines of code, including blank lines and `{}`. I added one new `include` file, a new function, one new structure, and then modified my main function to use threads.

Homework – Chapter 4

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <regex.h>

/* compile() - compile the regular expression. This
must be done once before the matcher can be used. Only
one "matcher" can use this regular expression at one time. */
int compile(regex_t* reg, const char* pattern)
{
    return regcomp(reg, pattern, 0);
}

/* delete() - free the memory allocated during compile() */
void delete(regex_t* reg)
{
    regfree(reg);
}

/* matches() - returns true if the given line matches the regular expression.
reg - the compiled regular expression pattern, and
line - the line of text to match. */
int matches(regex_t* reg, const char* line)
{
    int rc = regexec(reg, line, 0, NULL, 0);
    if (rc == 0) return 1;
    else return 0;
}

/* handle_file - read the text, line by line,
and print those lines that match the pattern.
reg - the compiled regular expression pattern, and
line - the line of the text to match. */
void handle_file(regex_t *reg, const char* pathname)
{
    FILE* fp = fopen(pathname, "r");           // open the file
    if (fp == NULL)                          // see if it failed
    {
        fprintf(stderr, "Error - could not open file %s", strerror(errno));
        exit(-1);
    }

    char buff[1024];                          // create a buff to hold a line

    while (1) {
        memset(buff, 0, sizeof(buff));        // zero out the buff
        char *ptr = fgets(buff, sizeof(buff)-1, fp); // read the line
        if (ptr == NULL)                      // check for end of file / error
            break;                            // break if that happened

        if (matches(reg, buff)) {             // check if the line matches the pattern
            puts(buff);                       // if so, print it
        }
    }
}
```

```
/** main() - read the files identified by the argv list,
    check if they match the pattern. */
int main(int argc, char **argv)
{
    regex_t reg;
    int i;

    if (argc < 3) {
        fprintf(stderr, "error - run as %s pattern file1 ... fileN\n",
            argv[0]);
    }

    compile(&reg, argv[1]);
    for (i = 2; i < argc; i++) {
        handle_file(&reg, argv[i]);
    }
    delete(&reg);
}
```