DOCUMENTATION: ΑΣΚΗΣΗ 2η

Φοιτητής: Σουκαράς Σωτήρης ,

AM:6205,

email: soukaras@ceid.upatras.gr

1. ***mysh1:*** ***Μικρή περιγραφή:*** Με την πληκτρολόγηση από τον χρήστη του ./mysh1 μπαίνει μέσα στο shell. Η φιλοσοφία του mysh1 είναι ότι μέσα σε έναν ατέρμονο βρόγχο δέχεται τις εντολές που του δίνει ο χρήστης και βγαίνει από το shell μόνο όταν δοθεί η εντολή “exit” ή η εντολή ctrl^c. Συγκεκριμένα, σε κάθε εντολή που δίνει ο χρήστης και είναι exe δημιουργεί ένα πίνακα στο οποίο κάθε εντολή του promt αποθηκεύεται στον πινάκα. Στη συνέχεια για κάθε εντολή του πίνακα το shell δημιουργεί με την χρήση της εντολής fork() μια νέα διεργασία-παιδί η οποία το εκτελεί. Αν ο χρήστης δώσει την εντολή exit η οποία είναι buildin εντολή καθώς δεν βρίσκεται κάπου σαν exe δημιουργεί την εντολή πριν δώσει το δικαίωμα στην διεργασία-παιδί να την εκτελέσει.

   *****το κύριο πρόβλημα που είχα στο πρώτο shell ήταν η εντολή “exit”. Αρχικά, αντιμετώπιζα την εντολή όπως τις εντολές ls κτλ. Αυτό είχε ως αποτέλεσμα κάθε φορά που πληκτρολογούσα την εντολή exit να εμφανίζει το shell ότι δεν βρήκε την εντολή αυτή! Τελικά βρήκα ότι η εντολή “exit” είναι εντολή buildin και όχι exe όπως η ls κλπ. Δημιούργησα λοιπόν ένα μίνι πίνακα ο οποίος να βλέπει αν στην πρώτη του θέση έχει την εντολή exit και αν την έχει να φεύγει από το shell

   *****μέσα στο κώδικα χρησιμοποιώ την getline για κάνω μπορώ να επεξεργαστώ το input του χρήστη.

   *****γίνεται χρήση της exec και συγκεκριμένα της execvp γιατί αυτή η εντολή λαμβάνει υπόψιν το PATH και επιπλέον δίνει τη δυνατότητα εκτέλεσης ακόμα και εάν αριθμός των παραμέτρων δεν είναι γνωστός εξαρχής το οποίο χρειάζεται στο mysh1.

   *****για την ανάλυση και εκτέλεση της εκάστοτε εντολής που δίνει ο χρήστης χρησιμοποιήθηκε η εντολή strtok η οποία κάθε token (λεκτικό σύμβολο) που αποθηκευόταν και οριοθετούνταν από τους delimiters που είχαμε ορίσει .

2. ***Mysh2:*** ***Μικρή περιγραφή***:μέσα στη επανάληψη της Main τυπώνεται το Promt, έπειτα το shell διαβάζει τη γραμμή που δόθηκε από το χρήστη. Στη συνέχεια αναλύει τη γραμμή. Για κάθε εντολή που διαβάζει την αποθηκεύει στο array arguments . Στη συνέχεια κάθε θέση του array arguments έχει ότι έχει πληκτρολογήσει ο χρήστης και εκτελείται. Το shell τερματίζει με την εντολή exit. Για την cd εντολή συγκεκριμένα καλείται μια νέα συνάρτηση που ελέγχει αν στο array arguments στην πρώτη θέση υπάρχει το “cd” αν υπάρχει καλεί την chdir η οποία ελέγχει αν υπάρχει το directory που θα βρίσκεται στο argvp[1] και αν υπάρχει θα αλλάξει το τρέχων directory σε αυτό του argvp[1].

*****ο grader εβγαζε το σφαλμα: (HINT: Is your parsing flexible with white space? I.e., are you merging consecutive spaces/tabs into a single space?) , το πρόβλημα λύθηκε με τη χρήση επιπλέον ελέγχου για το αν ο χρήστης έβαζε EOF ή '\n' και το αντικαθιστούσε με '\0' λύνοντας το πρόβλημα.

*****κατά την δημιουργία του κώδικα υπήρχε το θέμα της διαχείρισης μνήμης. Πχ αν ο χρήστης έδινε μια εντολή που υπερέβαινε κατά πολύ το μέγεθος (του buffer πίνακα) τότε το πρόγραμμα ήταν αδύνατο να εκτελεστεί. Τη λύση έδωσε η εντολή της realloc που αύξανε το μέγεθος του buffer_input κατά το μέγεθος που είχαμε δώσει για το input_size στην αρχή.

3. ***Mysh3:*** ***Μικρή Περιγραφή:*** ο κώδικας του mysh3 ελέγχει και αυτός αν ο χρήστης δίνει εντολές buildin (cd,exit) και ανάλογα κάνει τις ίδιες λειτουργίες με τα προηγούμενα shell. Μέσα σε ένα βρόγχο επανάληψης ελέγχει αν ο χρήστης έχει δώσει στο το πίνακα το σύμβολο του pipe ("|") αν υπάρχει καλεί την συνάρτηση pipe() και χρησιμοποιώντας τους 2 fd(file descriptors) βάζει την εντολή στα αριστερά του Pipe που εκτελείται από το παιδί να γίνει stdin στην εντολή του πατέρα.

   *****χρήση της strlen για να μετρήσουμε το μέγεθος του input για να χρησιμοποιηθεί μετά ως check για το αν ο χρήστης έγραψε "|" και να εκτελεστεί η γραμμή που έγραψε.

   *****χρήση pipe(fd) όπου fd(file descriptors) είναι ένας πίνακας 2 θέσεων. Ο fd[0] αφορά το read και ο fd[1] το write.

   ***** γίνεται χρήση της dup2 αντί της dup, γιατί η dup2 μας αφήνει να διαλέξουμε τον fd που θα καθοριστεί και αυτομάτως τον κλείνει και τον αντικαθιστά άμα είναι ήδη πιασμένος.

4. **Mysh4:** ***Μικρή Περιγραφή:*** στο σχεδιασμό είναι ίδιος με το προηγούμενο shell.

   *****το πρόβλημα ήταν να μπορεί ο χρήστης να δώσει πολλαπλά Pipes ανάμεσα στις εντολές του. Το πρόβλημα λύθηκε με την εισαγωγή ενός πίνακα. Στο πίνακα αυτό αποθηκευόταν όλα τα tokes του χρήστη. Με την strlen μετράμε το μήκος της εντολής. Κάθε φορά που βρίσκουμε "|" το shell αποθηκεύει το σε ένα char και στον πίνακα του input το "|" αντικαθίσταται με whitespase. Ελέγχει αν το char είναι pipe και αν έχει εκτελεί την εντολή που βρίσκεται αριστερά και την διοχετεύει στην δεξιά εντολή κλπ , αυτό συνεχίζεται μέχρι στον πίνακα input αν μην υπάρχει pipe και ύστερα γίνεται free(input)

   *****όταν ο χρήστης έβαζε εντολή που ξεπερνούσε κατά πολύ το μέγεθος του δηλωμένου input προκαλούσε σφάλμα στην υλοποίηση. Τη λύση έδωσε η malloc

   *μέσα στην συνάρτηση parse_command χρησιμοποιείται η calloc αντί της malloc γιατί στην έξοδο αρχικοποιεί τα περιεχόμενα της δεσμευμένης μνήμης με μηδενικά το οποίο θέλουμε στη συγκεκριμένη περίπτωση ενώ με την malloc θα έπρεπε ν κάνουμε επιπλέον αυτή την λειτουργία.