

## CSCI 2271 Computer Systems

### Assignment 10: Threads

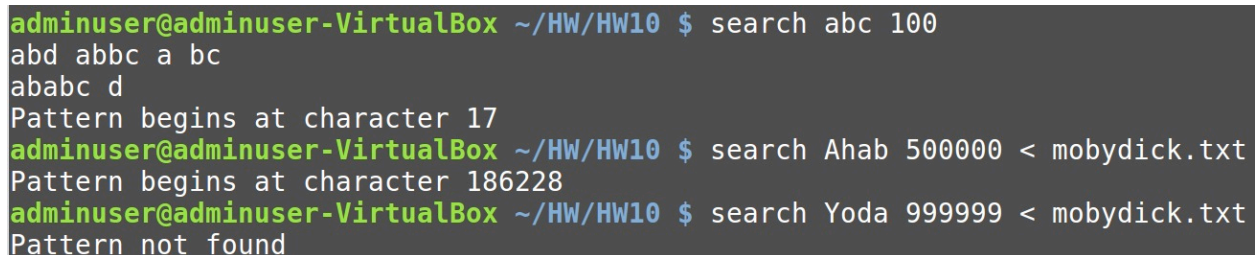
Due Monday, May 2

In this assignment you will write a program, named *search.c*, which will find the instance of a pattern string inside a text string.

#### Using the program

The program will have two command-line arguments. The first argument is the pattern string. The second argument is the size of the array that will hold the text string. Your program should then read characters from the standard input, storing them in the text array until either an EOF is found or the array has been filled. It then searches the text array for the pattern, and returns the result of that search. Note that if the text contains multiple occurrences of the pattern, the program is allowed to return the position of any one of those occurrences.

For example, the following screenshot shows my program in action.



```
adminuser@adminuser-VirtualBox ~/HW/HW10 $ search abc 100
abd abbc a bc
ababc d
Pattern begins at character 17
adminuser@adminuser-VirtualBox ~/HW/HW10 $ search Ahab 500000 < mobydic.txt
Pattern begins at character 186228
adminuser@adminuser-VirtualBox ~/HW/HW10 $ search Yoda 999999 < mobydic.txt
Pattern not found
```

In the first example of the screenshot, the pattern is "abc" and the text array has up to 100 characters. I then entered two lines of text, followed by <CTRL-d> on line 3 to denote EOF. The code found the pattern beginning at character position 17 of the text. (Note that the newline character counts as a character.)

In the second example, the pattern is "Ahab" and the text array has 500,000 characters. The input is taken from the file *mobydic.txt*.

In the third example, the pattern is "Yoda" and the text array has 999,999 characters.

#### The Structure of the Program

Let A be the array containing the N-character text string, and let B be the n-character pattern string. A simple brute-force approach is to use a single thread that executes code that looks something like this:

```

for (i=0; i<N-n; i++) {
    for (j=0; j<n; j++) {
        if (A[i+j] != B[j])
            break; //no match found at position i
        else
            j++;
    }
    if (j == n)
        return i; //a match found at position i
}
return -1; // substring not found.

```

Instead of having one thread process the entire array, we want multiple threads to work together, with each thread only having to search part of it. In particular, your program should behave like this:

The main thread should do the following:

- Allocate space for the text array and read the text characters into it.
- Create a fixed number of searcher threads. Use a `#define` statement to specify how many threads get created. Your code should work correctly regardless of how many searcher threads there are.
- Wait for all of the searcher threads to finish.
- Print the result.

Each searcher thread should do the following:

- Pick an unexamined starting position.
- See if the pattern begins at that position.
- Repeat until either:
  - the pattern is found (by this thread or another one), or
  - there are no more unexamined starting positions.

You are responsible for deciding how the threads should communicate, and what the critical sections are. Please write comments in your code explaining your design decisions.

When you are finished, submit your file *search.c* to Canvas.