ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ

Snel: Ένα κέλυφος για το linux

ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΧΡΗΣΤΟΣ ΧΟΥΤΟΥΡΙΔΗΣ ΑΕΜ:8997
cchoutou@ece.auth.gr
+30 697 8894932

1. Περιγραφή

Το Snel είναι ένα κέλυφος για το λειτουργικό σύστημα linux και αναπτύχθηκε ως εργασία για το μάθημα λειτουργικά συστήματα. Το όνομά του προέρχεται από την Γερμανική λέξη schnell που σημαίνει γρήγορα και τις αγγλικές λέξεις snail και shell και αναφέρεται στο γεγονός ότι η ανάπτυξή του έγινε πολύ γρήγορα ναι μεν, αλλά το ίδιο το κέλυφος δεν είναι πιθανόν όσο γρήγορο θα μπορούσε να είναι. Παρόλα αυτά το snel υποστηρίζει batch και interactive mode, σχόλια και pipes.

Δυστυχώς το snel δεν υποστηρίζει ιστορικό, αναγνώριση πλήκτρων (όπως πχ το backspace), μεταβλητές περιβάλλοντος, αλλαγή καταλόγων, εντολές για προγραμματισμό όπως if, switch κτλ. Με λίγα λόγια το snel δεν υποστηρίζει σχεδόν τίποτα το οποίο καθιστά ένα κέλυφος λειτουργικό.

2. Λειτουργία

Η λειτουργία του snel βασίζεται στην ανάγνωση γραμμής προς γραμμή ενός αρχείου ή της stdin, στον κατακερματισμό της σε tokens και εν τέλη την εκτέλεση των εντολών που αναφέρονται σε αυτήν. Για να το επιτύχει αυτό δημιουργεί τις κατάλληλες δομές δεδομένων.

2.1 Child

Η βασική δομή πάνω στην οποία χτίζεται το snel είναι η Child. Ουσιαστικά η child περιγράφει την κάθε εντολή προς εκτέλεση και σε αυτή περιέχονται όλες οι απαραίτητες πληροφορίες που χρειάζεται το snel για την ορθή κλήση του εκτελέσιμου που αφορά. Έτσι ως μέλη(μεταξύ άλλων) έχει:

- **command.** Την εντολή προς εκτέλεση σε μορφή string, στην οποία δεν έχει γίνει καμία περαιτέρω επεξεργασία.
- **arguments.** Ένα διάνυσμα με δείκτες σε αλφαριθμητικά μετά από επεξεργασία του command, ώστε να είναι ακριβώς στην μορφή που μπορούν να περάσουν στην execvp().
- **files**. Ένας πίνακας που περιέχει τους file descriptors αρχείων που η εντολή πρέπει να ανακατευθύνει αν ζητείται.
- **pipes**_. Μια δομή που περιέχει τους file descriptos μιας σωλήνωσης, για την περίπτωση που η έξοδος της εντολής πρέπει να ανακατευθυνθεί στην επόμενη εντολή με τις αντίστοιχες σημαίες για το αν η σωλήνωση γράφει στην επόμενη εντολή ή αν διαβάζει από την αντίστοιχη δομή τις προηγούμενης εντολής

Κατά την δημιουργία της Child στην κλάση περνιέται το αλφαριθμητικό command και δεν γίνεται καμία επεξεργασία σε αυτό. Η επεξεργασία (parsing) γίνεται πριν την εκτέλεση της, με την συνάρτηση **make_arguments()**. Έτσι αν για παράδειγμα λόγο κάποιου λογικού τελεστή(πχ &&) στην εισαγωγή του χρήστη, η εντολή δεν θα χρειαστεί να εκτελεστεί, τότε δεν

θα γίνει και η επεξεργασία του αλφαριθμητικού **command** που περάστηκε στην Child. Όπως γίνεται αντιληπτό αυτή η συμπεριφορά απαιτεί σωστό διαχωρισμό της εισόδου του χρήστη, ώστε το command να αφορά ένα και μόνο εκτελέσιμο καθώς και τυχών λογικούς τελεστές ή σωληνώσεις που έπονται. Αυτό είναι δουλειά της κλάσης Sequencer που περιγράφεται παρακάτω και ποιο συγκεκριμένα της συνάρτησης parse().
Εκτός από την make_arguments() η Child περιέχει και τη συνάρτηση **execute()** που είναι ίσως και η συνάρτηση που κάνει όλη την *"πραγματική"* δουλειά. Σε αυτή τη συνάρτηση βλέπουμε αν πρέπει να γίνει ανακατεύθυνση εισόδου ή εξόδου και ετοιμάζουμε τους file descriptors με την dup() και dup2(). Έπειτα βλέπουμε αν πρέπει να γίνει σωλήνωση. Αν ναι δημιουργούμε τους file descriptors της σωλήνωσης με την pipe().
Έπειτα χρησιμοποιούμε την fork() και για τον γονέα επιστρέφουμε τυχόν ανακατευθύνσεις, κλείνουμε τις σωληνώσεις από την μεριά μας και περιμένουμε το παιδί να τελειώσει την εκτέλεσή του. Για το παιδί κάνουμε ανακατευθύνσεις για τις σωληνώσεις αν απαιτείται και καλούμε την execvp() ώστε να *"τρέξουμε"* το ζητούμενο εκτελέσιμο. Από εδώ και μετά δεν υπάρχει επιστροφή στην συνάρτηση.

2.2 ArgList

Όπως αναφέραμε παραπάνω η Child περιέχει ένα διάνυσμα με τα ορίσματα στην μορφή που μπορούν να περάσουν στην execvp(), το **arguments.** Ο τύπος του arguments είναι ArgList.

Η κλάση ArgList είναι ένας wrapper γύρο από τον container std::vector. Για κάθε εντολή, αυτό το διάνυσμα περιέχει όλα τα ορίσματα που την απαρτίζουν. Πχ για την εντολή `ls -l -a` θα δημιουργηθεί ένα διάνυσμα δεικτών **`char*`** ως εξής **`vector<"ls", "-l", "-a", NULL>`**. Το vector επιλέχθηκε ως βάση γιατί είναι μεταβλητού μεγέθους, που σημαίνει ότι δεν μας ενδιαφέρει πόσα ορίσματα θα περάσει ο χρήστης στο εκτελέσιμο και επίσης μπορεί χωρίς καμία τροποποίηση να περάσει ως δείκτης `char* argv[]`, παίρνοντας απλώς την διεύθυνσή του. Επομένως στην ArgList υλοποιήσαμε μια push_back() που δέχεται από την **make_arguments()** τα ορίσματα σε μορφή string δεσμεύει δυναμικά τα αλφαριθμητικά στο σωρό και δημιουργεί το διάνυσμα που περιγράψαμε παραπάνω.

2.3 Sequencer

Η κύρια κλάση του snel μπορούμε με βεβαιότητα να πούμε πως είναι ο Sequencer. Σε αυτή την κλάση έχουμε μια δομή από ένα ***διάνυσμα διανυσμάτων από Child***. Δηλαδή **`vector<vector<Child>>`**. Στην δομή αυτή αποθηκεύουμε τα Child(ren) με μορφή που να αντικατοπτρίζει τον τρόπο εκτέλεσής τους. Το κάθε αντικείμενο του έξω vector (ας το ονομάσουμε chain) αφορά μια έκφραση την οποία το snel πρέπει να εκτελέσει ως μια οντότητα. Πχ για είσοδο την έκφραση `ls && date` το snel πρέπει να εκτελέσει και τις δύο εντολές προτού επιστρέψει το prompt αλλά η date θα εκτελεστεί μόνο αν η ls είναι επιτυχής. Αυτές τις αλληλουχίες εντολών(expressions) τις ονομάζουμε chains και τις παριστάνουμε με **`vectors<Child>`**

Έτσι για παράδειγμα στην είσοδο:

```
ls –la | more || date ; uname –a && du . –sh
```

το snel θα δημιουργήσει την μορφή:

```
vector <
   vector <Child(ls –la |), Child(more ||), Child(date)>,
   vector <Child(uname –a &&), Child(du . –sh)>
>
```

Η διαδικασία αυτή λαμβάνει χώρα στην συνάρτηση **parse()** του sequencer. Η συνάρτηση αυτή δέχεται ένα string που είναι η είσοδος του χρήστη και δημιουργεί το διάνυσμα που περιγράψαμε παραπάνω.

Εκτός από την συνάρτηση **parse()** ο Sequencer έχει και την συνάρτηση **execute()**. Σε αυτή τη συνάρτηση διατρέχουμε το παραπάνω δισδιάστατο διάνυσμα γραμμή προς γραμμή και καλούμε για το κάθε Child διαδοχικά την **make_arguments()** και την **execute()**. Αν η Child.execute() μας επιστρέψει **stop**, τότε δεν συνεχίζουμε παραπέρα στην τρέχουσα γραμμή και πηγαίνουμε στην επόμενη. Αν στο παραπάνω παράδειγμα φερ' ειπείν η more επιστρέψει επιτυχώς, τότε για το **Child("date")** δεν εκτελούμε τις make_arguments() και execute().

2.4 Διαχείριση μνήμης

Όπως αναφέραμε και παραπάνω στην κλάση Child υπάρχουν μέλη που αφορούν πόρους. Αυτά τα μέλη είναι το arguments που εν τέλη περιέχει δείκτες σε δεσμευμένη μνήμη στο σωρό, οι file descriptors κτλ... Αν προσπαθούσαμε να διαχειριστούμε την δέσμευση και την αποδέσμευση αυτών των πόρων με το χέρι θα είχαμε πρόβλημα. Αυτό γιατί το πρόγραμμα θα γινόταν πολύπλοκο και σίγουρα θα κάναμε κάποιο λάθος. Αντίθετα αποφασίσαμε να χρησιμοποιήσουμε την τεχνική RAII. Έτσι για κάθε πόρο που δεσμεύουμε η αποδέσμευση του γίνεται στον destructor του εν λόγο αντικειμένου. Έπειτα φροντίζουμε τα αντικείμενα που περιέχουν του πόρους να βρίσκονται στην στοίβα. Έτσι όταν θα βγουν εκτός πεδίου εφαρμογής(scope) ο destructor θα καλεστεί αυτόματα, ακόμα και αν υπάρξει κάποιο exception. Πράγματι, όλοι οι πόροι που διαχειριζόμαστε βρίσκονται με κάποιο τρόπο κάτω από τη δομή Child. Το διάνυσμα με τα Child(ren) είναι στον Sequencer και το αντικείμενο του sequencer βρίσκεται σας αυτόματη μεταβλητή στην main(), άρα στη στοίβα. Για τους file descriptors διαχειριζόμαστε το άνοιγμα, κλείσιμό και ανακατεύθυνση τους ώστε το snel να είναι λειτουργικό, αλλά και πάλι για το φόβο του ανοιχτού file descriptor λόγο κάποιου exception οι destructors κλείνουν ότι μπορεί να έχει μείνει ανοιχτό.

3 Επίλογος

Αν και το snel αναπτύχθηκε σε περιορισμένο χρονικό διάστημα (δύο απογεύματα για την ακρίβεια), θεωρούμε πως πληρεί τις προδιαγραφές τις εκφώνησης. Μπορεί λόγο της χρήσης std::string και όχι char* να στερείται των επιδόσεων που θα μπορούσε να επιτύχει αλλά είναι ευανάγνωστο και πανεύκολο στην υλοποίηση.