ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
2η Εργασία - Τμήμα Αρτίων Αριθμών Μητρώου
K22: Λειτουργικά Συστήματα – Χειμερινό Εξάμηνο '15
Ημερομηνία Ανακοίνωσης: 7 Νοεμβρίου
Ημερομηνία Υποβολής: 30 Νοεμβρίου και Ώρα 23:59

Εισαγωγή στην Εργασία:

Ο στόχος αυτής της εργασίας είναι να εξοικειωθείτε με την δημιουργία *ιεραρχιών από διεργασίες* χρησιμοποιώντας την κλήση συστήματος fork() που σε συνδυασμό με την κλήση exec*() μπορεί να βοηθήσει να επιτύχουμε διαφοροποίηση και επιμερισμό εργασίας.

Το αρχικό σας πρόγραμμα μαζί με όλα τα παιδιά διεργασίες που θα δημιουργήσετε αποτελούν μία ιεραρχική δομή που στόχο έχει να καταμετρά τους ψήφους (δηλ. tally) που οι κάτοικοι μιας χώρας έχουν καταθέσει στα πλαίσια εκλογής προέδρου της δημοκρατίας. Η δενδρική δομή ταυτόχρονων προγραμμάτων διαθέτει το αρχικό σας πρόγραμμα (*root*), εσωτερικούς κόμβους που ονομάζονται *splitters/mergers* και τέλος κόμβους φύλλα που ονομάζονται *sorters*. Το δένδρο βοηθά ώστε να καταμετρηθούν σωστά οι ψήφοι των πολιτών οι οποίοι εμπεριέχονται σε ένα ASCII κείμενο τυχαίου μεγέθους (που μπορεί να είναι και εξαιρετικά μεγάλο). Στο εν λόγω αρχείο, η κάθε γραμμή παρουσιάζει την ανώνυμη επιλογή ενός πολίτη.

Η ιεραρχία διασπά την καταμέτρηση σε πολλαπλές μικρότερες δουλειές και κατόπιν προχωρά στην σύνθεση των αποτελεσμάτων αλλά και στατιστικών που απαιτούνται. Στο τέλος, η εφαρμογή σας παρουσιάζει τα αποτελέσματα.

Ο τρόπος με τον οποίο επιλύουμε το πρόβλημα καταμέτρησης ψήφων για την εκλογή προέδρου της Δημοκρατίας ακολουθεί την λογική του «διαίρει και κατέκτησε»: οι εσωτερικοί κόμβοι (*splitters/mergers*) αναθέτουν εργασία είτε σε υποκείμενους (*splitters/mergers* ή σε *sorters*). Από κάθε κόμβο διεργασίας μπορούν να δημιουργηθούν l παιδιά–διεργασίες. Οι εργάτες (*sorters*) εμφανίζονται στο τελευταίο επίπεδο d της ιεραρχίας και είναι οι κατά βάσει υπεύθυνοι για την καταμέτρησή ψήφων από το τμήμα του αρχείου που τους αναλογεί. Τα αποτελέσματα περνούν από τους εργάτες στους γονείς τους όπου γίνεται σύνθεση αποτελεσμάτων και οι τελευταίοι επαναλαμβάνουν την ρουτίνα μέχρις έως ότου τα αποτελέσματα να 'ταξιδέψουν' στον κόμβο *root*.

Για τον παραπάνω σκοπό, οι ενδιάμεσοι κόμβοι και τα φύλλα της ιεραρχίας χρησιμοποιούν ανεξάρτητα (και διαφορετικά) προγράμματα για να επιτύχουν τον επιμέρους σκοπό τους. Οι επικοινωνίες μεταξύ κόμβων (δηλ. διεργασιών) σε διάφορα επίπεδα γίνεται με την χρήση είτε απλών pipes ή named-pipes (FIFOs).

Σε αυτή την άσκηση:

1. θα δημιουργήσετε μια ιεραρχία διεργασιών χρησιμοποιώντας fork(),
2. θα εκτελέσετε διαφοροποιημένα κομμάτια κώδικα ή και ανεξάρτητα προγράμματα από τους κόμβους της ιεραρχίας, και
3. θα χρησιμοποιήσετε διάφορες κλήσεις συστήματος περιλαμβανομένων και των read(), write(), dup(), dup2(), mkfifo(), wait(), times(), κλπ.

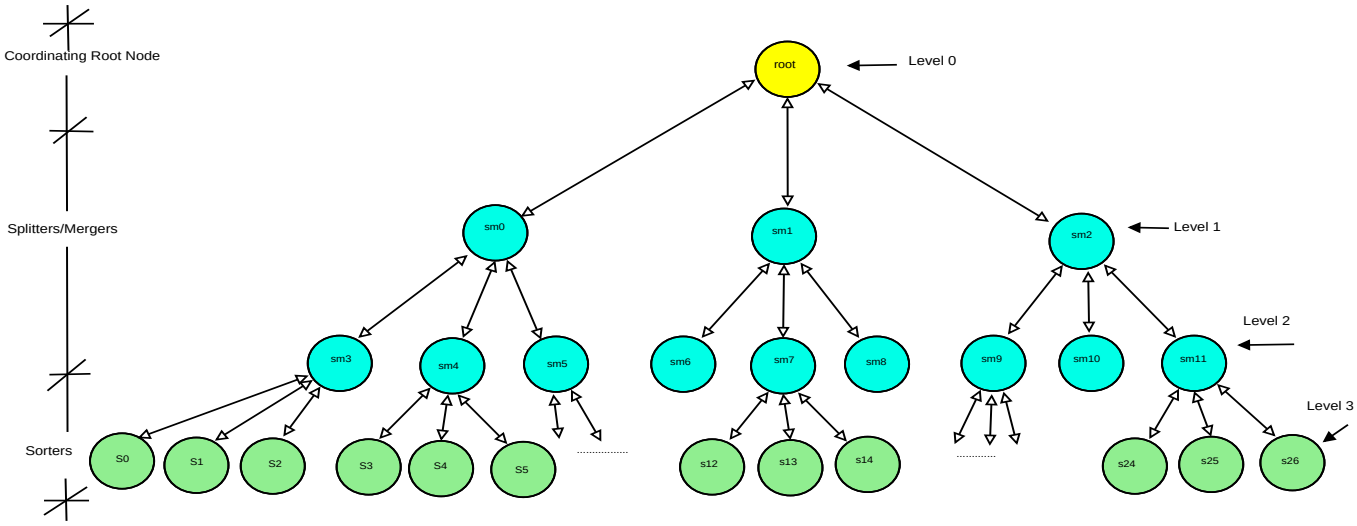
Διαδικαστικά:

Το πρόγραμμά 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. Παρακολουθείτε την ιστοσελίδα του μαθήματος για επιπρόσθετες ανακοινώσεις στο URL `http://www.di.uoa.gr/~ad/k22/`.

- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση κλπ.) είναι οι κ. Ευάγγελος Νικολόπουλος `e.nikolopoulos+AT-di` και Δημήτρης Μήλιος `dmilios+AT-di`.
- Μέσα από την σελίδα `https://piazza.com/uoa.gr/fall2015/k22sectiondelis/home` θα μπορείτε να κάνετε ερωτήσεις και να δείτε απαντήσεις ή/και διευκρινήσεις που δίνονται σχετικά με την άσκηση (όπως έγινε με την πρώτη άσκηση).

Η Ιεραρχία Διεργασιών:

Το Σχήμα 1 δείχνει ένα παράδειγμα ιεραρχίας διεργασιών που θα πρέπει το πρόγραμμά σας να δημιουργήσει. Ο αριθμός των διαδικασιών-παιδιών που κάθε κόμβος μπορεί να δημιουργήσει (children) παραμένει σταθερό και δίνεται σαν παράμετρος (l) στο πρόγραμμά σας. Το Σχήμα 1 δίνει ένα τέτοιο παράδειγμα με ένα τριαδικό-δένδρο (turnary tree) όπου κάθε κόμβος έχει τρεις διαδικασίες-παιδιά. Ο κάθε κόμβος τρέχει ένα δικό του πρόγραμμα και επιτελεί μια επί μέρους δουλειά στην καταμέτρηση των ψήφων. Το βάθος του δένδρου των διαδικασιών ορίζεται από την παράμετρο d που επίσης παρέχεται στην γραμμή εντολής. Ο *root* κόμβος παρέχει στους ενδιάμεσους



Σχήμα 1: Ιεραρχία Διεργασιών με l=3 και βάθος d=3. Ο κάθε εργάτης (*worker*) δουλεύει με το 1/27 του αρχείου των ψήφων.

splitters/mergers το εύρος γραμμών μέσα σε ένα αρχείο που οι τελευταίοι θα πρέπει να επεξεργαστούν και να δημιουργήσουν επί μέρους καταμετρήσεις (δηλ. για το συγκεκριμένο εύρος ψήφων ποιος είναι ο αριθμός ψήφων που έχει πάρει ο κάθε υποψήφιος).

Οι ενδιάμεσοι κόμβοι του δένδρου έχουν ένα πολλαπλό ρόλο: 1) σπάνε το πρόβλημα της καταμέτρησης σε μικρότερα προβλήματα (splitting) και το προωθούν στους πιο κάτω κόμβους. Αυτό επαναλαμβάνεται μέχρι να φτάσουμε στο επίπεδο φύλλων (sorters), 2) προχωρούν στην σύνθεση των (μερικών) αποτελεσμάτων (merging) τα οποία προέρχονται από υποκείμενους κόμβους, και 3) προωθούν τα αποτελέσματα που έχουν συνθέσει στην διεργασία που είναι ή μητέρα τους.

Οι κόμβοι-φύλλα ουσιαστικά ταξινομούν τους υποψηφίους και καταμετρούν τους ψήφους που κάθε υποψήφιος έχει λάβει. Μπορούν φυσικά, και *θα πρέπει* να κάνουν και οποιαδήποτε άλλη δουλειά 'ελαφρύνει' την εργασία της ρίζας.

Εν γένει, το πρόβλημα της καταμέτρησης «μεταφέρεται σταδιακά» στους κόμβους-φύλλα, όπου γίνεται η ουσια-

στική εργασία καταγραφής αποτελεσμάτων. Τα αποτελέσματα αυτά με την σειρά τους και επίσης με σταδιακό τρόπο «ανεβαίνουν» στην ιεραρχία μέχρι να φτάσουν στην ρίζα. Η ρίζα σαν διεργασία είναι υπεύθυνη για την τελική καταμέτρηση και την παρουσίαση των αποτελεσμάτων είτε σε μορφή απλής εξόδου κειμένου ή χρησιμοποιώντας ένα γράφημα.

Η ρίζα, οι εσωτερικοί κόμβοι και τα φύλλα λειτουργούν με την βοήθειά τριών αυτόνομων, διαφορετικών προγραμμάτων. Κάθε φορά που υπάρχει ανάγκη να δημιουργηθεί ένα παιδί στην ιεραρχία μία κλήση fork() θα πρέπει να εκτελεστεί. Εάν ο χώρος μια διαδικασίας πρέπει να αντικατασταθεί με ένα άλλο εκτελέσιμο τότε αυτό επιτυγχάνεται με την κλήση μια exec*() –της δικιά σας επιλογής– που έχει και την *σχετική λίστα παραμέτρων*. Η λίστα αυτή μπορεί να περιέχει *ότι πληροφορία είναι χρήσιμη* για να δουλέψουν με επιτυχία οι κόμβοι τύπου splitter/merger (ενδιάμεσοι κόμβοι) και sorter (κόμβοι φύλλα).

Τα παρακάτω σημεία δίνουν μια περιγραφή της λειτουργίας των διαφορετικών τύπων κόμβων και των ανεξαρτήτων προγραμμάτων που θα πρέπει να δημιουργήσετε ώστε να επιτευχθεί ο στόχος της άσκησης:

1. Οι τρεις τύποι κόμβων που αποτελούν την ιεραρχία εκτελούν ανεξάρτητα αλλά συνεργαζόμενα προγράμματα.

2. Ο κόμβος ρίζας (root) –δηλ. το βασικό σας πρόγραμμα– λειτουργεί σαν 'άγκυρα' για όλη την ιεραρχία και διεκπεραιώνει την τελική παρουσίαση των αποτελεσμάτων.

3. Σύμφωνα με τον αριθμό l που έχει καθοριστεί στην γραμμή κλήσης του κυρίου προγράμματος, η ρίζα δημιουργεί l splitters/mergers. Ο κάθε splitter/merger παίρνει σαν είσοδο το $1/l$ των γραμμών του αρχείου κειμένου που θα πρέπει να επεξεργαστούμε. Η επεξεργασία που επιτελεί ο κάθε splitter/merger είναι η υποδιαίρεση της εργασίας, η προώθηση προς τα «κάτω», η συλλογή και σύνθεση αποτελεσμάτων από τους υποκείμενους κόμβους και τέλος η προώθηση των αποτελεσμάτων σε κόμβο γονιό. Οι ενδιάμεσοι κόμβοι βρίσκονται στα επίπεδα 1 μέχρι $d-1$ του δένδρου διεργασιών και όλοι επιτελούν την παραπάνω επεξεργασία.

4. Ο κάθε splitter/merger θα πρέπει να ανοίξει διόδους επικοινωνίας με *όλους* τους υποκείμενους κόμβους αλλά και τον υπερκείμενο κόμβο ώστε να περάσει (η να λάβει) σχετικές πληροφορίες/αποτελέσματα. Οι δίοδοι αυτοί υλοποιούνται είτε με απλά pipes ή με named pipes.

5. Όταν ένας sorter ολοκληρώσει την εργασία του στέλνει ένα ***USR1*** σήμα [1] ειδοποιώντας έτσι την ρίζα ότι έχει ολοκληρώσει την εργασία που του αναλογεί. Αν η ρίζα λάβει l^d τέτοια σήματα μπορεί να ολοκληρώσει την εργασία της προχωρώντας στον υπολογισμό στατιστικών και απεικονισμένων αποτελεσμάτων. Αν όμως τελικά η ρίζα λάβει λιγότερα σήματα από l^d σήματα, απλά αναφέρει πόσα έχει λάβει και τερματίζει.

6. Ο κάθε splitter/merger ολοκληρώνει την εργασία του αφού συγκεντρώσει τα αποτελέσματά από τους υποκείμενους κόμβους, τα συνθέτει (merge) σε μια νέα ταξινομημένη λίστα (ή λίστες) αποτελεσμάτων την οποία αποστέλλει στο γονιό της. Επιπλέον ο κάθε splitter/merger και οι sorters αναφέρουν στον root το χρόνο που χρειάστηκαν για να ολοκληρώσουν την δουλειά τους.

7. Η ρίζα είναι υπεύθυνη να δημιουργήσει όλα τα αποτελέσματα σε απόλυτους αριθμούς (δηλ. πόσους ψήφους πήρε ο κάθε υποψήφιος και πόσοι είναι οι άκυρες ψήφοι), ποσοστώσεις καθώς επίσης και τα top εκλογικά τμήματα που δέχτηκαν «συνολικά» τουλάχιστον το p % των ψήφων. Το root τυπώνει τον αριθμό του σημάτων ***USR1*** που έχει 'αποδεχτεί' με επιτυχία όπως και τους χρόνους εκτέλεσης όλων των υποκείμενων

[1] unreliable messaging

κόμβων. Τέλος, το root παράγει ένα γράφημα με την βοήθεια του `gnuplot` δίνοντας ένα γράφημα με το ποσοστά επιτυχίας για τους υποψηφίους ταξινομημένα σε φθίνουσα σειρά επιτυχίας και τερματίζει.

8. Αν αντί για *`USR1`* χρησιμοποιήστε *`SIGRTMIN+1`* το αποτέλεσμα του αριθμού των σημάτων που μπορείτε να πάρετε στην ρίζα είναι διαφορετικός; Γιατί; (5 έξτρα πόντοι βαθμολογίας).

Όλες οι διαδικασίες του Σχήματος 1 θα πρέπει να τρέχουν ταυτόχρονα και να προοδεύουν ασύγχρονα. Μια διεργασία γονιός μπορεί να δουλεύει με πολλαπλά named-pipes (η pipes) για να δέχεται τα επιμέρους αποτελέσματα.

Γραμμή Κλήσης της Εφαρμογής:

Η εφαρμογή μπορεί να κληθεί με τον παρακάτω τρόπο:
./mytally -i TextFile -l numOfSMs -d Depth -p Percentile -o OutputFile
όπου:

- *mytally* είναι το εκτελέσιμο της εφαρμογής που δημιουργεί την ιεραρχία διεργασιών και καλεί διαφορετικά προγράμματα,
- *TextFile* είναι το ASCII αρχείο με τα δεδομένα εισόδου (μία ψήφος ανά γραμμή),
- *numOfSMs* είναι ο αριθμός των κόμβων splitters/mergers που θα πρέπει να δημιουργηθούν από κάθε κόμβο,
- *Depth* είναι το βάθος του δένδρου που θα δημιουργηθεί,
- *Percentile* είναι το ποσοστό ψήφων που θα αποδώσουν τα top εκλογικά κέντρα,
- *OutputFile* το αρχείο που ο root να τυπώσει αποτελέσματα αν κάτι τέτοιο ζητηθεί.

Οι σημαίες *-i/-l/-d/-p/-o* μπορούν να χρησιμοποιηθούν με οποιαδήποτε σειρά στην γραμμή εκτέλεσης του προγράμματος.

Τα δεδομένα δίνονται σε ASCII μορφή σε ένα αρχείο. Κάθε γραμμή δεδομένων περιγράφει μια ανώνυμη ψήφο που έχει τα εξής στοιχεία: «`επίθετο υποψηφίου, αριθμός εκλογικού τμήματος, έγκυρο/άκυρο`».

Τι πρέπει να Παραδοθεί:

1. Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματος σας (1-2 σελίδες σε ASCII κειμένου είναι αρκετές).
2. Ένα Makefile (που να μπορεί να χρησιμοποιηθεί για να γίνει αυτόματα το compile του προγράμματος σας).
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 οτιδήποτε άλλο χρειάζεται.

Άλλες Σημαντικές Παρατηρήσεις:

1. Οι εργασίες είναι *ατομικές*.
2. Το πρόγραμμα σας θα πρέπει *να τρέχει σε* `Linux` αλλιώς *δεν θα 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. Σε καμιά περίπτωση τα MS-Windows *δεν είναι επιλέξιμη* πλατφόρμα για την παρουσίαση αυτής της άσκησης.

ΠΑΡΑΡΤΗΜΑ: Χρονισμός στο Unix:

```
#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);
}
```