Athens university

Department of Informatics and Telecommunications

22nd Assignment - Department: Odd Registration

Numbers K22: Operating Systems - Winter Semester ΄20

Announcement Date: Friday 22 October 2020 Submission Date:

Thursday 19 November 2020 Time 23:59

## Introduction to Work:

The goal of this work is to familiarize you with system calls that create new ones, manage existing ones, and help coordinate and complete processes that all work together in a hierarchy. Your program by name myprime, will find the prime numbers that exist in a value field [ *m, n]* with the help of a flexible process tree. At work you should:

1. create a process hierarchy with the call fork (),
2. nodes at different levels of the hierarchy perform different and independently executable communicated connect to each other with piping and signals, and
3. use different system calls to achieve your computing goal
   may include calls exec * (), mkfifo (), pipe (), open (), close (), read (), write (), poll (), wait (), kill () and exit ().

The process hierarchy you create has the basic executable myprime at its root which dynamically creates internal nodes and eventually leaf nodes.

Leaf nodes search for prime numbers in specific value subfields, while internal nodes handle all prime numbers in a sorted list. The internal nodes communicate with their children and the root for the composition of the list through piping
( pipes the named-pipes).

The first numbers are found by the nodes they use independently and different
programs to do their job. Obviously processes that either compose or search must use the call exec * () to be able to load the correct executables in the space they hold in main memory.

## Procedural:

Your program should be written in *C (* the *C ++* if you want but without the use STL / Templates) and run on machines Linux of the Department.

Your source code ( source code) must consist of at least two ( and preferably more) different files and should necessarily to be used separate compilation.

Follow the course website http://www.di.uoa.gr/~ad/k22/ for additional announcements but also the electronic-list (the-list) of the course in URL https://piazza.com/uoa.gr/
fall2020 / k22 / home

?? Responsible for this exercise (questions, evaluation, grading, etc.) is Mr. Marios Papami-
halopoulos cs3190006 + AT-di, and Mr. George M. ί ΄χας cs2190024 + AT-di.

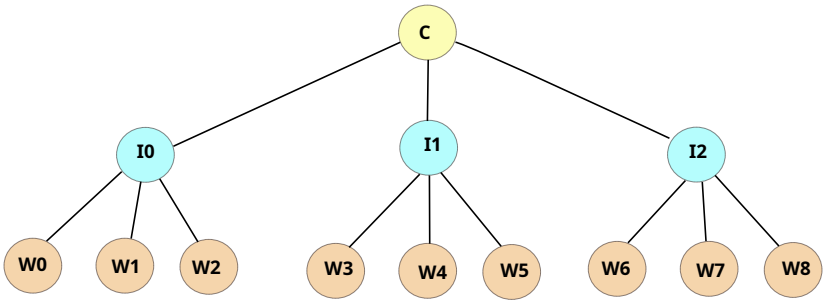


Figure 1: Example of a possible hierarchy of processes for implementation myprime where each inner node of the hierarchy creates 3 children.

## Hierarchy Composition for myprime:

Figure 1 shows a representative picture of the composition of the hierarchy we want to solve the problem of finding all prime numbers in the range [ *m, n].*

The depth of the hierarchy is always constant at 2 but each parent must have a predefined number of children defined on the command line and remain constant during an execution. Thus in Figure 1 each parent has created 3 children.

Using 3 different algorithms to find the first [1], the leaf nodes $W_0$, $W_1$, .. $W_i$, .. find prime numbers in sub-intervals specified by their parents. For every first number that they find 'send' to their parent the relevant result along with the time it took to find it. Leaf nodes should also timer the total time taken to calculate all their results. The total execution time for each procedure $W_i$ should also be shipped to the parent before the $W_i$ complete. For Figure 1 the nodes $W_0$, $W_1$, $W_2$, $W_3$, $W_4$ .. use the Algorithms respectively 1, 2, 3, 1, 2, .. Every child before completing its execution sends a signal *USR1* in root.

The processes that implement the inner-sheets take as parameters whatever information they need from the root and their main goal is:

a) to create their children and assign the sub-intervals of work as well as the right ones executable $I_k$ so that the sheets do the work,

b) through piping they should read all the results together with response times for each result derived from sheets as well as the total result for the execution time of each sex,

c) the executable running on internal nodes $I_k$ should 'compose ' the results that come from his children in ascending order and by piping to 'send' them to the root,

d) he should also be informed by each of his children about the execution time of the latter. The This information should be sent to the root for the final selection of maximum and minimum runtime statistics from all child nodes.

The root process initially creates as many internal nodes as needed, makes the right configuration-poetry and provides to each child from $I_k$ the right subfield of action that is (almost) equal for all. Finally, it receives all the results sent to it by the nodes $I_k$ and prints the following statistics:

[1] In Annex 2 we provide the codes for 2 different implementations and you are invited to offer another 3rd implementation of your choice.

1. the first numbers found in ascending order together with the time taken to find each result,

2. the minimum and maximum time it took the leaf nodes to complete their work.

3. the number of signals *USR1* which have been omitted from the root.

It is important to mention that the executables will be loaded with exec * () in intermediate nodes and nodes children are self-contained symbolized program that each presents its own functionality as described above.

The results will be displayed in tty from which it is called myprime. Note that all procedures should work simultaneously and it is not *blocking.*

## Application Dial Bar:

The application can be called in the following strict way on the command line ( tty):

. / myprime -l lb -u ub -w NumofChildren

where

?? myprime is the executable you create,
?? lb is the minimum number that will be checked if it is prime,
?? ub is the maximum number that will be checked if it is prime,
?? NumofChildren is the specific fixed number of children that the root and the intermediate nodes create to load independently executable.

The above in-line parameters (and corresponding flags) are required for their call to the command line. Flags can be displayed with any order. The exact expected output format for the program is given by detail on the course page.

## Program Features You Must Write:

1. You can not do pre-allocate any space since structure (s) should be able to grow with virtually no limit on the number of records they can store. Using static tables / structures that are bound during the symbol translation of your program is not an acceptable option.

2. For any design choice you adopt, you must justify it in your report.

3. Before the execution of the application is terminated, the contents of the structures are released in a gradual and controlled manner.
4. If any parts of your code are likely to come from a public source, you should to give reference to that source whether it is a book, notes, Internet URL etc. and explain exactly how you used that reference.
5. You have complete freedom to choose the way in which you will eventually implement auxiliary structures.

## What to Deliver:

1. A brief and comprehensive explanation of the choices you have made in designing your program (2-3 pages in ASCII text are sufficient).
2. Definitely one Makefile ( which can be used to do it automatically compile of your program). More details about ( Makefile) and how this is created are given on the course website.

3. One tar-file with all your work in a directory that probably bears your name and will contains all your work ie. source files, header files, output files ( if any), and your report.

## ΣηOther Important Remarks:

1. The tasks are individual.
2. Your program should run on Linux systems of the department otherwise can not grade confessed.
3. Although you are expected to discuss with friends and colleagues how you will try to solve the problem, copying code (in any form) is something that not allowed and should not be done. Anyone found involved in copying code it just gets zero in class. This applies to those involved independently from who gave / took etc.

4. The above applies if found even partial ignorance of the code you have submitted or it just exists suspicion that the code is a product of a transaction with a third party (s).
5. Programs that do not use separate compilation they lose automatically 10% of degree.
6. In no case the Windows is not eligible platform for presenting this exercise.

## ANNEX 1: Time Measurement at Linux

```
# include <stdio.h>                    /* printf () */
# include <sys / times.h>              /* times () */
# include <unistd.h>                   /* sysconf () */

int main ( void ) {
        double t1 , t2 , cpu_time ;
        struct tms tb1 , tb2 ;
        double ticspersec ;
        int            i , sum = 0;

        ticspersec = ( double ) sysconf ( _SC_CLK_TCK );
        t1 = ( double ) times (& tb1 );
        for ( i = 0; i <100000000; i ++)
                        sum + = i ;
        t2 = ( double ) times (& tb2 );
        cpu_time = ( double ) (( tb2 . tms_utime + tb2 . tms_stime ) -
                                        ( tb1 . tms_utime + tb1 . tms_stime ));
        printf ( "Run time was% lf sec (REAL time) although we used the CPU for% lf sec (CPU time)
                . \ n " , ( t2 - t1 ) / ticspersec , cpu_time / ticspersec );
}
```

## ANNEX 2: Algorithms for Finding Prime Numbers

### Method 1 for Finding Prime Numbers

```
# include <stdio.h>
# include <stdlib.h>

# define YES 1
# define NO          0

int prime ( int n ) {
                int i ;
                if ( n == 1) return ( NO );
                for ( i = 2; i < n ; i ++)
                                if ( n % i == 0) return ( NO );
                return ( YES );
}
```

```
int main ( int argc , char * argv []) {
        int lb = 0, ub = 0, i = 0;

        if (( argc ! = 3)) {
                printf ( "usage: prime1 lb ub \ n" );
                exit (1); }

        lb = atoi ( argv [1]);
        ub = atoi ( argv [2]);

        if (( lb <1)                    | | ( lb > ub )) {
                printf ( "usage: prime1 lb ub \ n" );
                exit (1); }

        for ( i = lb ; i <= ub ; i ++)
                if ( prime ( i ) == YES )
                        printf ( "% d" , i );
        printf ( "\ n" );
}
```

## Method 2 for Finding Prime Numbers

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>

# define YES 1
# define NO          0

int prime ( int n ) {
        int i = 0, limitup = 0;
        limitup = ( int ) ( sqrt (( float ) n ));

        if ( n == 1) return ( NO );
        for ( i = 2; i <= limitup ; i ++)
                if ( n % i == 0) return ( NO );
        return ( YES );
}

int main ( int argc , char * argv []) {
        int lb = 0, ub = 0, i = 0;

        if (( argc ! = 3)) {
                printf ( "usage: prime1 lb ub \ n" );
                exit (1); }

        lb = atoi ( argv [1]);
        ub = atoi ( argv [2]);

        if (( lb <1)                    | | ( lb > ub )) {
                printf ( "usage: prime1 lb ub \ n" );
                exit (1); }

        for ( i = lb ; i <= ub ; i ++)
                if ( prime ( i ) == YES )
                        printf ( "% d" , i );
        printf ( "\ n" );
}
```