

# String Manipulation

---

Rishi Dua, 2010EE50557

20 January, 2014

## 1 MANIPULATING TEXT

### 1.1 PROBLEM STATEMENT

String manipulation using shell scripts - writing a program (called NumOfOccurrences) to find how many times a substring s2 occurs in a bigger string s1

Let 'file.txt' be a text file containing string s1 of alphanumeric characters.

A substring s2 is input by the user in either of the following ways (a) either the user types s2 on the terminal or (b) from the shell prompt

**INVOCATION 1 :** \$ NumOfOccurrences s2

(only one string is given and it is understood as s2,s1 is hardcoded in the program)

**INVOCATION 2 :** \$ NumOfOccurrences s1 s2

(both strings are specified on the command line and the sequence determines which s s1 and which is s2)

Using csh, sed and awk (either separately or together - see pipes and |tee connectors to send the output of one program into another) find the total number of occurrences of a substring s2 to be entered by the user. Logically divide the string s1 into 'm' number of even partitions and parallelly search to count the total number of occurrences of s2 parallelly using threads, where 'm' is the number of threads to be entered by the user.

Let the length of string s1 [denoted as len(s1) ] be n1 and number of threads be m, then the length of each partition would be (n1/m) and n2 = len(s2) ( < (n1/m)) During runtime, print using an external subroutine StringPrint(\*char s) the 'id' of thread in which the substring s2

is found. Each thread will find the occurrence in its local string partition. IPC and message queues can be used for communication among threads, and finally print the total number of occurrences of s2 in s1.

The output of StringPrint(.) looks moreover like this:

<current UTC time> <id of thread where s2 is found> [n times]

Total embeddings of s2 in s1 = < # >

In the last line of output only the words s2 and s1 are bold (emphasized) when the sentence is printed on the terminal.

Repeated the above using C

Profile the code in Section a and compare its size with the source code in Section b.

## 1.2 ABSTRACT

String searching algorithms, sometimes called string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text.

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. Central processing units have hardware support to efficiently execute multiple threads.

POSIX Threads, usually referred to as Pthreads, is a POSIX standard for threads. The standard, POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995), defines an API for creating and manipulating threads.

## 1.3 SPECIFICATION AND ASSUMPTIONS

### **Specifications:**

Language used: Bash

Platform: Ubuntu 12.04

Additional tools used: awk, csh

Bash Version: GNU bash, version 4.2.25(1)-release (x86\_64-pc-linux-gnu)

**Problem specifications:** For C, threading libraries used: Pthreads

For Bash, since it doesn't inherently support multithreading, we just add & after the command:

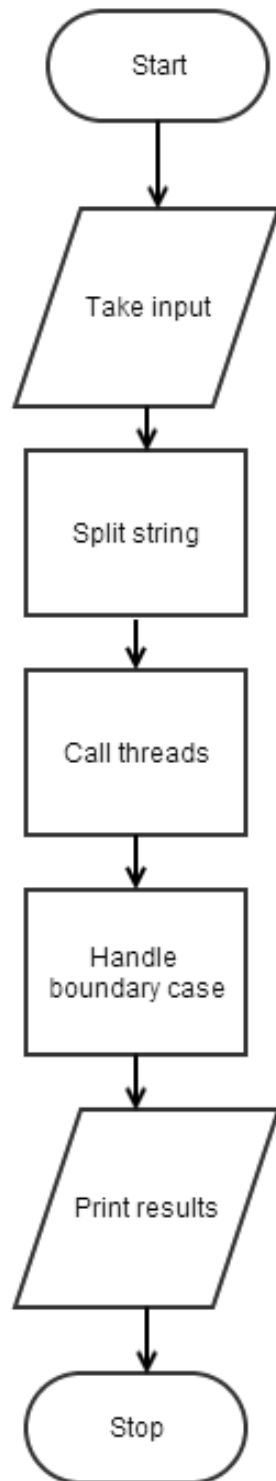
threadinstance1 &

```
threadinstance2 &  
..  
..  
threadinstancem &  
wait
```

All those jobs will then run in the background simultaneously. The wait command will then wait for all the jobs to finish.

Each command will run in a separate process, so it's technically not "multithreading", but I believe it solves your problem.

#### 1.4 FLOW CHART



## 1.5 LOGIC IMPLEMENTATION

First break the string as mentioned in the problem and execute individual steps  
Maintain a global variable to store the total. Whenever a thread finds the sub string, the variable is updated.

Wait for all the threads to finish their job. Once all the threads have terminated, we then solve separately for the boundary conditions. ie. for strings getting split into two parts.

Note: This might appear an inefficient algorithm for solving problems involving small string as the boundary case is dealt separately. For a faster implementation, the following can be implemented:

Accelerating String Matching Using Multi-Threaded Algorithm on GPU, Cheng-Hung Lin , Sheng-Yu Tsai ; Chen-Hsiung Liu ; Shih-Chieh Chang ; Shyu, J.-M.

## 1.6 EXECUTION DIRECTIVE

### For Bash:

```
./NumOfOccurrences
```

### For C:

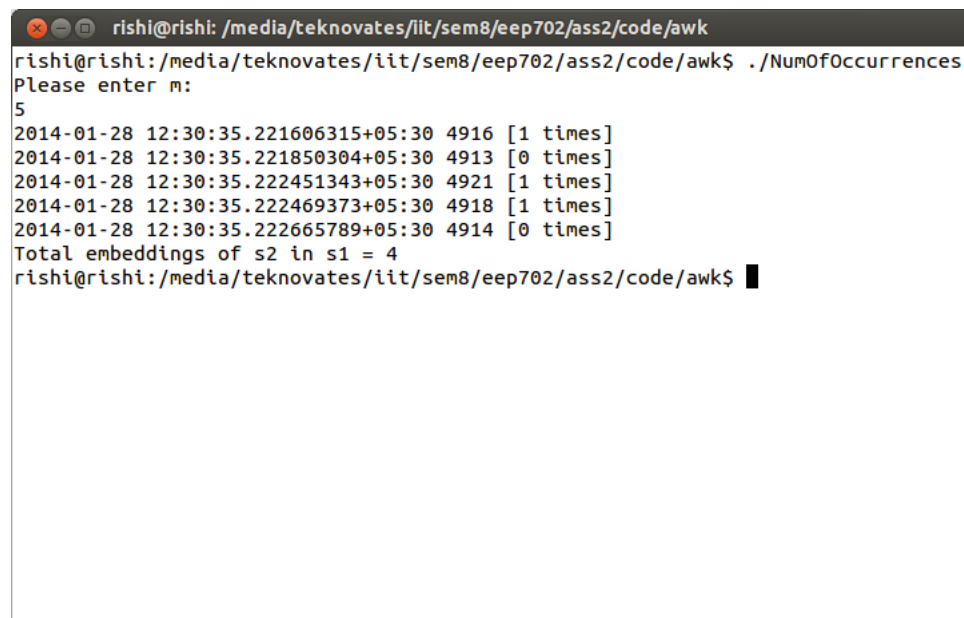
To compile

```
gcc -pthread NumOfOccurrences.c -o NumOfOccurrences
```

To run the program

```
./NumOfOccurrences
```

## 1.7 OUTPUT OF THE PROGRAM



```
rishi@rishi: /media/teknovates/iit/sem8/eep702/ass2/code/awk
rishi@rishi:/media/teknovates/iit/sem8/eep702/ass2/code/awk$ ./NumOfOccurrences
Please enter m:
5
2014-01-28 12:30:35.221606315+05:30 4916 [1 times]
2014-01-28 12:30:35.221850304+05:30 4913 [0 times]
2014-01-28 12:30:35.222451343+05:30 4921 [1 times]
2014-01-28 12:30:35.222469373+05:30 4918 [1 times]
2014-01-28 12:30:35.222665789+05:30 4914 [0 times]
Total embeddings of s2 in s1 = 4
rishi@rishi:/media/teknovates/iit/sem8/eep702/ass2/code/awk$
```

## 1.8 RESULT

The program finds how many times a string is embedded into a larger string. The code has been implemented using C Posix library and in bash using the concept of background processes.

### **Code size:**

Bash: 1,433 bytes + 214 bytes = 1647

C: 1,438 bytes

The bash code is of slightly larger size because C is a higher level language than shell scripting.

### **Profiling results**

The c code performs faster than bash code because C has a well implemented POSIX (Pthread) library. Whereas in bash, we are using background processes and waiting for them to terminate to emulate multithreading which is not a very efficient way.

The program has default values for strings which can be overwritten using command line arguments.

## 1.9 CONCLUSION

Finding the number of occurrences of a string in a larger string is a standard problem. Most languages have inbuilt functions to perform this task. We observe that the standard algorithms perform better than our for string size in ordinary use cases. However, for string larger than a few megabytes, this multithreading approach is better.

We have successfully developed a code that uses multithreaded approach to find how many times a string is embedded. The algorithm we have implemented is useful for searching in large files. Eg: searching for DNA patterns in Genome sequence.

Future work would involve efficiently handling the boundary overhead problem.