sdi1800152 – Ευάγγελος Παπακώστας – ΣυσΠρο_Εργασία_1

ReadMe

Η εργασία μου έχει αναπτυχθεί σε Windows VSCode σε γλώσσα C. Έχω οργανώσει τον κωδικά μου με αρκετές συναρτήσεις.

- Main(): αρχίζει ορίζοντας τις βασικές δομές μου που πρέπει να χειρίζομαι καθ' όλη τη διάρκεια του προγράμματος. Γενικά έχω 3 structs:
  (1) command→ κρατάει το όνομα μιας εντολής & τον αριθμό από args της
  (2) aliases → κρατάει το όνομα ενός alias, πίνακα με την εντολή+args της που αντικαθιστά πλέον + αυτό τον # από args
  (3) reverse_stack → είναι ένας πίνακας από strings που στο καθένα μπαίνει κάθε νέα εντολή που πληκτρολογεί ο χρήστης & κρατάει την τρέχουσα θέση στον πίνακα αυτό (βοηθάει στο κυκλικό γέμισμά του)
  Ξεκινάει λοιπόν το γενικό while-loop και καλείται η συνάρτηση parse_input() για την είσοδο του χρήστη. Αφού γεμίσουν οι βοηθητικοί μου πίνακες (λεπτομέρειες παρακάτω), ελέγχεται αν η τρέχουσα εντολή είναι δημιουργία! Alias (όχι η ίδια να είναι alias) και στην περίπτωση που είναι προχωράμε στην επόμενη είσοδο αλλιώς καλείται η execute_command() που θα τρέξει την εντολή βάσει όλων των στοιχείων που ετοίμασε η parse_input().

- parse_input(): παίρνει με fgets() το input του χρήστη κ το αποθηκεύει στο cmd ως string. Το προσθέτει στο ιστορικό κατευθείαν και μετά το κάνει tokenize στο cmd_args[] πίνακα.
  - Κάνει ελέγχους για κάποιες βασικές εντολές του shell.
  - Στη συνέχεια υπάρχει η διαχείριση των Redirections.
  - Έπειτα για κάθε token του input:
    (Λειτουργία αυτού του for-loop→ με τη χρήση του flag, πάντα ξεκινάμε περιμένοντας Εντολή. → μετά περιμένουμε Arguments μέχρι να συναντήσουμε κάτι που υποδηλώνει ότι

πρέπει να υπάρχει Εντολή ακριβώς μετά→ παίζουμε με το if(εντολή)-else(args) δηλαδή)
    - στο if μέρος συμπληρώνεται η δομή command,
    - γίνεται έλεγχος για δημιουργία-καταστροφή alias (εδώ θα έπρεπε να είναι και έλεγχος για τα υπόλοιπα στοιχεία της εργασίας που δεν ανέπτυξα, πχ. fg, bg).
    - στο else μέρος κοιτάζουμε αν υπάρχει pipe ή αν είναι απλό argument.
  - Αφού έχουν συμπληρωθεί οι βασικοί μας πίνακες (pipes, δομή command) είμαστε έτοιμη για εκτέλεση της εντολής

- execute_command(): ανάλογα με τα δεδομένα που έχουμε, φτιάχνει τα pipes αν υπάρχουν, κάνει το fork() για την εκτέλεση του παιδιού και αν ΔΕΝ ΕΙΝΑΙ ΔΗΜΙΟΥΡΓΙΑ ALIAS, τότε ελέγχει αν είναι το ίδιο το command ένα alias (αν ναι καλεί την *run_alias()*) και αν δεν είναι φτιάχνει τα string που θα περαστούν στην exec και τρέχει την εντολή. Τέλος ο πατέρας διορθώνει τα stdin/stdout και περιμένει τα παιδιά.

Από την *parse_input()* καλούνται 2 μόνο συναρτήσεις:

- *add_to_history()* → προσθέτει στη δομή μας την καινούρια εντολή του χρήστη και επίσης κοιτάζει αν είναι η εντολή *myHistory* (*num*) ώστε να κληθεί η *print_history()* ή να εκτελεστεί η εντολή num αντίστοιχα.
  Εκτελεστεί εννοώ ότι κάνει το string cmd να είναι η επιθυμητή εντολή και συνεχίζει κανονικά η *parse_input()* χωρίς να ξέρει ότι άλλαξε κάτι.
- *remove_alias()*→ επειδή δεν χρειαζόμαστε κάτι έξτρα για να διαγράψουμε ένα alias, αυτή η διαδικασία μπορεί να γίνει μέσα στην *parse_input()*, ελέγχοντας απλά το όνομα του alias χωρίς να επηρεαστεί κάτι άλλο.
  Η *remove* δηλαδή ψάχνει να βρει το όνομα που δόθηκε και αν υπάρχει το διαγράφει ανανεώνοντας τη δομή aliases.

Μετά την *parse_input()* στη main() καλείται η *add_alias() σε περίπτωση που έχει σηκώσει σημαία (its_a_new_alias) η parse_input()* ότι πρόκειται για δημιουργία καινούριου alias.

- *add_alias()* → ελέγχει αν υπάρχει alias με το ίδιο όνομα και αν όχι τότε προσθέτει το νέο στη δομή μας μαζί με τα args του.

Από την *execute_commands()* καλείται μόνο η *run_alias()*:

- *run_alias():* κάνει ακριβώς ότι κάνει και η *execute_commands()* με πολύ λίγες διαφορές δηλαδή τρέχει εντολή πού υπάρχει ήδη σε δομή.
- *print_all_aliases():* κάνει απλά print όλα τα aliases που υπάρχουν για debugging σκοπούς.

Γενικά:

- *print_history()* → καλείται από την add_to_history() είτε για να κάνει print το ιστορικό όπως ζητείται είτε για debugging σε άλλα σημεία.
- *init_rs()* → αρχικοποιεί τη δομή reverse_stack (το όνομά της δεν είναι 100% πετυχημένο λόγω συνεχής αλλαγής της λειτουργίας της).
- *isFull()* → επιστρέφει αν ο πίνακας που κρατάμε το ιστορικό εντολών είναι γεμάτος (βάσει αυτού ενεργοποιείται το κυκλικό γέμισμα).
- *isEmpty()* → επιστρέφει αν ο πίνακας που κρατάμε το ιστορικό εντολών είναι άδειος.

Αυτά ήταν γενικά περί δομής του προγράμματος.
Όσον αφορά τη λειτουργικότητά του, αναγνωρίζω:

- **Βασικές εντολές** του συστήματος (ls, cat, diff, …) και (exit, cd - ξεχωριστός χειρισμός)
  πχ. cat file1
  diff file1 file2
  ls
  ls -l
  ls -l -r
  ls -l -r -a
  ls -lra
  echo hi (χωρίς “ ”)
  head, tail -num file1
  wc -l/-w/-c
- **Pipes** – λειτουργούν ΧΩΡΙΣ ΑΝΑΚΑΤΕΥΘΥΝΣΗ
  Σημείωση: στην grep ΔΕΝ πρέπει να περνάω “…” αλλά σκέτη τη λέξη για να δουλέψει
  πχ. cat file1 | grep pid

ls -l | wc -l
ls -l | grep ... | wc -l
diff file1 file2 | wc -l
diff file1 file2 | grep ...
diff file1 file2 | grep ... | wc -l

- **Aliases** – λειτουργούν κανονικά (πχ μπορεί να έχουν και pipes μέσα). Στο τέλος ΔΕΝ πρέπει να μπαίνει “;” για να λειτουργήσουν. Στην εκφώνηση το είδα ΜΕ “;”,

```
:> createalias myhome “cd /home/users/smith”;
```

στα test cases ΧΩΡΙΣ.

> ***Διαχείριση aliases***
>
> ```
> nikosp@linux01:lll
> returns
> nikosp@linux01:lll: Command not found.
> then
> nikosp@linux01:createalias lll "ls -las"
> ```

Επιλογή ήταν να το υλοποιήσω χωρίς.
πχ. createalias ll "ls -l"
createalias lll "ls -l -r -a"
createalias lpl "cat file"
createalias qq "ls -l | wc -l"
createalias qqq "cat suspro1.c | wc -l"
createalias kkk "ls -l | grep ... | wc -l"
destroyalias ll
destroyalias lll etc.

- **History** – λειτουργεί κανονικά με κάποια θεματάκια αν ζητηθεί command σε θέση ακριβώς πριν γεμίσει ο πίνακας και κάποιες φορές η 1η εντολή. Πριν και μετά το γέμισμα (20 θέσεις) λειτουργεί κανονικά.
πχ. η εντολή myHistory 5, όπου η 5 εντολή είναι ll, όπου ll είναι ένα alias δουλεύει κανονικά

- **Redirections** – έχει γίνει προσπάθεια υλοποίησής τους, μέχρι το σημείο αλλαγής stdin/stdout είναι σωστό, όμως υπάρχει θέμα στην επαναφορά τους. Οπότε αν τρέξει πχ κάποιο redirection θα γίνει η εκτύπωση στο πχ αρχείο “> out.txt”, όμως θα συνεχίσει γενικά εκεί η εκτύπωση του προγράμματος.

Γενικά υπάρχουν βοηθητικά σχόλια σε όλο τον κώδικα.
Σε κάποιοες εντολές γίνονται κάποια ανεπιθύμητα prints τα οποία δεν πρόλαβα να διαγράψω.
Γενικά είναι από τους κακούς κώδικες που έχω γράψει στο di απλά δεν υπήρχε πολύς χρόνος λόγω παράλληλων εργασιών.

Παρακάτω υπάρχει το απαιτούμενο συμπληρωμένο χαρτί βαθμολόγησης.

Τέλος, η εργασία έχει μεταγλωττιστεί και τρέξει στα μηχανήματα Linux της σχολής.

Συμπληρωμένο ονλάιν χαρτί βαθμολόγησης:

	γενικα σχόλια	ΝΑΙ	ΜΕΡΙΚΩΣ	ΟΧΙ
shell	Βασική λειτουργία κελύφους εισαγωγής και κλήσης απλών εντολών πχ ls	X		
	επιστροφή και εκτύπωση αποτελεσμάτων	X		
redirections	μονή ανακατευθυνση > ή < (5)		X	
	διπλή ανακατεύθυνση		X	
	ανακατεύθυνση προσθήκης >>		X	
pipes	απλό pipe cat file |grep "nikos" (5)	X		
	Συνδυασμός με ανακατεύθυνση cat file |grep "nikos" >file2		X	
				X
background execution	εκτέλεση εντολών στο background			X
	Εκτελεση πολλων εντολών σε μία γραμμη sort file1 &; ls &;			X
	Επιστροφή ολοκλήρωσης πισω στο shell			X
wild chars	υποστηριξη * (για τρέχοντα κατάλογο μονο)			X
	υποστήριξη ? (για τρέχοντα κατάλογο μονο)			X
aliases	create και χρήση alias θα πρέπει να το φτιαξουν εξ αρχής την δομή και πριν την εκτέλεση κάθε εντολής θα πρέπει να ελέγχουν τη δομή αυτή)	X		
	destroy (διαγραφή)	X		
history	myHistroy save και print	X		
	κληση εντολής χωρίς επαναπληκτρολόγηση (!numCommand , myhistory 5)	X		
signals	control-C α) σκοτώνει την εσωτερική διεργασία			X
	control-C β) δεν επηρεάζει το κέλυφος			X
	control-Z α) σκοτώνει την εσωτερική διεργασία			X
	control-Z β) δεν επηρεάζει το κέλυφος			X