Προγραμματισμός Συστήματος / Εργασία 1

Ιωάννης Πετράκης / sdi1800157

Εντολή μεταγλώττισης: make

Εκτελέσιμο: mysh

Περιεχόμενα

6 αρχεία κώδικα, 1 makefile, 1 header file

mainloop.c : Περιέχει 2 συναρτήσεις, την shellLoop(), που είναι η βασική συνάρτηση/λούπα του shell μου και την main που απλά καλεί την shellLoop.

redirection.c : Περιέχει τις συναρτήσεις που υλοποιούν τις ανακατευθύνσεις.

pipes.c : Περιέχει τις συναρτήσεις που υλοποιούν τις σωληνώσεις (pipes).

wild_characters : Περιέχει τις συναρτήσεις που υλοποιούν την υποστήριξη wild characters.

aliases.c : Περιέχει τις συναρτήσεις που υλοποιούν την υποστήριξη aliases.

functions.c : Περιέχει τις υπόλοιπες συναρτήσεις του προγράμματος, που αφορούν το ιστορικό, το input, parsing, κτλ , θα εξηγηθούν παρακάτω.

Πώς λειτουργεί το shell

Το κέλυφος λειτουργεί έχοντας μία λούπα (στην συνάρτηση shellLoop) όπου δέχεται στην αρχή της μία εντολή από τον χρήστη και την εκτελεί. Πιο συγκεκριμένα:

- Δέχεται την εντολή μέσω της συνάρτησης input, αποθήκευση στη μεταβλητή

  char str[MAXCHARS].

- Τσεκάρει αν το input που δόθηκε είναι η εντολή history. Αν είναι, εκτυπώνει το ιστορικό και περιμένει τον χρήστη να δώσει αριθμό εντολής προς εκτέλεση. Στη συνέχεια στο str που είχαμε “history” , βάζουμε την εντολή από το ιστορικό.

- Η εντολή (str) μπαίνει στο ιστορικό στην πρώτη θέση του πίνακα history και μετακινούνται όλες οι εντολές του ιστορικού μία θέση πιο μετά.
- Τσεκάρει αν η εντολή περιέχει wild characters (είτε *, είτε ?). Αν περιέχει, αντικαθιστά τη λέξη που περιέχει * ή & με την κατάλληλη (πχ αν δοθεί η ls ne&.txt θα αντικατασταθεί με το ls new.txt).
- Αν η εντολή είναι η createalias ή η destroyalias προσθέτει ή αφαιρεί στους πίνακες που αφορούν τα aliases. Αν δεν είναι createalias ή destroyalias τσεκάρει αν οποιαδήποτε λέξη της εντολής είναι κάποιο alias και το αντικαθιστά με τη σωστή εντολή. Πχ αν υπάρχει alias “lll” που αντιστοιχεί στο “ls -l” και σε αυτό το βήμα έχουμε την εντολή “ lll | ls | lll “, θα αντικατασταθεί με την “ls -l | ls | ls-l”.
- Σε αυτήν τη φάση η εντολή έχει “φιλτραριστεί” και είναι έτοιμη προς εκτέλεση. Έχουν αντικατασταθεί τυχών wild characters, aliases ή έχει εισαχθεί από το ιστορικό και μετά έχει φιλτραριστεί.
- Για να εκτελέσω τις εντολές, τις διακρίνω σε τρεις περιπτώσεις.
    1. Εντολές που έχουν pipes (μπορούν να περιέχουν και redirection).
    2. Εντολές που έχουν μόνο redirection.
    3. Εντολές που δεν έχουν τίποτα από τα παραπάνω
- Έχω μία if, else if, else που τσεκάρει για τις περιπτώσεις. Αν είναι στην πρώτη, καλεί την handle_pipes, αν είναι στην δεύτερη καλεί την handle_redirection και αν είναι στην τρίτη καλεί την simple_case_forking. Και οι τρεις συναρτήσεις δέχονται σαν όρισμα ένα char* , τους περνιέται η φιλτραρισμένη εντολή.

Υλοποιήσεις/ Συναρτήσεις

Πώς δέχεται το input:

int input(char*) : Με τη συνάρτηση int input(char *) η οποία δέχεται σαν όρισμα τη μεταβλητή στην οποία θα βάλει αυτό που θα δεχτεί από τον χρήστη. Η συνάρτηση φτιάχνει έναν πίνακα χαρακτήρων δυναμικά με τη χρήση της malloc (buffer) και μέγεθος MAXCHARS δηλαδή 2048. Στη συνέχεια δέχεται τον πρώτο χαρακτήρα με χρήση της getchar() και αν αυτός είναι enter ή EOF επιστρέφει -1. Στην main όταν καλώ την input τσεκάρω αν επιστρέφει -1 και αν ναι, κάνω continue έτσι ώστε να αγνοεί την εντολή.

Σε περίπτωση που ο πρώτος χαρακτήρας δεν είναι το enter, μπαίνω σε μία λούπα όπου στο str, δηλαδή το char * που θα μπει το τελικό αποτέλεσμα αντιγράφω τον πίνακα buffer και διαβάζω τον επόμενο χαρακτήρα. Αν ο επόμενος χαρακτήρας είναι enter ή EOF ή ; , η συνάρτηση επιστρέφει γυρνώντας τον αριθμό χαρακτήρων που διάβασε.

***Επειδή υλοποίησα το redirection έτσι ώστε να μην μπορεί να εκτελέσει εντολές που δεν έχουν κενό ενδιάμεσά τους (πχ η “ls > new.txt τρέχει κανονικά αλλά η “ls>new.txt” όχι) κάθε φορά που δίνεται “>” ή “<” ή “>>” , η input βάζει κενό πριν και μετά τους χαρακτήρες αυτούς.

char ** split(char *line) : Η συνάρτηση αυτή δέχεται ένα char *, το χωρίζει με την strtok βάση του κενού και επιστρέφει έναν πίνακα char ** που περιέχει σε κάθε θέση του μία λέξη από το χώρισμα και στην τελευταία NULL.

Redirection:

int redirection(char *) : Επιστρέφει 1 αν υπάρχει redirection στο char * που της δόθηκε και 0 αν δεν υπάρχει, απλά τσεκάρει χαρακτήρα χαρακτήρα μέχρι να βρει < ή >.

Η συνάρτηση χρησιμοποιείται στη βασική λούπα στη φάση που η εντολή είναι έτοιμη προς εκτέλεση. Αν βρεθεί redirection, θα καλεστεί η handle_redirection.

int handle_redirection(char *) : Υπεύθυνη για να χωρίσει την εντολή, να βάλει το κομμάτι που πρέπει να εκτελεστεί σε έναν πίνακα char ** , να κάνει I/O redirection χρησιμοποιώντας dup και dup2 και τέλος να εκτελέσει κάνοντας forking και execvp.

Πιο συγκεκριμένα, δέχεται σαν όρισμα την εντολή προς εκτέλεση σε μορφή char *, την χωρίζει χρησιμοποιώντας τη συνάρτηση split βάζοντας το αποτέλεσμα στον πίνακα char **line και αποθηκεύει με την dup το input και output.

Η split χωρίζει με βάση το κενό, οπότε στον πίνακα line θα βρίσκονται στις αρχικές του θέσεις αυτό που θα πρέπει να γίνει execvp παρακάτω, κάποιο > ή < ή >> και στη συνέχεια αυτό που θα πρέπει να γίνει το επόμενο input ή output (ή και τα δύο σε περίπτωση που έχουμε κάτι της μορφής a < b > c ή a < b >> c).

Επομένως έχω μία λούπα που τσεκάρει το περιεχόμενο του line, το βάζει στον πίνακα newargv μέχει να βρει > ή < . Όταν βρει, βάζει στην τελευταία θέση του newargv NULL και η επόμενη θέση του line θα περιέχει το νέο input ή output. Χρησιμοποιεί την open για να ανοίξει το αρχείο με τα κατάλληλα ορίσματα (O_RDONLY για input , O_WRONLY | O_CREAT | O_TRUNC για output κτλ) και στη συνέχεια χρησιμοποιεί την dup2 για να κάνει την ανακατεύθυνση. Τέλος κάνει forking και execvp με τον πίνακα newargv και dup2 με το αρχικά αποθηκευμένο input/output έτσι ώστε να επανέλθει το αρχικό. Το διπλό redirection τσεκάρεται στην περίπτωση που έχω < .

Pipes:

int pipes(char *) : Αντίστοιχη με την redirection αλλά επιστρέφει 1 αν βρεθεί | και 0 αν δεν βρεθεί.

Αν βρεθούν pipes, καλείται η handle_pipes.

int handle_pipes(char *) : Χωρίζει την εντολή με βάση τον χαρακτήρα | , με χρήση της strtok και βάζει στον πίνακα char *pipes[1024] το αποτέλεσμα. Ο πίνακας αυτός θα περιέχει όσες εντολές πρέπει να εκτελεστούν, αλλά όχι χωρισμένες. Επομένως για κάθε εντολή (for loop) τη χωρίζω με βάση το κενό και τη βάζω στον πίνακα args, τρέχω την εντολή pipe για να φτιαχτεί το pipe και διακρίνω τρεις περιπτώσεις, το να είναι μία εντολή η πρώτη στο pipe, η τελευταία ή κάποια ενδιάμεση εντολή. Όταν

είναι πρώτη, τσεκάρω αν υπάρχει και < , οπότε αλλάζω το stdin και τρέχω την exec_pipes με ορίσματα αυτό που είναι να γίνει execvp το stdin (ή το αλλαγμένο λόγω redirection) και το fd[1] για να γράψει στο pipe. Αν είναι ενδιάμεση καλεί την exec_pipes με ορίσματα αυτό που είναι να γίνει execvp, fd[0] για να διαβάσει input από το pipe και fd[1] για να γράψει στο pipe για την επόμενη. Αν είναι τελευταία, καλεί την exec_pipes με ορίσματα αυτό που είναι να γίνει execvp, fd[0] για να διαβάσει input από το pipe αυτό που έγραψε η προηγούμενη και stdout ή κάποιο redirected output αν υπάρχει > ή >> .

int exec_pipes(char **, int , int) : Κάνει fork και execvp. Στο παιδί που δημιουργείται, πριν την execvp αλλάζει το input και output με αυτά που έχουν δοθεί ως ορίσματα.

Background:

Κάθε φορά πριν από οποιαδήποτε fork καλείται η **check_background(char **)** , η οποία δέχεται τον πίνακα που είναι να εκτελεστεί με την execvp παρακάτω. Αν είναι να εκτελεστεί στο background, θα περιέχει στην τελευταία θέση το “&”. Η συνάρτηση αυτή αν βρει στο τέλος αυτόν τον χαρακτήρα, τον αντικαθιστά με NULL και επιστρέφει 1, αλλιώς επιστρέφει 0. Το αποτέλεσμα σώζεται σε μία μεταβλητή και μετά το forking ο πατέρας ελέγχει την τιμή της μεταβλητής και αν είναι 0 προχωράει κανονικά σε waitpid , αλλιώς όχι , έτσι ώστε το κέλυφος να συνεχίσει και να μην περιμένει να τελειώσει το παιδί.

Wild characters:

Η int **wild_characters(char *)** ελέγχει αν υπάρχουν * ή ? σε ένα string που της δίνεται και επιστρέφει 0 ή 1 ανάλογα.

Η **char * handle_wildcharacters(char *)**, δέχεται ένα string, το χωρίζει με την split και για κάθε λέξη ελέγχει αν περιέχει ? ή * με την wild_characters και αν ναι, χρησιμοποιεί την glob για να βρει με τι θα πρέπει να αντικαταστήσει τη λέξη που περιέχει * ή ?. Κάθε λέξη είτε αντικαθίσταται με το αποτέλεσμα της glob είτε όχι, γίνεται strcat για να προστεθεί στο new_str που είναι αυτό που επιστρέφει η handle_wildcharacters.

Aliases:

Η υλοποίηση των aliases γίνεται με δύο δυναμικά ορισμένους πίνακες, τον alias και τον real που υπάρχει αντιστοιχία μεταξύ τους. Η συναρτήσεις createalias και destroyalias δέχονται ένα string και ελέγχουν αν περιέχει την εντολή createalias ή destroyalias αντίστοιχα. Ο έλεγχος γίνεται στη φάση της λούπας που “φιλτράρεται” ακόμα η εντολή. Αν είναι createalias, καλώ την handle_createalias η οποία βάζει στους πίνακες alias και real, την πρώτη λέξη μετα το “createalias” και τις υπόλοιπες στους πίνακες alias και real αντίστοιχα. Αυτό το κάνει καλώντας την add_one που κάνει realloc έτσι ώστε να αλλάξει το μέγεθος των πινάκων.

Αν έχω destroyalias καλείται η handle_destroyalias, η οποία καλεί την remove_one αφαιρώντας έτσι το alias που δόθηκε και την αντιστοιχία του από τους πίνακες alias και real αντίστοιχα. Η remove_one χρησιμοποιεί κι αυτή realloc.

Η search_alias ψάχνει στον πίνακα alias ένα string και αν βρεθεί επιστρέφει 1 αλλιώς 0.

Signals:

Χρησιμοποιώ την sigaction για να κάνω ignore τα σήματα SIGSTP και SIGINT. Ignore κάνω στην αρχή της βασικής μου λούπας και για να μπορούν να σταματάνε ή να διακόπτονται τα προγράμματα που τρέχουν από το κέλυφος, σε κάθε forking, όταν το αποτέλεσμα της fork είναι 0, δηλαδή στα παιδιά, πάλι με sigaction κάνω reestablish το default των δύο αυτών σημάτων.

History:

Έχω έναν πίνακα history 20 θέσεων, char history[20][MAXCHARS]. ‘Οταν ο χρήστης δώσει τη λέξη history στο input , χρησιμοποιώ την printHistory για να εκτυπώσω το history. Αν στο history δεν έχω τίποτα ακόμα, η printHistory επιστρέφει -1, εκτυπώνεται στην οθόνη "No history yet" και γίνεται continue για να συνεχιστεί η λούπα. Αλλιώς, εκτυπώνεται το ιστορικό και το πρόγραμμα περιμένει έναν αριθμό από τον χρήστη και εκτελεί τον αριθμό εντολής που δόθηκε (δεν εκτελείται επιτόπου η εντολή, απλώς αντικαθίσταται το str με την εντολή που θέλουμε και συνεχίζεται η εκτέκεση της λούπας, επειδή μπορεί η εντολή να περιέχει alias, wild_characters, pipes κτλ). Αφού γίνει η διαδικασία ελέγχου για το αν έχει δοθεί το history κτλ μετακινώ μία θέση τα περιεχόμενα του history και βάζω στην πρώτη την τωρινή εντολή.

Σημείωση: Ο συνδυασμός pipes με redirection λειτουργεί γενικά, σε διάφορες δοκιμές μου, αλλά δεν λειτουργεί για το test case που μας έχει δοθεί, δηλαδή το

cat gene_with_protein_product.txt | cut -f2 > gene_with_protein_product_names_only_v2.txt

όπου “κολλάει” στην waitpid της όταν εκτελείται η cat.

Αν δοθεί κάποιο άλλο .txt στην cat τότε λειτουργεί κανονικά. Πχ :

cat kati.txt | cut -f2 > gene_with_protein_product_names_only_v2.txt

Δεν βρήκα τι φταίει.