ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
22η Εργασία - Τμήμα: Περιττών Αριθμών Μητρώου
K22: Λειτουργικά Συστήματα – Χειμερινό Εξάμηνο '20
Ημερομηνία Ανακοίνωσης: Παρασκευή 22 Οκτωβρίου 2020
Ημερομηνία Υποβολής: Πέμπτη 19 Νοεμβρίου 2020 Ώρα 23:59

Εισαγωγή στην Εργασία:

Ο στόχος αυτής της εργασίας είναι να εξοικειωθείτε με κλήσεις συστήματος που δημιουργούν νέες, διαχειρίζονται υπάρχουσες και βοηθούν στον συντονισμό και την ολοκλήρωση διεργασιών που όλες μαζί λειτουργούν ταυτόχρονα σε μια ιεραρχία. Το πρόγραμμα σας με όνομα `myprime`, θα βρίσκει τους πρώτους αριθμούς που υπάρχουν σε ένα πεδίο τιμών $[m, n]$ με την βοήθεια ενός *ευέλικτου δένδρου διεργασιών*. Στην εργασία θα πρέπει:

1. να δημιουργήσετε μια ιεραρχία διεργασιών με την κλήση `fork()`,
2. κόμβοι σε διαφορετικά επίπεδα της ιεραρχίας εκτελούν *διαφορετικά* και *ανεξάρτητα εκτελέσιμα* που επικοινωνούν μεταξύ τους με σωληνώσεις και σήματα, και
3. να χρησιμοποιήσετε διάφορες κλήσεις συστήματος για να επιτύχετε τον υπολογιστικό σας στόχο που μπορεί να συμπεριλαμβάνουν τις κλήσεις `exec*()`, `mkfifo()`, `pipe()`, `open()`, `close()`, `read()`, `write()`, `poll()`, `wait()`, `kill()` και `exit()`.

Η ιεραρχία διεργασιών που θα δημιουργήσετε έχει το βασικό εκτελέσιμο `myprime` στην ρίζα του η οποία δυναμικά δημιουργεί εσωτερικούς κόμβους και τελικά κόμβους-φύλλα.

Οι κόμβοι-φύλλα αναλαμβάνουν την αναζήτηση πρώτων αριθμών σε συγκεκριμένα υπό–πεδία τιμών, ενώ οι εσωτερικοί κόμβοι αναλαμβάνουν την σύνθεση όλων των πρώτων αριθμών σε μια ταξινομημένη λίστα. Οι εσωτερικοί κόμβοι επικοινωνούν με τα παιδιά τους και την ρίζα για την σύνθεση της λίστα μέσω σωληνώσεων (pipes ή named-pipes).

Η εύρεση των πρώτων αριθμών γίνεται από τους κόμβους φύλλα που χρησιμοποιούν *ανεξάρτητα* και *διαφορετικά* προγράμματα για να κάνουν την δουλειά τους. Προφανώς οι διεργασίες που κάνουν είτε σύνθεση είτε αναζήτηση πρέπει να χρησιμοποιούν την κλήση `exec*()` για να μπορέσουν να φορτώσουν τα σωστά εκτελέσιμα στο χώρο που κατέχουν στην κυρίως μνήμη.

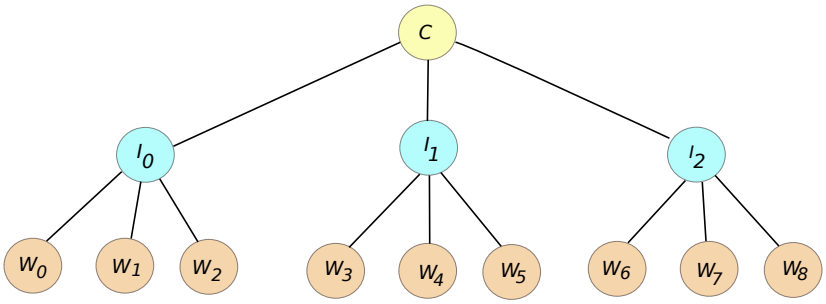
Διαδικαστικά:

Το πρόγραμμα σας θα πρέπει να γραφτεί σε *C* (ή *C++* αν θέλετε αλλά χωρίς την χρήση STL/Templates) και να τρέχει στις μηχανές Linux του τμήματος.

Ο πηγαίος κώδικας σας (source code) πρέπει να αποτελείται από ***τουλάχιστον δυο*** (και κατά προτίμηση πιο πολλά) διαφορετικά αρχεία και θα πρέπει *απαραιτήτως να γίνεται χρήση* separate compilation .

Παρακολουθείτε την ιστοσελίδα του μαθήματος `http://www.di.uoa.gr/~ad/k22/` για επιπρόσθετες ανακοινώσεις αλλά και την ηλεκτρονική-λίστα (η-λίστα) του μαθήματος στο URL `https://piazza.com/uoa.gr/fall2020/k22/home`

- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση κλπ.) είναι ο κ. Μάριος Παπαμιχαλόπουλος `cs3190006+AT-di`, και ο κ. Γιώργος Μίχας `cs2190024+AT-di`.



Σχήμα 1: Παράδειγμα πιθανής ιεραρχίας διεργασιών για την εφαρμογή `myprime` όπου κάθε εσωτερικός κόμβος της ιεραρχίας δημιουργεί 3 παιδιά.

Σύνθεση Ιεραρχίας για το `myprime`:

Το Σχήμα 1 δείχνει μια αντιπροσωπευτική εικόνα για την σύνθεση της ιεραρχίας που επιθυμούμε να λύσει το πρόβλημα του να βρούμε όλους τους **πρώτους αριθμούς στο εύρος** $[m, n]$.

Το βάθος της ιεραρχίας είναι πάντα σταθερό στο 2 αλλά κάθε γονιός πρέπει να έχει ένα προκαθορισμένο αριθμό από παιδιά που ορίζεται στην γραμμή εντολής και παραμένει σταθερός στην διάρκεια μιας εκτέλεσης. Έτσι στο Σχήμα 1 κάθε γονιός έχει δημιουργήσει 3 παιδιά.

Κάνοντας χρήση 3 διαφορετικών αλγορίθμων για την εύρεση πρώτων [1], οι κόμβοι-φύλλα $W_0, W_1, ..W_i, ..$ βρίσκουν πρώτους αριθμούς σε υπό–διαστήματα που ορίζουν οι γονείς τους. Για κάθε πρώτο αριθμό που βρίσκουν 'στέλνουν' στο γονιό τους το σχετικό αποτέλεσμα μαζί με το χρόνο που χρειάστηκε για βρεθεί. Οι κόμβοι-φύλλα θα πρέπει επίσης να χρονομετρούν το συνολικό χρόνο που τους πήρε για να υπολογίσουν όλα τα αποτελέσματα τους. Ο συνολικός χρόνος εκτέλεσης για κάθε διαδικασία W_i θα πρέπει επίσης να αποστέλλεται στο γονιό πριν η W_i ολοκληρώσει. Για το Σχήμα 1 οι κόμβοι $W_0, W_1, W_2, W_3, W_4..$ χρησιμοποιούν αντίστοιχα τους Αλγορίθμους $1, 2, 3, 1, 2, ..$ Κάθε παιδί *πριν ολοκληρώσει την εκτέλεση του* στέλνει ένα σήμα *USR1* στην ρίζα.

Οι διεργασίες που υλοποιούν τα εσωτερικά-φύλλα λαμβάνουν σαν παραμέτρους οποιαδήποτε πληροφορία χρειάζονται από τη ρίζα και ο βασικός τους στόχος είναι:

α) να δημιουργήσουν τα παιδιά τους και να αναθέσουν τα υπο-διαστήματα εργασίας καθώς και τα ορθά εκτελέσιμα I_k ώστε τα φύλλα να κάνουν την δουλειά,

β) μέσω σωληνώσεων θα πρέπει να διαβάζουν όλα τα αποτελέσματα μαζί με χρόνους απόκρισης για κάθε αποτέλεσμα που προέρχεται από φύλλα όπως και το συνολικό αποτέλεσμα για την χρόνο εκτέλεσης κάθε φύλου,

γ) το εκτελέσιμο που τρέχει σε εσωτερικούς κόμβους I_k θα πρέπει να 'συνθέτει' τα αποτελέσματα που προέρχονται από τα παιδιά του σε αύξουσα σειρά και με σωλήνωση να τα 'στέλνει' στην ρίζα,

δ) επίσης θα πρέπει να ενημερώνεται από κάθε παιδί του για τον χρόνο εκτέλεσης του τελευταίου. Οι πληροφορίες αυτές θα πρέπει να στέλνονται στην ρίζα για την τελική επιλογή στατιστικών μέγιστου και ελάχιστου χρόνου εκτέλεσης από όλους τους κόμβους-παιδιά.

Η διαδικασία ρίζας αρχικά δημιουργεί όσους εσωτερικούς κόμβους χρειάζονται, κάνει την σωστή παραμετροποίηση και παρέχει σε κάθε παιδί από τα I_k το ορθό υπο-πεδίο δράσης που να είναι (σχεδόν) ίσον για όλους. Τέλος παραλαμβάνει όλα τα αποτελέσματα που του στέλνουν οι κόμβοι I_k και τυπώνει τα εξής στατιστικά:

[1] Στο Παράρτημα 2 παρέχουμε τους κωδικες για 2 διαφορετικές υλοποιήσεις και καλείστε να προσφέρετε και μια ακόμα 3η υλοποίηση δικιάς σας επιλογής.

1. τους πρώτους αριθμούς που βρέθηκαν σε αύξουσα σειρά μαζί με την διάρκεια που πήρε για να βρεθεί κάθε αποτέλεσμα,
2. τον ελάχιστο και μέγιστο χρόνο που χρειάστηκαν οι κόμβοι-φύλλα για να ολοκληρώσουν την εργασία τους.
3. τον αριθμό των σημάτων *USR1* που έχουν παραληφθεί από την ρίζα.

Είναι σημαντικό να αναφερθεί οτι τα εκτελεσιμα που θα φορτωθούν με `exec*()` σε ενδιάμεσους κόμβους και κόμβους παιδία είναι αυτοδύναμα συμβολομεταφρασμένα προγράμμα που το κάθε ένα παρουσιάζει την δικιά του λειτουργικότητα όπως περιγράφεται παραπάνω.

Τα αποτελέσματα θα εμφανιστούν στο `tty` από το οποίο καλείται το `myprime`. Να σημειώσουμε ότι όλες οι διαδικασίες θα πρέπει να *δουλεύουν ταυτόχρονα* και δεν είναι *blocking*.

Γραμμή Κλήσης της Εφαρμογής:

Η εφαρμογή μπορεί να κληθεί με τον παρακάτω αυστηρό τρόπο στην γραμμή εντολής (`tty`):
./myprime -l lb -u ub -w NumofChildren
όπου

- myprime είναι το εκτελέσιμο που θα δημιουργήσετε,
- lb είναι ο ελάχιστος αριθμός που θα ελεγχθεί αν είναι πρώτος,
- ub είναι ο μέγιστος αριθμός που που θα ελεγχθεί αν είναι πρώτος,
- NumofChildren είναι ο συγκεκριμένος σταθερός αριθμός των παιδιών που η ρίζα και οι ενδιάμεσοι κόμβοι δημιουργούν για να φορτώσουν ανεξάρτητα εκτελέσιμα.

Οι παραπάνω in-line παράμετροι (και αντίστοιχες σημαίες) είναι υποχρεωτικές όσον αφορά την κλήση τους στην γραμμή εντολής. Οι σημαίες μπορούν να εμφανίζονται με *οποιαδήποτε σειρά*. Η ακριβής αναμενόμενη *μορφή εξόδου* για το πρόγραμμα δίνεται με *λεπτομέρεια* στην σελίδα του μαθήματος.

Χαρακτηριστικά του Προγράμματος που Πρέπει να Γράψετε:

1. Δεν μπορείτε να κάνετε pre-allocate οποιοδήποτε χώρο αφού δομή(-ές) θα πρέπει να μπορεί(-ουν) να μεγαλώσει(-ουν) χωρίς ουσιαστικά κανέναν περιορισμό όσον αφορά στον αριθμό των εγγραφών που μπορούν να αποθηκεύσουν. Η χρήση στατικών πινάκων/δομών που δεσμεύονται στην διάρκεια της συμβολομετάφρασης του προγράμματος σας δεν είναι αποδεκτές επιλογές.
2. Για οποιαδήποτε σχεδιαστική επιλογή που θα υιοθετήσετε, θα πρέπει να να την δικαιολογήσετε στην αναφορά σας.
3. Πριν να τερματίσει η εκτέλεση της εφαρμογής, το περιεχόμενο των δομών απελευθερώνεται με σταδιακό και ελεγχόμενο τρόπο.
4. Αν πιθανώς κάποια κομμάτια του κωδικά σας προέλθουν από κάποια δημόσια πηγή, θα πρέπει *να δώσετε αναφορά στη εν λόγω πηγή* είτε αυτή είναι βιβλίο, σημειώσεις, Internet URL κλπ. και να εξηγήσετε πως ακριβώς χρησιμοποιήσατε την εν λόγω αναφορά.
5. Έχετε πλήρη ελευθερία να επιλέξετε το τρόπο με τον οποίο τελικά θα υλοποιήσετε βοηθητικές δομές.

Τι πρέπει να Παραδοθεί:

1. Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματος σας (2-3 σελίδες σε ASCII κειμένου είναι αρκετές).
2. Οπωσδήποτε ένα `Makefile` (που να μπορεί να χρησιμοποιηθεί για να γίνει αυτόματα το compile του προγράμματος σας). Πιο πολλές λεπτομέρειες για το (`Makefile`) και πως αυτό δημιουργείται δίνονται στην ιστοσελίδα του μαθήματος.

3. Ένα `tar-file` με όλη σας την δουλειά σε έναν κατάλογο που πιθανώς να φέρει το όνομα σας και θα περιέχει όλη σας την δουλειά δηλ. `source files, header files, output files` (αν υπάρχουν), και η αναφορά σας.

Άλλες Σημαντικές Παρατηρήσεις:

1. Οι εργασίες είναι **ατομικές**.
2. Το πρόγραμμα σας θα πρέπει να τρέχει στα `Linux` συστήματα του τμήματος αλλιώς **δεν μπορεί να βαθμολογηθεί**.
3. Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πως θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιαδήποτε μορφής) είναι κάτι που **δεν επιτρέπεται** και δεν πρέπει να γίνει. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικά **απλά παίρνει μηδέν** στο μάθημα. Αυτό ισχύει για **όσους εμπλέκονται ανεξάρτητα** από το ποιος έδωσε/πήρε κλπ.
4. Το παραπάνω ισχύει αν διαπιστωθεί *έστω και μερική άγνοια* του κώδικα που έχετε υποβάλει ή *άπλα υπάρχει υποψία* ότι ο κώδικας είναι προϊόν συναλλαγής με τρίτο/-α άτομο/α.
5. Προγράμματα που **δεν χρησιμοποιούν** separate compilation χάνουν αυτόματα 10% του βαθμού.
6. Σε καμιά περίπτωση τα Windows *δεν είναι επιλέξιμη* πλατφόρμα για την παρουσίαση αυτής της άσκησης.

ΠΑΡΑΡΤΗΜΑ 1: Μέτρηση Χρόνου στο Linux

```c
#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);
}
```

ΠΑΡΑΡΤΗΜΑ 2: Αλγόριθμοι για την Εύρεση Πρώτων Αριθμών

Μέθοδος 1 για Εύρεση Πρώτων Αριθμών

```c
#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");
}
```

Μέθοδος 2 για Εύρεση Πρώτων Αριθμών

```
#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");
}
```