

ΔΗΜΙΟΥΡΓΙΑ ΕΝΟΣ ΚΕΛΥΦΟΥΣ (SHELL) ΣΤΟ UNIX

Φλώρος-Μαλιβίτης Ορέστης 7796

A large, bold, black serif font spelling out the word "UNIX". The letters are thick and closely spaced. The background is a light gray rectangle with a subtle gradient.

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Εξαμηνιαία Εργασία

Table of Contents

Εισαγωγικά.....	3
Δομή του project.....	3
Εργαλεία που χρησιμοποιήθηκαν / Πηγές.....	3
Χαρακτηριστικά και λειτουργίες.....	4
Συνοπτική παρουσίαση κώδικα.....	5
Πιθανές βελτιώσεις.....	6

Εισαγωγικά.

Σημείωση: Το project θα ανέβει στο GitHub μετά την λήξη της διορίας [εδώ](#)

Δομή του project.

`/bin/`: Φάκελος που περιέχει τα εκτελέσιμα αρχεία του προγράμματος.
`/doc/`: Φάκελος που περιέχει αρχεία σχετικά με την τεκμηρίωση του κώδικα.
`/doc/Doxyfile`: Το configuration που χρησιμοποιήθηκε για να παραχθεί το documentation του κώδικα μέσω του doxygen.
`/doc/report.pdf`: Αυτή η αναφορά.
`/doc/html/`: Το αποτέλεσμα της εκτέλεσης του doxygen σε μορφή html.
`/src/`: Φάκελος που περιέχει αρχεία με τον κώδικα του προγράμματος.
`/src/builtins.c`: Κώδικας σε C σχετικός με της builtin εντολές του shell.
`/src/main.c`: Ο κυρίως κώδικας του shell
`/src/Makefile`: Το Makefile του προγράμματος χρησιμοποιείται για την παραγωγή του εκτελέσιμου κώδικα μέσω της εντολής make.
`/src/utils.h`: Header file.
`/assignment.pdf`: Η εκφώνηση.

Εργαλεία που χρησιμοποιήθηκαν / Πηγές.

- Τεκμηρίωση κώδικα: [Doxygen](#)
- Εισαγωγή εντολών και ιστορικό : [The GNU Readline Library](#)
- Αντίστοιχα Open Source προγράμματα / projects:
 - [GNU Bash](#)
 - [Zsh](#)
 - [bdsh](#)
 - [BD-shell \(a.k.a. bdsh\)](#)
 - Writing a UNIX shell by Jon Madison
- [Implementing a Job Control Shell](#)
- [Writing Your Own Shell By Hiran Ramankutty](#)

Χαρακτηριστικά και λειτουργίες.

Το floShell έχει τα εξής χαρακτηριστικά που έχουν ελεγχθεί και λειτουργούν σωστά:

- Εκτύπωση προτροπής που περιέχει πληροφορίες όπως το όνομα του συνδεδεμένου χρήστη, το όνομα του host και την τρέχουσα διαδρομή (current working directory).
- Εισαγωγή, αποκωδικοποίηση και εκτέλεση εντολών που εισάγει ο χρήστης. Μια εντολή μπορεί να έχει μέχρι 4 ορίσματα (όριο που μπορεί εύκολα να αυξηθεί χωρίς να δημιουργηθούν προβλήματα). Μέσω του shell μπορούν να εκτελεστούν προγράμματα που βρίσκονται στο PATH ή μέσω της εισαγωγής της διαδρομής τους. Για παράδειγμα, μέσω της εντολής './t 1' θα εκτελεστεί το πρόγραμμα 't' που βρίσκεται στην τρέχουσα διαδρομή και θα περαστεί ως όρισμα το '1'. Ο χρήστης έχει την επιλογή της εκτέλεσης της διεργασίας στο προσκήνιο ή στο παρασκήνιο μέσω του χαρακτήρα '&'.

Σε περίπτωση που ο χρήστης στείλει σήμα ctrl-c ή στείλει κενή γραμμή, το πρόγραμμα δεν τερματίζεται. Με την εισαγωγή του eof (ctrl-d) το shell τερματίζεται.

- Διαχείριση σημάτων και τερματισμός διεργασίας μέσω του πατήματος του ctrl+c (κανονικά σήμα SIGINT). Μετά την λήψη του interrupt signal SIGINT γίνεται ερώτηση στον χρήστη για τον τερματισμό της τρέχουσας εργασίας προσκηνίου. Αν υπάρξει θετική απάντηση, στέλνεται το σήμα SIGTERM. Εδώ σημειώνεται ότι αυτή η συμπεριφορά δεν είναι ίδια με άλλα shells. Για παράδειγμα, το bash στέλνει σήμα SIGINT στην ενεργή διεργασία και όχι SIGTERM.

Αν δεν υπάρχει διεργασία στο προσκήνιο, η εκτέλεση του κυρίως προγράμματος δεν τερματίζεται. Οι background διεργασίες δεν επηρεάζονται από τα interrupt signals (σχεδιαστική επιλογή, όχι αδυναμία υλοποίησης). Φυσικά, η εντολή kill (/bin/kill) λειτουργεί κανονικά. Όταν μια διεργασία τερματιστεί μέσω σήματος εκτυπώνεται σχετικό μήνυμα στον χρήστη.

- Διάφορες εσωτερικές εντολές που συναντώνται σε άλλα shells, όπως πχ exit για τερματισμό, cd για αλλαγή τρέχουσας διαδρομής, jobs για εκτύπωση ενεργών background διεργασιών και help. Μια λίστα είναι διαθέσιμη μέσω της εντολής 'help'.
- Διατήρηση ιστορικού των εντολών που έχουν εισαχθεί. Ο χρήστης μπορεί να επιλέξει εντολές που εισήχθησαν στο παρελθόν μέσω των βελών ↑↓. Το αρχείο του ιστορικού βρίσκεται στην διαδρομή ~/.history. Ο χρήστης μπορεί να απενεργοποιήσει το αρχείο μέσω της εντολής hof.
- Μερικό “autocomplete” εντολών που προσφέρεται από την βιβλιοθήκη GNU readline.

Συνοπτική παρουσίαση κώδικα

Το κέλυφος λειτουργεί με έναν ατέρμονα βρόχο που βρίσκεται στην κεντρική συνάρτηση `main()` του προγράμματος. Μετά από την εκτύπωση χαιρετιστήριου μηνύματος το πρόγραμμα δημιουργεί ένα `prompt` μέσω της συνάρτησης `create_prompt_message()`. Τα δεδομένα που εισάγει ο χρήστης διαβάζονται μέσω της συνάρτησης `readline` της βιβλιοθήκης GNU Readline στον `pointer char* line`.

Κενές γραμμές ή γραμμές που διακόπηκαν από `ctrl-c` αγνοούνται. `NULL` στον `line pointer` σημαίνει τερματισμός μέσω `ctrl-d`. Στη συνέχεια, αναζητείται ο χαρακτήρας `'&'` στην γραμμή και με την χρήση της συνάρτησης `strtok()` το `string` διαχωρίζεται με χρήση `delimiters` τον κενό χαρακτήρα `' '` και την νέα γραμμή `'\n'`.

Ο έλεγχος για κλήση `builtin` συνάρτησης γίνεται μέσω της συνάρτησης `check_if_builtin()` που επιστρέφει τον κωδικό της συνάρτησης που βρέθηκε ή `-1`. Η `check_if_builtin()` του αρχείου `builtins.c` συγκρίνει το πρώτο `argument` της γραμμής με το `string cmd` της κάθε `builtin` συνάρτησης. Οι `builtin` συναρτήσεις υλοποιούνται μέσω ενός πίνακα `const builtin_struct builtins[BUILTINS_NUM]` που κάθε στοιχείο του έχει πεδία `code`, `command`, `action` και `help text`.

Αν δεν βρεθεί `builtin` εντολή δημιουργείται νέο αντικείμενο στην συνδεδεμένη λίστα των ενεργών διεργασιών. Η λίστα αυτή υλοποιείται ως εξής:

Σε ένα `struct process` ορίζονται τα πεδία `pid`, `completed`, `status` και `bg` αλλά και ένας δείκτης `struct process* next` που δείχνει στο επόμενο στοιχείο της λίστας. Όποτε δημιουργείται ένα νέο στοιχείο `current`, το πεδίο `next` ορίζεται να δείχνει στην διεύθυνση κορυφής `head` και μετά η κορυφή `head` δείχνει στο `process current`. Ένα αντικείμενο μπορεί να διαγραφεί από την λίστα μέσω της συνάρτησης `pop_from_pid()`.

Τελικά, πραγματοποιείται `fork()` και το παιδί (`pid==0`) καλεί μέσω της `execvp()` την εντολή που όρισε ο χρήστης, με τα ορίσματα που βρέθηκαν από την `strtok()`. Ο γονιός (`pid>0`) στην περίπτωση διεργασίας που εκτελείται στο `background` δεν κάνει τίποτα καθώς ο θάνατος του παιδιού θα διαχειριστεί από τον handler `harvest_dead_child()`. Στην περίπτωση `foreground` διεργασίας ορίζεται ως handler των σημάτων `SIGINT` (`ctrl-c`) η συνάρτηση `killer_interrupt_handle()` που ζητάει επιβεβαίωση από τον χρήστη και στέλνει σήμα `SIGTERM` στην συνάρτηση με `pid: current->pid`. Για τον τερματισμό της διεργασίας ο γονιός περιμένει την λήψη σήματος `SIGCHLD` μέσω της συνάρτησης `sigsuspend()`. Η `sigsuspend()` δεν blockαρει το σήμα και έτσι πρώτα καλείται η συνάρτηση `harvest_dead_child()`, η `current->complete` ορίζεται ως 1 και τελικά η `main()` βγαίνει από τον βρόχο `while(!(current->completed))`.

Πιθανές βελτιώσεις.

Το floShell είναι αρκετά απλοϊκό και υπάρχουν διάφορα βασικά χαρακτηριστικά άλλων κελυφών που θα μπορούσαν να προστεθούν:

- Pipelining και “αλυσίδες” εντολών (πχ `ls -l | grep key | less`), redirection των standard streams (πχ `command > file`). Για την εφαρμογή αυτών των δυνατοτήτων χρειάζεται να αναβαθμιστεί ο parser και ίσως να αντικατασταθεί η `strtok()` με πιο σύνθετη συνάρτηση. Επίσης, θα πρέπει να προστεθεί ένα νέο struct που θα υλοποιεί μια συνδεδεμένη λίστα “jobs”. Ένα νέο job θα δημιουργείται με την κάθε γραμμή που περνάει ο χρήστης στο πρόγραμμα και θα αρχίζει μια συνδεδεμένη λίστα διεργασιών που πρέπει να εκτελεστεί.
- Υποστήριξη regex και globs.
- Υποστήριξη μεταβλητών, δυνατότητες scripting και έλεγχος ροής (if, for, while, ...).
- Αυτόματη διόρθωση τυπογραφικών λαθών και συμβουλές.