ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
2η Εργασία - Τμήμα: Περιττών Αριθμών Μητρώου
K22: Λειτουργικά Συστήματα – Χειμερινό Εξάμηνο '18
Ημερομηνία Ανακοίνωσης: Παρασκευή 26 Οκτωβρίου 2018
Ημερομηνία Υποβολής: Τετάρτη 21 Νοεμβρίου 2018 Ώρα 23:59

Εισαγωγή στην Εργασία:

Στα πλαίσια της εργασίας θα πρέπει να γράψετε ένα πρόγραμμα το οποίο δημιουργεί νέες διεργασίες-παιδιά (children) με τη βοήθεια της εντολής συστήματος `fork()`. Τα παιδιά-διεργασίες συνθέτουν μια ιεραρχία διεργασιών που έχει τη μορφή ενός δυαδικού δέντρου (binary tree). Το δέντρο αυτό έχει εσωτερικούς κόμβους καθώς και κόμβους-φύλλα (leaf nodes).

Ο σκοπός της άσκησης είναι η εκτέλεση ερωτημάτων (queries) σε ένα 'υψηλού-επιπέδου δυαδικό αρχείο εγγραφών' και η ταξινόμηση των αποτελεσμάτων της αναζήτησης. Οι διεργασίες του δυαδικού δέντρου διεργασιών υλοποιούν την αναζήτηση εγγραφών στο αρχείο. Οι κόμβοι-φύλλα αναλαμβάνουν την αναζήτηση εγγραφών σε τμήματα του αρχείου, ενώ οι εσωτερικοί κόμβοι συνθέτουν τα αποτελέσματα που παίρνουν από τα παιδιά τους και τα *προωθούν* στους γονείς τους. Η τελική ταξινόμηση των αποτελεσμάτων γίνεται από το πρόγραμμα συστήματος `sort()`, αφού τελειώσει η διαδικασία της αναζήτησης.

Στα πλαίσια αυτής της άσκησης θα κάνετε τα εξής:

- Δημιουργία ιεραρχίας διεργασιών με την κλήση συστήματος `fork()`.
- Εκτέλεση διαφόρων προγραμμάτων χρήστη/συστήματος από τις διεργασίες της ιεραρχίας.
- Χρήση ενός αριθμού από κλήσεις συστήματος (π.χ. `fork()`, `exec*()`, `getpid()`, `getppid()`, `wait()`, `waitpid()`, `read()`, `write()`, `pipe()`, `mkfifo()` κ.α.).

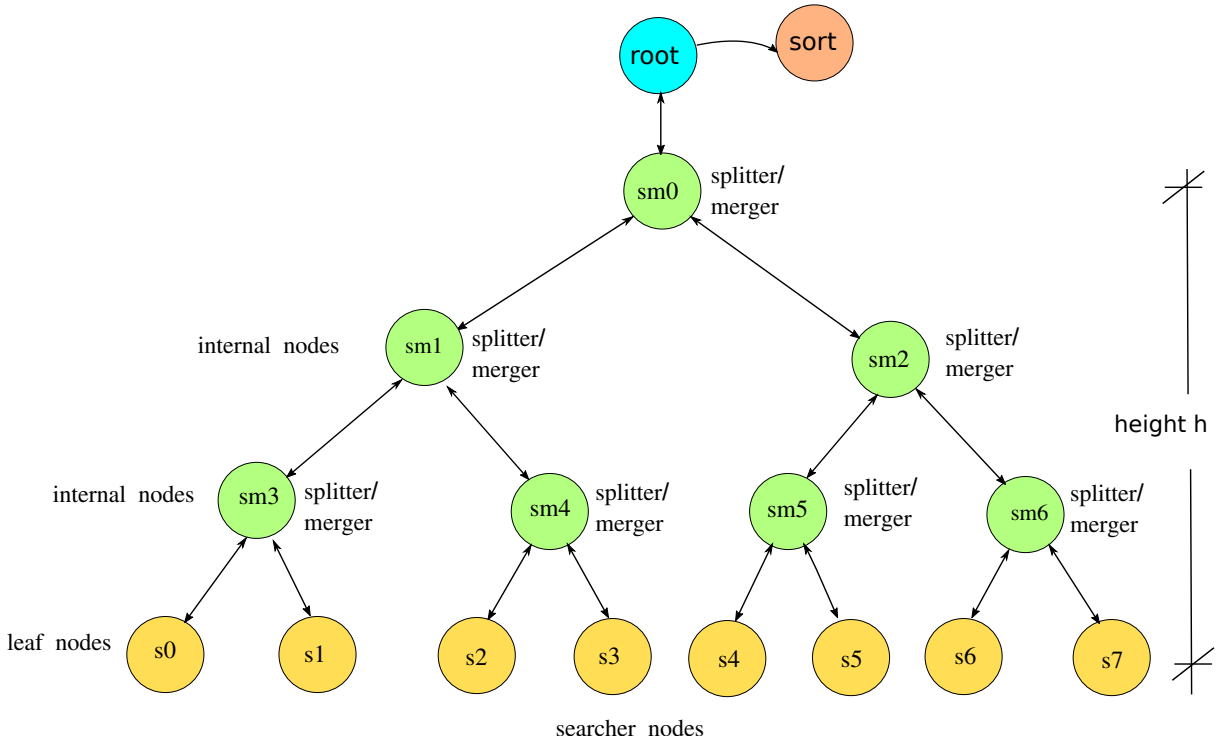
Διαδικαστικά:

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 workstations` του τμήματος.

- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση κλπ.) είναι ο κ. Στράτος Σκλεπάρης `eskleparis+AT-di`, η κ. Αριάννα Τριανταφύλλου `sdi1300179+AT-di`, και ο κ. Οδυσσέας Τρισπιώτης `odytrisp+AT-di`.
- 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/fall2018/k22/home`
- Το πρόγραμμα σας (source code) πρέπει να αποτελείται από ***τουλάχιστον δυο*** (και κατά προτίμηση πιο πολλά) διαφορετικά αρχεία. Το πρόγραμμά σας θα πρέπει *απαραιτήτως να κάνει χρήση* separate compilation .

Η Ιεραρχία των Διεργασιών :

Το Σχήμα 1 δείχνει μια αντιπροσωπευτική ιεραρχία διεργασιών που το πρόγραμμά σας θα πρέπει να δημιουργήσει. Ο στόχος αυτής της ιεραρχίας είναι να βοηθήσει να γίνει αναζήτηση εγγραφών που πληρούν κάποια κριτήρια, από ένα (μεγάλο) αρχείο εγγραφών. Το αρχείο είναι σε binary format. Όλο το δυαδικό δένδρο έχει στόχο ένα κοινό σκοπό για τον οποίο δουλεύουν οι κόμβοι splitters/mergers και searchers με συνεργατικό τρόπο. Έτσι, οι κόμβοι εκτελούν διαφορετικά εκτελέσιμα προγράμματα ανάλογα με το ρόλο που έχουν: τα εκτελέσιμα splitter/merger και οι searcher(s) υλοποιούν την αναζήτηση, το root κάνει την ενορχήστρωση όλων των διεργασιών και το sort ταξινομεί το τελικό αποτέλεσμα. Γενικά, οι διεργασίες παίρνουν παραμέτρους από τους γονείς τους ώστε



Σχήμα 1: Παράδειγμα ιεραρχίας διεργασιών

να κάνουν την συγκεκριμένη δουλειά που τους έχει ανατεθεί και επίσης παράγουν κάποια αποτέλεσμα/-τα και στατιστικά τα οποία τα επιστρέφουν συνήθως στην διεργασία μητέρα τους.

Λειτουργία Κόμβων:

Οι λειτουργίες των διαφόρων κόμβων (διεργασιών) έχουν ως εξής:

1. ***leaf node:*** η διαδικασία λαμβάνει από την μητέρα της το όνομα του αρχείου, το εύρος της αναζήτησης (από ποια εγγραφή μέχρι ποια θα πρέπει ο συγκεκριμένος κόμβος να ψάξει) και τέλος την συνθήκη με την οποία γίνεται η αναζήτηση. Τα αποτέλεσμα θα πρέπει να περάσουν στην διαδικασία γονιός με την βοήθεια ενός απλού pipe ή ενός *named pipe (FIFO)*. Επίσης με τον ίδιο τρόπο μπορούν να περάσουν στο γονιό στατιστικά για τον χρόνο εκτέλεσης του κόμβου.

2. ***splitter/merger node:*** μεταξύ άλλων παραμέτρων, η διαδικασία λαμβάνει από την μητέρα της το όνομα του αρχείου, το εύρος της αναζήτησης, τη συνθήκη με την οποία γίνεται η αναζήτηση όπως επίσης και το 'βάθος' του (εναπομείναντος) δένδρου.

   Εάν το βάθος *δεν είναι* 1, η διαδικασία *splitter/merger* δημιουργεί δύο παιδιά που τρέχουν το ίδιο εκτελέσιμο *splitter/merger*. Ανάλογα με το τρόπο διαχείρισης εύρους αναζήτησης μπορεί το κάθε ένα από αυτά τα παιδιά να λαμβάνει το μισό εύρος αναζήτησης (και όλες τις άλλες απαιτούμενες παραμέτρους συμπεριλαμβανομένου και του *νέου* βάθους μειωμένου κατά 1).

   Αν το βάθος *είναι* 1, η διαδικασία δημιουργεί δύο παιδιά που όμως τώρα τρέχουν το εκτελέσιμο του *leaf node* (πάντα με τα σχετικά ορίσματα).

   Στο τέλος της λειτουργίας του, ένας κόμβος *splitter/merger* παραλαμβάνει τα αποτελέσματα των παιδιών του (με την βοήθεια των δύο υφιστάμενων *pipes* ή *named pipes–FIFOs*), τα συγχωνεύει και το νέο

σύνθετο αποτέλεσμα προωθείται στο γονιό.

Στο γονιό προωθούνται επίσης στατιστικά για το χρόνο εκτέλεσης των παιδιών (πάλι με *pipe/FIFO*) και οποιαδήποτε άλλη πληροφορία κρίνετε απαραίτητη.

3. ***root node:*** η αρχική αυτή διεργασία λειτουργεί σαν 'άγκυρα' του δυαδικού δέντρου και κάνει την συνολική ενορχήστρωση του εγχειρήματος.

   Σε πρώτη φάση δημιουργεί την υψηλότερη διαδικασία *splitter/merge* η οποία με την σειρά της λειτουργεί όπως αναφέραμε παραπάνω. Οι κατάλληλες παράμετροι περνούν από την την *root* στην *splitter/merge* που βρίσκεται στην κορυφή του δυαδικού δένδρου.

   Μόλις η *root* συγκετρώσει όλα τα αποτελέσματα αναζήτησης, τα παραδίδει στην διαδικασία που διεκπεραιώνει το πρόγραμμα `sort()` που ταξινομεί τα αποτελέσματα και τα δίνει στο `tty`.

   Η διεργασία *initial* επίσης τυπώνει και τα απαραίτητα στατιστικά (δηλ. το μέσο χρόνο εκτέλεσης των παιδιών της) και έτσι ολοκληρώνεται η δουλειά της ιεραρχίας.

4. ***sort node:*** η διαδικασία αυτή ταξινομεί όλα τα αποτελέσματα της αναζήτησης με τη βοήθεια του προγράμματος συστήματος `sort()` Για την πλήρη περιγραφή του εν λόγω προγράμματος επικαλεστείτε στην γραμμή εντολής του `tty` σας: `man 1 sort`.

   Το εν λόγω πρόγραμμά λαμβάνει σαν είσοδο ASCII αρχεία και παράγει σχετική έξοδο. Για ευκολία θεωρείστε ότι η ταξινόμηση γίνεται στην πρώτη κολόνα το αρχείου εισόδου.

Νέες διαδικασίες στην ιεραρχία δημιουργούνται με `fork()` και όταν αυτό είναι απαραίτητο το address space των νέων αυτών διεργασιών επικαλύπτεται με `exec*()` που θα πρέπει να δέχεται τα κατάλληλα ορίσματα όπως το υπό αναζήτηση αρχείο το εύρος αναζήτησης, το βάθος του δυαδικού δένδρου που πρέπει να δημιουργηθεί και οτιδήποτε άλλο είναι απαραίτητο. Για την επικοινωνία των διεργασιών με γονείς/παιδιά θα πρέπει να χρησιμοποιήσετε `pipe()`, `mkfifo()`, κλπ.

Τέλος, όταν οι searchers τερματίζουν στέλνουν και ένα σήμα *SIGUSR2* στο root. Ο τελευταίος μαζί με τα στατιστικά που τελικά παράγει επίσης παρέχει και τον αριθμό στο σημάτων *SIGUSR2* που έχει λάβει από τις διαδικασίες- κόμβους στο επίπεδο φύλλων.

Μορφοποίηση Δεδομένων στο Αρχείο Αναζήτησης:

Το δυαδικό αρχείο αναζήτησής αποτελείται από εγγραφές που έχουν την εξής μορφοποίηση:

- Αριθμός Μητρώου τύπου `long`
- Όνομα τύπου `char[20]`
- Επίθετο τύπου `char[20]`
- Οδός Κατοικίας τύπου `char[20]`
- Αριθμός Κατοικίας τύπου `int`
- Όνομα Πόλης τύπου `char[20]`
- Ταχυδρομικός Τομέας τύπου `char[6]`
- Μισθός τύπου `float`

Μορφοποίηση Εξόδου:

Το αποτέλεσμα μιας επερώτησης που δεν είναι κενό, θα πρέπει μαζί με τις εγγραφές που βρίσκει το πρόγραμμά μας (σε μορφή ASCII) να παρέχει και τα εξής στατιστικά:

- Min, Max και μέσος όρος χρόνου εκτέλεσης για τις διαδικασίες searchers.

– Min, Max και μέσος όρος χρόνου εκτέλεσης για τις διαδικασίες splitters/mergers.
– Turnaround Time για την ολοκλήρωση της επερώτησης.

Γραμμή Κλήσης της Εφαρμογής:

Η εφαρμογή μπορεί να κληθεί με τον παρακάτω αυστηρό τρόπο στην γραμμή εντολής (`tty`):
./myfind –h Height –d Datafile -p Pattern -s
όπου

- myfind είναι το εκτελέσιμο που θα δημιουργήσετε,
- Datafile είναι το binary αρχείο εισόδου δεδομένων.
- Height είναι το βάθος του πλήρους δυαδικού δέντρου αναζήτησης που θα δημιουργηθεί, με μέγιστη επιτρεπόμενη τιμή βάθους = 5. Η ελάχιστη τιμή βάθους που επιτρέπεται είναι 1 που σημαίνει ένα κόμβο *splitter-merger* και δύο παιδιά *searcher* που κάνουν αναζήτηση το καθένα στο μισό αρχείο, βάθος= 2 σημαίνει ένα κόμβο *splitter-merger* με δύο παιδιά *splitter-merger* που το καθένα έχει δύο παιδιά *searcher* που μπορεί να κάνουν αναζήτηση στο 1/4 του αρχείου το καθένα κτλ.
- Pattern είναι το (sub-)string το οποίο ψάχνουμε στο (δυαδικό) αρχείο δεδομένων (οποιοδήποτε πεδίο).
- Η εμφάνιση της σημαίας `-s` δηλώνει ότι οι κόμβοι searchers ψάχνουν τμήματα του αρχείου που είναι ανομοιόμορφα το πλήθος (skew) των εγγραφών που ψάχνουν. Αν υποθέσουμε ότι έχουμε 2^h searchers και το αρχείο `Datafile` έχει k εγγραφές, ο πρώτος searcher θα πρέπει να αναλάβει $k * 1/\Sigma_{n=1}^{2^h} n$ πρώτες εγγραφές, ο δεύτερος searcher $k * 2/\Sigma_{n=1}^{2^h} n$ εγγραφές που ακολουθούν, ο i^{th} searcher $k * i/\Sigma_{n=1}^{2^h} n$ εγγραφές που αντιστοιχούν, κλπ.

  Η *μη* εμφάνιση της σημαίας `-s` δηλώνει οτι όλοι οι searchers αναλαμβάνουν τμήματά του `Datafile` που έχουν τον ίδιο αριθμό από εγγραφές (δηλ. $k/2^h$).

Όλες οι παραπάνω σημαίες μπορούν να εμφανίζονται με οποιαδήποτε σειρά.

Η έξοδος του προγράμματος σας θα πρέπει να γίνεται στο `tty` εκτέλεσης του προγράμματος.

Χαρακτηριστικά του Προγράμματος που Πρέπει να Γράψετε:

1. Δεν μπορείτε να κάνετε pre-allocate οποιοδήποτε χώρο αφού η 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 την δομή που θα υιοθετήσετε θα πρέπει να μπορείτε να εξηγήσετε (και να δικαιολογήσετε στην αναφορά σας) την πολυπλοκότητα που οι τεχνικές που υιοθετείτε παρουσιάζουν.
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 οτιδήποτε άλλο χρειάζεται.

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 5% του βαθμού.
6. Σε καμιά περίπτωση τα Windows *δεν είναι επιλέξιμη* πλατφόρμα για την παρουσίαση αυτής της άσκησης.