ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Εξαμηνιαία εργασία μαθήματος

Δημιουργία διερμηνέα γραμμής εντολών(κέλυφος)

Μάστορας Ραφαήλ-Ευάγγελος
7918
Δεκέμβριος 2016

Περιγραφή του προγράμματος

Το πρόγραμμα που παραθέτω στο αρχείο myshell.c λειτουργεί με δύο τρόπους. Ο ένας απαιτεί την εισαγωγή από τον χρήστη των εντολών μέσω πληκτρολόγησης (Interactive) ενώ ο δεύτερος χρειάζεται κάποιο αρχείο που εμπεριέχει εντολές προς εκτέλεση (Batch file) . Παρέχει δυνατότητα εκτέλεσης πολλών εντολών ταυτόχρονα με την χρήση του τελεστή << ; >> , ωστόσο ελέγχει ο αριθμός των χαρακτήρων που εισάγουμε να μην υπερβαίνει το μέγιστο των 512 χαρακτήρων. Το μέγιστο στην περίπτωση της λειτουργίας Batch File είναι 512 χαρακτήρες ανά σειρά.

Αρχικά ή μεταγλώττιση του εκτελέσιμου αρχείου γίνεται με την εντολή make. Επίσης δίνεται η δυνατότητα διαγραφής του εκτελέσιμου με την εντολή make clean αλλά και η συμπίεση του κώδικα και του αρχείου Makefile σε ένα αρχείο myshell.tar μέσω της εντολής make release.
Αφότου κάνουμε compile τον πηγαίο κώδικα, δίνεται η δυνατότητα χρήσης του προγράμματος είτε ως Interactive mode με την εντολή **./myshell** ,είτε ως Batch File mode με την εντολή **./myshell filename ,**όπου filename είναι το όνομα του αρχείου μαζί με την προέκτασή του, παραδείγματος χάρη commands.txt .

Συνάρτηση main(int argc, char *argv[]) :
Ελέγχεται ότι τα ορίσματα του προγράμματος είναι ορθά δοσμένα αλλιώς εμφανίζει κατάλληλο μήνυμα. Εφόσον έχει επιλεγεί η Interactive λειτουργία , μέσα σε ένα ατέρμων loop τυπώνεται πρώτα mastoras_7918> και έπειτα περιμένουμε την εισαγωγή απο τον χρήστη μιας εντολής ή σειράς από εντολές με την συνάρτηση read_command(buffer). Ο buffer έχει τεθεί 4 φορές μεγαλύτερος από τις απαιτήσεις της εργασίας για να αποφευχθούν segmentation faults. Έπειτα καλείται η συνάρτηση split_arguments(buffer,args) η οποία χωρίζει τις εντολές μεταξύ τους κρατώντας τα ορίσματα κάθε μίας. Επιπλέον επιστρέφει τον συνολικό αριθμό των εντολών ώστε να χρησιμοποιηθεί αργότερα. Στην συνέχεια καλείται η συνάρτηση execute(commands,args) , η οποία είναι υπεύθυνη για την εκτέλεση των εντολών. Η συγκεκριμένη συνάρτηση επιστρέφει 0 ή 1 προκειμένου να τερματιστεί η όχι το πρόγραμμα, εφόσον βρει κάποια εντολή quit. Τέλος αρχικοποιούνται πάλι οι τιμές του buffer και του args ώστε να αποφευχθούν προβλήματα.
Παρόμοια λειτουργία παρουσιάζει η main και για Batch File mode. Η σημαντική διαφορά εδώ είναι πως αντί της συνάρτησης read_command χρησιμοποιείται η συνάρτηση read_file, η οποία διαβάζει από το αρχείο που εισάχθηκε σαν όρισμα, εφόσον βέβαια υπάρχει, τις εντολές ανά σειρά και τις εκτελεί. Η συνάρτηση αυτή επιστρέφει 0 η 1 ανάλογα αν φτάσαμε στο τέλος του αρχείου.

Συνάρτηση int read_file(char *buffer,char *argv[],int offset) :
Η συνάρτηση αυτή ανοίγει το αρχείο με όνομα αυτό που εισηχθεί από τον χρήστη κατά την εκτέλεση του προγράμματος. Εφόσον το αρχείο υπάρχει, το διαβάζει και εισάγει στον buffer μια σειρά κάθε φορά που καλείται. Ελέγχει σαφώς τα επιθυμητά όρια καθώς και αν το αρχείο είναι άδειο. Τέλος επιστρέφει 0 αν φτάσαμε στο τέλος του αρχείου ή 1 σε αντίθετη περίπτωση, προκειμένου να συνεχίσει το πρόγραμμα.

Συνάρτηση void read_command(char *buffer) :
Χρήση της συνάρτησης αυτής γίνεται στο Interactive mode. Μέσω της συνάρτησης fgets(buffer, 2048, stdin) εισάγεται στον buffer αυτό που έχουμε τυπώσει στην κονσόλα. Έχω μεριμνήσει ο τελευταίος χαρακτήρας να είναι ‘\0’ και να μη συμπεριληφθεί ο χαρακτήρας Enter ‘\n’. Τέλος ελέγχει αν το μήκος του buffer έχει ξεπεράσει τους 512 χαρακτήρες και τυπώνει κατάλληλο μήνυμα πριν τερματίσει την εκτέλεση του προγράμματος.

Συνάρτηση void removeSpaces(char *str1) :

Η συνάρτηση αυτή χρησιμεύει στην απαλοιφή των κενών χαρακτήρων. Χρειάστηκε να την χρησιμοποιήσω καθώς αντιμετώπιζα προβλήματα με την εκτέλεση κενών εντολών (; ;) διαφόρων μηκών.

Συνάρτηση int split_arguments(char *buffer,char *args[1000][512]) :

Η συγκεκριμένη συνάρτηση αναλαμβάνει να χωρίσει τις εντολές και τα ορίσματα τους. Αρχικά με χρήση της εντολής token=strtok(buffer,";") χωρίζει τις εντολές μεταξύ τους ενώ έπειτα για κάθε εντολή χωρίζω τα επιμέρους ορίσματα τους με χρήση της token=strtok(*args[i], DELIM). Η μεταβλητή DELIM έχει ορισθεί ως " \t\r\n\a". Τέλος επιστρέφει τον συνολικό αριθμό των εντολών.

Συνάρτηση int execute(int commands,char *args[1000][512]) :

Με αυτή την συνάρτηση γίνεται η εκτέλεση όλων των εντολών. Αρχικά δημιουργείτε ένας πίνακας pid [commands]. Για κάθε διαφορετικό command αντιστοιχεί και ένα κελί του πίνακα αυτού. Έπειτα δημιουργούνται child process , ένα για κάθε command. Εφόσον δημιουργηθούν με επιτυχία κάθε child ελέγχει την εντολή του. Αν αυτή είναι κενή εντολή, κάνει απλά exit, αν αυτή είναι εντολή quit μεταβάλει μία shared memory μεταβλητή σε 1 ,στις υπόλειπες περιπτώσεις απλά εκτελεί την εντολή. Στην περίπτωση που η εντολή δεν υπάρχει , δηλαδή η συνάρτηση execvp(*args[i], args[i]) επέστρεψε -1 , εμφανίζεται το μήνυμα “Unknown Command” .Σε κάθε περίπτωση μετά το πέρας της εκτέλεσής της εντολής γίνεται exit από το child process. Για την shared memory μεταβλητή έκανα χρήση μιας static global μεταβλητής καθώς και χρησιμοποίησα την εντολή

glob_var = mmap(NULL, sizeof *glob_var, PROT_READ | PROT_WRITE,MAP_SHARED | MAP_ANONYMOUS, -1, 0)

Η μητρική διεργασία σε όλη αυτή την διάρκεια περιμένει να τελειώσει η εκτέλεση όλων των εντολών με χρήση εντολής wait() και έπειτα ελέγχει αν έχει αλλαχθεί η τιμή της shared memory μεταβλητής σε 1. Εφόσον έχει αλλαχθεί επιστρέφει την τιμή 0, ώστε να τερματιστεί το πρόγραμμα. Σε κάθε άλλη περίπτωση επιτυχούς εκτέλεσης επιστρέφει 1 για την συνέχιση του προγράμματος.