ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
2η Εργασία - Τμήμα: Περιττών Αριθμών Μητρώου
K22: Λειτουργικά Συστήματα – Χειμερινό Εξάμηνο '24
Ημερομηνία Ανακοίνωσης: Πέμπτη 31/10/24
Ημερομηνία Υποβολής: Τρίτη 26/11/24 Ώρα 23:55

Εισαγωγή στην Εργασία:

Ο στόχος αυτής της εργασίας είναι να εξοικειωθείτε με κλήσεις συστήματος που δημιουργούν νέες διεργασίες, διαχειρίζονται υπάρχουσες και βοηθούν στο συντονισμό και την ολοκλήρωση διεργασιών που όλες μαζί λειτουργούν ταυτόχρονα σε μια ιεραρχία. Το πρόγραμμα σας με όνομα `lexan`, δημιουργεί μία τέτοια ιεραρχία της οποίας οι κόμβοι συνολικά διαβάζουν κομμάτια αρχείου κειμένου τυχαίου μεγέθους, επιτελούν λεξικογραφική ανάλυση, και από κοινού παράγουν τις *top-k* μη–συνήθεις λέξεις που βρίσκουν στο εν λόγω αρχείο κειμένου.

Στην εργασία θα πρέπει:

1. να δημιουργήσετε την αναφερόμενη ιεραρχία διεργασιών με την κλήση `fork()`,
2. κόμβοι σε διαφορετικά επίπεδα της ιεραρχίας να εκτελούν πιθανώς *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. Οι κλήσεις αυτές συμπεριλαμβάνουν τις `exec*()`, `mkfifo()`, `pipe()`, `open()`, `close()`, `read()`, `write()`, `wait()`, `waitpid()`, `poll()`, `select()`, `dup()`, `dup2()`, `kill()` και `exit()`.

Η ιεραρχία διεργασιών που θα δημιουργήσετε, έχει σαν βασικό εκτελέσιμο το `lexan` στη ρίζα της. Το εκτελέσιμο δυναμικά δημιουργεί εσωτερικούς κόμβους ή/και άλλους κόμβους–φύλλα που θα πρέπει να εκτελούν διαφορετικά εκτελέσιμα αρχεία. Η δενδρική δομή ταυτόχρονων προγραμμάτων διαθέτει το αρχικό σας πρόγραμμα (σαν root), εσωτερικούς κόμβους που ονομάζονται splitters και τέλος κόμβους φύλλα που ονομάζονται builders. Κατά την ανάλυση αυτή, οι συχνότητες [1] από «χρήσιμες» ή ενδιαφέρουσες λέξεις υπολογίζονται. Σε αυτές τις «χρήσιμες» λέξεις ΔΕΝ περιλαμβάνονται οι λέξεις του exclusion list που παρέχονται σε ένα αρχείο με την μορφή in-line parameter σε κάθε εκτέλεση του `lexan`. Το τελικό αποτέλεσμα της ιεραρχίας είναι οι k πιο 'δημοφιλείς' λέξεις (δηλ. *top-k*) που εμφανίζονται στο κείμενο.

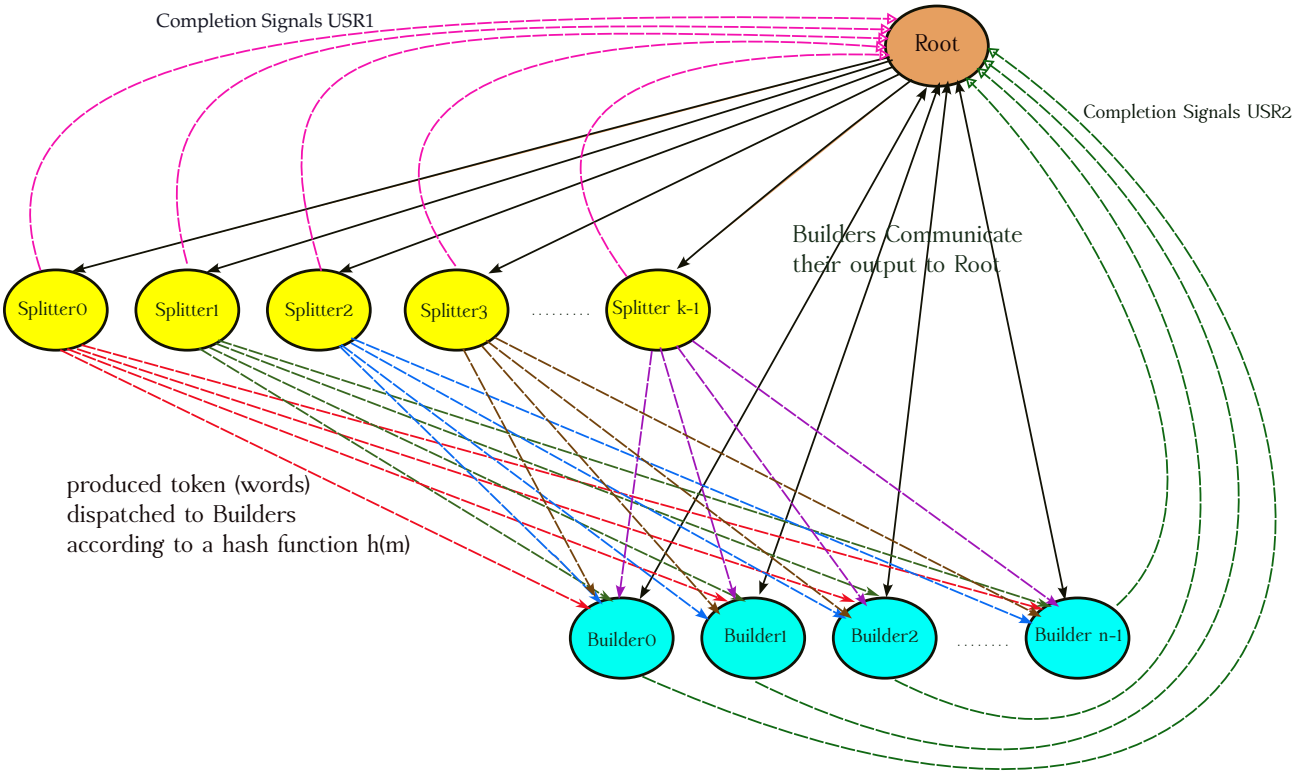
Ο τρόπος με τον οποίο επιλύουμε το πρόβλημα ανάλυσης κειμένου ακολουθεί την λογική του «διαίρει και κατέκτησε»: οι εσωτερικοί κόμβοι (splitters) αναθέτουν εργασία σε υποκείμενους (builders). Οι τελευταίοι οργανώνουν τις συλλεγμένες πληροφορίες ώστε να μπορούν να υπολογιστούν οι συχνότητες λέξεων, τις οποίες και παραδίδουν στον κόμβο συντονισμού που δημιουργεί μεταξύ άλλων, την παρουσίαση αποτελεσμάτων σε ένα γράφημα.

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 pipes είτε named-pipes (FIFOs).

Διαδικαστικά:

- Το πρόγραμμα σας θα πρέπει να γραφτεί σε `C` (ή `C++` αν θέλετε αλλά χωρίς τη χρήση `STL/Templates`) και να τρέχει στα Linux *workstations* 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 `https://www.alexdelis.eu/k22/` για επιπρόσθετες ανακοινώσεις.
- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση κλπ.) είναι ο Δρ. Σαράντης Πασκαλής `paskalis+AT-di`, η κ. Άννα Καββαδά `ankavvada+AT-di`, και ο κ. Νίκος Λαζαρόπουλος `niklaz+AT-di`.

[1] $f = WordOccurenceCounter/TotalNumberOfInterestingWordsInText$



Σχήμα 1: Παράδειγμα ιεραρχίας διεργασιών με k splitters & n builders.

Σύνθεση Ιεραρχίας του `lexan`:

Το Σχήμα 1 δείχνει ένα παράδειγμα της ιεραρχίας που θα πρέπει το πρόγραμμά σας να δημιουργήσει. Ο στόχος της ιεραρχίας είναι να βρίσκει την συχνότητα λέξεων που παρουσιάζουν 'ενδιαφέρον' σε ένα κείμενο και έτσι μπορεί να το ορίζουν πιθανώς μονοσήμαντα ή και να το οριοθετούν/tag. Ο root κόμβος παρέχει στους splitters το εύρος γραμμών μέσα σε ένα αρχείο που οι τελευταίοι θα πρέπει να διαβάσουν και να δημιουργήσουν απλές ακολουθίες από λέξεις (χωρίς σύμβολα, παρενθέσεις, σημεία στίξης, κλπ.). Οι builders απλά δημιουργούν λίστες λέξεων που έχουν βρεθεί με αντίστοιχους μετρητές συχνότητας. Όταν ολοκληρώσουν την εργασία τους, οι builders στέλνουν τα επιμέρους αποτελέσματα τους στον root ο οποίος τα συνθέτει και παράγει τις *top-k* πιο συχνές λέξεις, τυπώνει στατιστικά εκτέλεσης των διεργασιών στην ιεραρχία και τερματίζει.

Κάθε φορά που υπάρχει ανάγκη να δημιουργηθεί ένα παιδί στην ιεραρχία, μία κλήση `fork()` θα πρέπει να εκτελεστεί. Εάν ο χώρος μιας 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 και builder.

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

1. Οι τρεις τύποι κόμβων που αποτελούν την ιεραρχία αντιπροσωπεύουν 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 διαδικασία παρουσίασης των αποτελεσμάτων.

3. Σύμφωνα με τον αριθμό l που έχει καθοριστεί στην γραμμή κλήσης του κυρίου προγράμματος, η ρίζα δημιουργεί l splitters. Ο κάθε splitter παίρνει σαν είσοδο το $1/l$ των γραμμών του αρχείου κειμένου. Η δουλειά της splitter είναι η παραγωγή λέξεων που υπάρχουν στο κομμάτι του κειμένου της, *πλην* εκείνων των λέξεων που δίνονται στο exclusion list. Οι παραγόμενες λέξεις δεν θα πρέπει να έχουν σημεία στίξης, σύμβολα, ή ψηφία (νούμερα).

4. Κάθε παραγόμενη λέξη αφού περάσει από μια συνάρτηση κατακερματισμού $f()$ οδεύει σε ένα από του n builder κόμβους. Ο ρόλος της συνάρτησης $f()$ είναι να 'κατευθύνει' την ίδια λέξη ανεξάρτητα από το που παράγεται (splitter) σε ένα και μόνο συγκεκριμένο builder (από τους n που είναι διαθέσιμοι). Για αυτό το σκοπό ο κάθε splitter θα πρέπει να ανοίξει διόδους επικοινωνίας (pipes ή named pipes) με *όλους* τους builders. Οι λέξεις που δεν προωθούνται από splitters σε builders είναι εκείνες που βρίσκονται στο exclusion list. Η εν λόγω λίστα ορίζεται στην γραμμή κλήσης του προγράμματος.

5. Όταν ένας splitter ολοκληρώσει την εργασία του στέλνει ένα `USR1` σήμα ειδοποιώντας έτσι την ρίζα ότι έχει ολοκληρωθεί η εργασία της και μπορεί να τερματίσει.

6. Ο κόμβος ρίζας επίσης δημιουργεί n κόμβους τύπου builder. Το n είναι παράμετρος που παρέχεται από την γραμμή εντολής του βασικού προγράμματος. Όλες οι λέξεις που έρχονται σε ένα builder αποθηκεύονται σε μια δυναμική δομή (λίστα, πίνακας–κατακερματισμού, δένδρο, κλπ) μαζί με ένα μετρητή που δείχνει πόσες φορές έχει εμφανιστεί μέχρι στιγμής η εν λόγω λέξη στο κείμενο.

7. Ο κάθε builder ολοκληρώνει την εργασία του αφού στείλει όλα τα αποτελέσματά του στην ρίζα (με την βοήθεια μιας pipe ή named-pipe). Επιπλέον ο κάθε builder πρέπει να αναφέρει στον root το χρόνο που χρειάστηκε για να ολοκληρώσει την εκτέλεσή του.

8. Η κόμβος ρίζας αναλαμβάνει να 'συνθέσει' τα αποτελέσματά από όλους τους builders.

9. Κάθε builder που ολοκληρώνει την εργασία του στέλνει ένα σήμα `USR2` στην ρίζα και μπορεί να τερματίσει.

10. Η ρίζα δημιουργεί μια λίστα από τις *top-k* λέξεις που έχουν παραχθεί με σειρά φθίνουσας συχνότητας. Η εν λόγω λίστα μπορεί να αποθηκευτεί και σε ένα αρχείο επιλογής σας (μπορεί να ορίζεται στην γραμμή κλήσης). Η ρίζα τέλος τυπώνει τον αριθμό των `USR1` και `USR2` σημάτων που έχει λάβει όπως επίσης και τους χρόνους εκτέλεσης της root καθώς και των builder διεργασιών της ιεραρχίας και τερματίζει.

Όλες οι διαδικασίες του Σχήματος 1 θα πρέπει ιδεατά να τρέχουν ταυτόχρονα και να προοδεύουν ασύγχρονα. Η επικοινωνία μεταξύ των κόμβων γίνεται είτε με απλές είτε με μόνιμες σωληνώσεις (pipes/named-pipes). Το `lexan` ολοκληρώνει την εκτέλεση του τυπώνοντας στο `tty` τα εξής:

1. τις ταξινομημένες *top-k* λέξεις μαζί με την συχνότητά τους,
2. το χρόνο που χρειάστηκε για να ολοκληρώσει ο κάθε κόμβος–φύλλο την εργασία του,
3. τον αριθμό των σημάτων `USR1` και `USR2` που έχουν παραληφθεί από την ρίζα.

Γραμμή Κλήσης της Εφαρμογής:

Η εφαρμογή μπορεί να κληθεί με τον παρακάτω τρόπο:
./lexan -i TextFile -l numOfSplitter -m numOfBuilders -t TopPopular -e ExclusionList -o OutputFile
όπου:

- lexan είναι το εκτελέσιμο (*lex*-ical *an*-alysis),
- TextFile είναι το ASCII αρχείο εισόδου,
- numOfSplitter ο αριθμός των κόμβων splitter που θα πρέπει να δημιουργηθούν,

- numOfBuilders ο αριθμός των κόμβων builder που θα πρέπει να δημιουργηθούν (καλή ιδέα είναι να πρώτος αριθμός),
- TopPopular είναι ένα νούμερο που εκφράζει πόσες από τις πιο δημοφιλείς λέξεις θα πρέπει να εμφανιστούν,
- ExclusionList το αρχείο με τις λέξεις που ΔΕΝ θα πρέπει να περάσουν για επεξεργασία στους builders,
- OutputFile το αρχείο που ο root θα τυπώσει όλες τις λέξεις που βρέθηκαν στην διάρκεια της ανάλυσης μαζί με τους σχετικούς counters.

Οι σημαίες -i/-l/-m/-t/-e/-o μπορούν να χρησιμοποιηθούν με οποιαδήποτε σειρά στην γραμμή εκτέλεσης του προγράμματος. Τα δεδομένα δίνονται σε ASCII μορφή και δεν υπάρχει καμία πληροφόρια όσον αφορά στο μέγεθος του αρχείου κειμένου υπό ανάλυση.

Χαρακτηριστικά του Προγράμματος που Πρέπει να Γράψετε:

1. Δεν μπορείτε να κάνετε pre-allocate με στατικό τρόπο οποιοδήποτε χώρο αφού δομή(-ές) θα πρέπει να μπορεί(-ουν) να μεγαλώσει(-ουν) χωρίς ουσιαστικά κανέναν περιορισμό όσον αφορά στον αριθμό των λέξεων που μπορούν να χρησιμοποιήσουν. Η χρήση στατικών πινάκων/δομών που δεσμεύονται στη διάρκεια της συμβολομετάφρασης του προγράμματος σας *δεν είναι αποδεκτές επιλογές*.
2. Οποιαδήποτε σχεδιαστική επιλογή που θα υιοθετήσετε, θα πρέπει να τη δικαιολογήσετε στην αναφορά σας.
3. Έχετε πλήρη ελευθερία να επιλέξετε τον τρόπο με τον οποίο τελικά θα υλοποιήσετε τις 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 κείμενο είναι αρκετές).
2. Οπωσδήποτε ένα `Makefile` που να μπορεί να χρησιμοποιηθεί για να γίνει αυτόματα compile τα πρόγραμμα σας.
3. Ένα `zip/7z` αρχείο με όλη σας τη δουλειά σε έναν κατάλογο που πιθανώς να φέρει το όνομα σας και θα περιέχει όλη σας τη δουλειά δηλ. `source files, header files, output files` (αν υπάρχουν) και οτιδήποτε άλλο χρειάζεται.

Βαθμολόγηση Προγραμματιστικής Άσκησης:

Σημεία Αξιολόγησης Άσκησης	*Ποσοστό Βαθμού (0–100)*
Ποιότητα στην Οργάνωση Κώδικα & Modularity	25%
Σωστή Εκτέλεση του `lexan`	15%
Αντιμετώπιση όλων Απαιτήσεων	45%
Χρήση Makefile & Separate Compilation	08%
Επαρκής/Κατανοητός Σχολιασμός Κώδικα	07%

Άλλες Σημαντικές Παρατηρήσεις:

1. Η εργασία είναι **ατομική**.
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 επίσης 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 ή με συστήματα αυτόματης παραγωγής λογισμικού ή με αποθηκευτήρια (repositories) οποιασδήποτε μορφής.
5. Αναμένουμε ότι όποια υποβάλει την εν λόγω άσκηση θα πρέπει να έχει πλήρη γνώση και δυνατότητα εξήγησης του κώδικα. Αδυναμία σε αυτό το σημείο οδηγεί σε μηδενισμό στην άσκηση.
6. Υποβολές ασκήσεων που **δεν χρησιμοποιούν** separate compilation χάνουν αυτόματα 8% του βαθμού.
7. Σε καμιά περίπτωση τα Windows *δεν είναι επιλέξιμη* πλατφόρμα για την υλοποίηση 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);
}
```