Εισαγωγή

Ο στόχος αυτής της εργασίας είναι να εξοικειωθείτε με την δημιουργία διεργασιών (processes), με χρήση των κλήσεων συστήματος fork/exec, την επικοινωνία διεργασιών μέσω named-pipes/pipes, τη χρήση low-level I/O, τον χειρισμό signals, και τη δημιουργία bash scripts.

Στην εργασία αυτή θα υλοποιήσετε ένα κατανεμημένο εργαλείο διαχείρισης αρχείων. Θα υλοποιήσετε την εφαρμογή jobExecutor η οποία θα δημιουργεί μια σειρά από Worker διεργασίες που μαζί με την jobExecutor θα απαντάνε σε ερωτήματα του χρηστή.

Εφαρμογή jobExecutor

Η εφαρμογή jobExecutor θα χρησιμοποιείται ως εξής:

```
./jobExecutor -d docfile -w numWorkers
```

- Το docfile (ή κάποιο άλλο όνομα αρχείου) είναι ένα αρχείο που περιλαμβάνει μονοπάτια καταλόγων που περιέχουν ASCII text αρχεία. Κάθε γραμμή θα περιέχει ένα μονοπάτι μόνο.
- Η παράμετρος numWorkers είναι ο αριθμός των Worker διεργασιών που θα δημιουργήσει η εφαρμογή.

Ξεκινώντας, η εφαρμογή jobExecutor θα πρέπει να διαβάσει το αρχείο docfile, να ξεκινάει τις w Workers διεργασίες και να μοιράζει ομοιόμορφα τους καταλόγους που αναφέρονται στο docfile στους Workers. Θα ξεκινάει τους Workers και θα πρέπει να ενημερώνει με κάποιο τρόπο τον κάθε Worker για τους καταλόγους που θα αναλάβει ο Worker. Μπορείτε να υποθέσετε πως οι κατάλογοι θα είναι flat, δηλαδή, δεν θα περιέχουν υποκαταλόγους, αλλά θα έχουν μόνο αρχεία κειμένου.

Όταν η εφαρμογή τελειώσει τη δημιουργία των Worker processes, θα περιμένει είσοδο από τον χρηστή από το πληκτρολόγιο. Ο χρήστης θα μπορεί να δίνει τις ακόλουθες εντολές

```
/search q1 q2 q3 … qN  -d deadline
```

Ο χρήστης αναζητά γραμμές από τα αρχεία που περιέχουν μία η περισσότερες από τις συμβολοσειρές `q1`, `q2`,... `qN`. Η jobExecutor εφαρμογή θα πρέπει να στείλει σε όλους τους Workers τις συμβολοσειρές και να περιμένει απαντήσεις από όλους τους Workers πριν εμφανίσει κάτι στον χρήστη. Κάθε Worker θα ψάχνει όλα τα αρχεία που διαχειρίζεται και θα βρίσκει όλες τις γραμμές που περιέχουν τουλάχιστον μία από τις συμβολοσειρές. Για κάθε αρχείο που βρει, Θα επιστρέφει στην jobExecutor το μονοπάτι του αρχείου, τον αριθμό της κάθε γραμμής που περιέχει κάποια(ες) `qi`, και τα περιεχόμενα της γραμμής. Αν η jobExecutor δεν λάβει απάντηση από όλους τους Workers μέσα σε deadline δευτερόλεπτα, θα εμφανίσει στον χρήστη μόνο όσες απαντήσεις έχει λάβει πριν τη λήξη του deadline, με ένα αντίστοιχο μήνυμα πως μόνο π.χ., 3 από τους 4 Workers απάντησαν.

```
/maxcount keyword
```

Ο χρήστης αναζητά το αρχείο που περιέχει το keyword τις περισσότερες φορές. Η jobExecutor στέλνει το keyword σε όλους τους Workers και τους ζητά να της επιστρέψουν το αρχείο που έχει τις περισσότερες φορές τη συγκεκριμένη λέξη. Αφού μαζέψει τα μέγιστα από όλους τους Workers τυπώνει το ένα συνολικό μέγιστο στο χρήστη, δηλαδή το αρχείο που έχει τις περισσότερες φορές τη συγκεκριμένη λέξη. Σε περίπτωση ισοβαθμίας τυπώνεται το αρχείο που είναι μικρότερο αλφαβητικά σε full-path.

```
/mincount keyword
```

Το ίδιο με πριν αλλά ενδιαφερόμαστε για το αρχείο που περιέχει τον ελάχιστο αριθμό φορών τη συγκεκριμένη λέξη μεταξύ των αρχείων που έχουν τη λέξη τουλάχιστον μία φορά. Το αρχείο δηλαδή που

θα επιστρέφεται ως αποτέλεσμα θα πρέπει να περιέχει τη λέξη ≥ 1 φορές. Αν δεν υπάρχει κανένα αρχείο με αυτή τη λέξη θα τυπώνεται ανάλογο μήνυμα στο χρήστη.

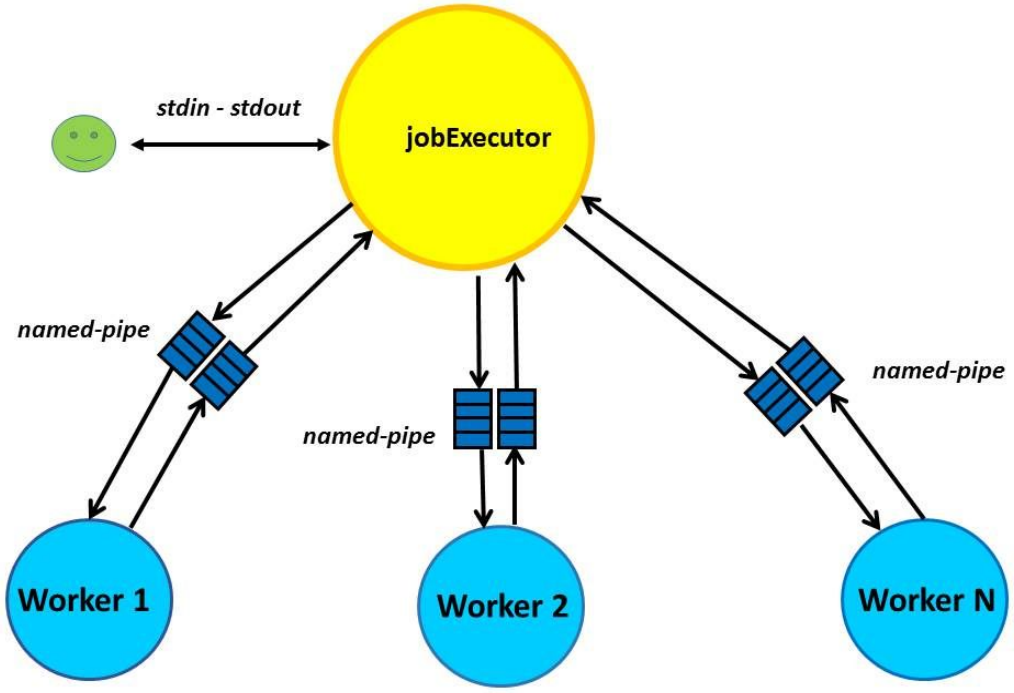
`/wc`
Θα τυπώνεται στο χρήστη ο συνολικός αριθμός χαρακτήρων (bytes), λέξεων και γραμμών από όλα τα αρχεία που έχουν οι Workers. Η jobExecutor θα ρωτάει για τα αντίστοιχα στατιστικά όλους τους Workers, θα τα αθροίζει και θα τα τυπώνει στο χρήστη.

`/exit`
Η jobExecutor θα πρέπει να ενημερώσει τους Workers να τερματίσουν. Πριν τερματίσει, ο κάθε Worker θα πρέπει να έχει καταγράψει στον κατάλογο log/Worker_XXX (όπου XXX είναι το process id του) εγγραφές με τη μορφή

Time of query arrival : Query type : string : pathname1 : pathname2 : pathname3 : … : pathnameN

όπου pathname1...pathnameN είναι τα μονοπάτια αρχείων που βρέθηκε το string στο σύνολο αρχείων που διαχειρίζεται ο Worker. Επίσης, ο Worker θα πρέπει να καθαρίσει οποιεσδήποτε δομές χρειαστεί, να ενημερώσει την jobExecutor για το συνολικό αριθμό strings που βρήκε στα αρχεία του, και να τερματίσει.

Επικοινωνία ανάμεσα στην jobExecutor και Worker



Στο Σχήμα 1 παρουσιάζεται μία αναπαράσταση της δομής της εφαρμογής. Η επικοινωνία ανάμεσα στην jobExecutor και κάθε Worker λαμβάνει χώρα μέσω named pipes. Η προκαθορισμένη συμπεριφορά των named pipes είναι να μπαίνει σε κατάσταση αναμονής η διεργασία που ανοίγει το ένα άκρο μέχρι να ανοιχτεί η σωλήνωση και από το άλλο άκρο. Βέβαια, μπορούμε να αποφύγουμε την παραπάνω συμπεριφορά αν θέσουμε το O_NONBLOCK flag στο δεύτερο όρισμα της κλήσεως συστήματος open(). Για

παράδειγμα, αν θέλουμε να ανοίξουμε ένα named pipe για ανάγνωση χωρίς να τεθούμε σε αναμονή, κάνουμε την κλήση open(pipe_name, O_RDONLY | O_NONBLOCK). Προσοχή, αν σε αυτήν την περίπτωση ανοιχτεί το named pipe για γράψιμο (από το ένα άκρο) και δεν είναι ανοιχτό για διάβασμα (από το άλλο άκρο), τότε θα επιστραφεί σφάλμα. Είστε ελεύθεροι να διαλέξετε όποια μέθοδο λειτουργίας των σωληνώσεων θέλετε.

Η jobExecutor πρέπει με κάποιον τρόπο να ενημερώνει τον Worker ότι σκοπεύει να του μεταβιβάσει κάποια εντολή μέσω της σωλήνωσης έτσι ώστε ο δεύτερος να πάει να τη διαβάσει. Αυτό, για παράδειγμα, θα μπορούσε να επιτευχθεί με την αποστολή κάποιου προσυμφωνημένου σήματος (signal-software interrupt) ή μέσω κάποιου πρωτοκόλλου χρήσης του named_pipe που θα σχεδιάσετε. Είστε ελεύθεροι να διαλέξετε οποία μέθοδο θέλετε.

Αν η jobExecutor μάθει ότι κάποιος Worker έχει τερματίσει (π.χ., λόγω λήψης σήματος SIGKILL), θα πρέπει να δημιουργήσει εκ νέου έναν Worker και να του μεταβιβάσει τα pathnames των καταλόγων που πρέπει να διαχειρίζεται. Κατά την αλλαγή κατάστασης μίας διεργασίας-παιδί, ο πυρήνας του Linux αποστέλλει στη γονική της διεργασία το σήμα SIGCHLD. Ως εκ τούτου, η jobExecutor θα πρέπει να διαχειρίζεται αυτό το σήμα έτσι ώστε να ενημερωθεί για τη κατάσταση αλλαγής σε κάποιο από τους Workers και να δράσει σωστά.

Θα πρέπει να αποφύγετε την ανάγνωση από έναν Worker των περιεχομένων όλων των αρχείων που διαχειρίζεται κάθε φορά που του έρχεται μια εντολή. Ίσως σας φανεί χρήσιμο η υλοποίηση κάποιων δομών δεδομένων από την πρώτη σας εργασία (p.x., trie όπου κάθε postings list θα μπορούσε να καταγράφει το μονοπάτι του αρχείο, τον αριθμό γραμμής που βρίσκεται η λέξη, και τα offsets της γραμμής για γρήγορη εντόπιση της γραμμής μέσα στο αρχείο.) Όποιες σχεδιαστικές αποφάσεις και επιλογές κάνετε κατά την υλοποίηση, θα πρέπει να τις περιγράψετε στο README, στα παραδοτέα.

Bash script(s)

Θα υλοποιήσετε ένα η περισσότερα Bash script που διαβάζουν τα log αρχεία που δημιουργούν οι Workers στον τερματισμό τους και υπολογίζουν τα ακόλουθα στατιστικά:

Total number of keywords searched: ο αριθμός *search* ερωτημάτων που έλαβαν συνολικά οι Workers
Keyword most frequently found: *keyword* [totalNumFilesFound: XXX]: η λέξη (*keyword*) που βρέθηκε τις περισσότερες φορές στα σύνολα αρχείων όλων των Workers. Σε αγκύλες θα αναφέρεται σε πόσα αρχεία βρέθηκε η *keyword*.
Keyword least frequently found: *keyword* [totalNumFilesFound: XXX]: η λέξη (*keyword*) που βρέθηκε τις ελάχιστες φορές στα σύνολα αρχείων όλων των Workers. Σε αγκύλες θα αναφέρεται σε ποσά αρχεία βρέθηκε η *keyword*.

Παραδοτέα

- Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματός σας. 1-2 σελίδες ASCII κειμένου είναι αρκετές. Συμπεριλάβετε την εξήγηση και τις οδηγίες για το compilation και την εκτέλεση του προγράμματός σας σε ένα αρχείο README μαζί με τον κώδικα που θα υποβάλετε.
- Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο ή αλλού θα πρέπει να αναφερθεί στον πηγαίο κώδικά σας αλλά και στο παραπάνω README.

- Όλη η δουλειά σας (πηγαίος κώδικας, Makefile και README) σε ένα tar.gz file με ονομασία OnomaEponymoProject2.tar.gz. Προσοχή να υποβάλετε μόνο κώδικα, Makefile, README και όχι τα binaries.

Διαδικαστικά

- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com. Η πλήρης διεύθυνση είναι https://piazza.com/uoa.gr/spring2018/k24/home. Η παρακολούθηση του φόρουμ στο Piazza είναι υποχρεωτική.
- Το πρόγραμμά σας θα πρέπει να γραφεί σε C (ή C++). Στην περίπτωσή που χρησιμοποιήσετε C++ δεν μπορείτε να χρησιμοποιήσετε τις έτοιμες δομές της Standard Template Library (STL). Σε κάθε περίπτωση το πρόγραμμά σας θα πρέπει να τρέχει στα Linux workstations του Τμήματος.
- Ο κώδικάς σας θα πρέπει να αποτελείται από τουλάχιστον δύο (και κατά προτίμηση περισσότερα) διαφορετικά αρχεία. Η χρήση του separate compilation είναι επιτακτική και ο κώδικάς σας θα πρέπει να έχει ένα Makefile.
- Βεβαιωθείτε πως ακολουθείτε καλές πρακτικές software engineering κατά την υλοποίηση της άσκησης. Η οργάνωση, η αναγνωσιμότητα και η ύπαρξη σχολίων στον κώδικα αποτελούν κομμάτι της βαθμολογίας σας.

Άλλες σημαντικές παρατηρήσεις

- Οι εργασίες είναι ατομικές.
- Όποιος υποβάλει / παρουσιάσει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο μηδενίζεται στο μάθημα.
- Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πώς θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που δεν επιτρέπεται. Οποιοσδήποτε βρεθεί αναμεμιγμένος σε αντιγραφή κώδικα απλά παίρνει μηδέν στο μάθημα. Αυτό ισχύει για όσους εμπλέκονται ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
- Οι ασκήσεις προγραμματισμού μπορούν να δοθούν με καθυστέρηση το πολύ 3 ημερών και με ποινή 5% για κάθε μέρα αργοπορίας. Πέραν των 3 αυτών ημερών, δεν μπορούν να κατατεθούν ασκήσεις.