ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

1η Εργασία

ΘΕΜΑ ΕΡΓΑΣΙΑΣ

Δημιουργία φλοιού Απομίμηση φλοιού Linux

Διδάσκων: Κος Ξυλωμένος

Εκπονήθηκε από τους:
Βασιλική Λαμπρινέα 3130113
Νικόλαος Παπασταματάκης 3130165
Γιάννης Παπαγεωργίου 3160129

Αρχικά δημιουργήσαμε ένα κοινό για όλα τα shell αρχείο το οποίο περιέχει μεθόδους που χρησιμοποιούνται σε όλες τις περιπτώσεις .Συγκεκριμένα περιέχει τις μεθόδους :

char * read1(): Η μέθοδος αυτή δέχεται την είσοδο που δίνει ο χρήστης και την βάζει σε ένα char *,το οποίο και επιστέφει.

char ** split_line(char * line): Χωρίζει την γραμμή σε tokens και επιστρέφει έναν πίνακα (char **) σε κάθε θέση του οποίου περιέχεται και ένα token.Κριτήριο διαχωρισμού μεταξύ των tokens είναι το κενό. (Ο διαχωρισμός επιτυγχάνεται μέσω της strtok.)

Void execute(char ** args): Κατά βάση είναι το αρχείο Fork_exec του εργαστηρίου και απλά ελέγχουμε το pid ,ώστε να λάβουμε την απόφαση αν υπάρχει σφάλμα, είτε πρέπει να περιμένουμε τον πατέρα είτε είμαστε σε θέση να καλέσουμε την execvp ώστε να γίνει η εκτέλεση της εντολής. Σε αυτή τη μέθοδο υλοποιείται δηλαδή και ο κατάλληλος έλεγχος ώστε να αποτρέπονται τις διαδικασίες zombie.

Void handle(char * args,int flag): Στη συγκεκριμένη μέθοδο με τη χρήση ενός flag αποφασίζουμε αν πρόκειται για αρχείο εισόδου ή αρχείο εξόδου, ανάλογα με τον τύπο του αρχείου καλείται με κατάλληλο τρόπο η open και ορίζονται τα ανάλογα δικαιώματα.

Void pipeline(char ** args): Αυτή η μέθοδος εφαρμόζεται στα shell4,5. Με την returnvalue ελέγχουμε αν εκτελέστηκε ορθά η pipe().Στην περίπτωση που δεν εκτελέστηκε ορθά τυπώνεται σχετικό μήνυμα σφάλματος, αλλιώς fork αν είμαστε στο παιδί κλείνουμε το input και εκτελούμε πάλι με execvp ,αν είμαστε στον πατέρα κλείνουμε το ρεύμα του output και περιμένει το παιδί να τελειώσει.

Void run2_3 : Αφορά τα shell2_3 ,εδώ δεσμεύουμε μνήμη για την μεταποίηση του args την οποία βάζουμε στο final.Όπου συναντάμε τα σύμβολα “<” ή” >” σημαίνει ότι έχουμε redirection και πρέπει να καλέσουμε με κατάλληλο τρόπο την handle(η ανάλυση της οποίας έγινε πιο πάνω).Αν συναντήσουμε το σύμβολο “>>” που λογίζεται ώς συντακτικό λάθος.Σε κάθε άλλη περίπτωση κρατάμε κάθε token στον final με την βοήθεια δικών μας μετρητών.

Void run 4(char** args,int filein,int fileout): Η μέθοδος αυτή αφορά αποκλειστικά τον τέταρτο φλοιό, με ανάλογες μεταβλητές μετράμε τα |,<,>,>> και τον αριθμό των tokens.Κάνουμε ελέγχους για να διαπιστώσουμε αν υπάρχει πάνω από 1 pipe ή κάποιο συντακτικό λάθος και αν είμαστε στον φλοιό 1(αριθμός tokens=1) ή απλά έχουμε απλές εντολές με απλές παραμέτρους(όχι <,>) π.χ. ls-l.Αν από τον έλεγχο προκύψει πως στην γραμμή που εισήγαγε ο χρήστης δεν περιέχεται το σύμβολο “|” βρισκόμαστε στον φλοιό 3,και καλούμε την run2_3 και execute,αλλιώς όποτε βρίσκουμε το σύμβολο “<” κάνουμε redirect input στο αρχείο(.txt) που βρίσκεται στο επόμενο token από το σύμβολο “<”,όταν βρίσκουμε το σύμβολο “ >” κάνουμε redirect output στο αρχείο που είναι στο επόμενο token και ενημερώνουμε την ανάλογη μεταβλητή .Όταν συναντάμε το σύμβολο”|” και δεν έχουμε κάνει redirection output καλούμε την pipeline ,αλλιώς τρέχουμε ως εδώ τις εντολές και ενημερώνουμε την ανάλογη μεταβλητή και κάνουμε memset ωστέ να μηδενίσουμε τα ήδη υπάρχοντα δεδομένα.

Void run 5(char** args,int filein,int fileout): Αυτή τη μέθοδο τη χρησιμοποιούμε αποκλειστικά τον φλοιό 5,έχει την δυνατότητα να τρέξει παραπάνω από μια σωληνώσεις και επιπλέον ό,τι έκαναν οι παραπάνω φλοιοί .Η λειτουργικότητά της είναι παρόμοια με την run4,απλώς δεν έχει όριο στις σωληνώσεις που μπορεί να δεχτεί και να εκτελέσει (σε ένα όριο πάντα 255 χαρακτήρων).

Εκτός από το κοινό για όλα τα shell αρχείο , δημιουργήσαμε και ένα headerfile στο οποίο γίνεται include για όλες τις βιβλιοθήκες που χρησιμοποιήσαμε και επιπλέον γίνεται η δήλωσης όλων των μεθόδων που χρησιμοποιήσαμε και η περιγραφή των οποίων έγινε παραπάνω.

Σχετικά με την υλοποίηση των φλοιών παρακάτω παραθέτουμε την λογική που ακολουθήσαμε όσον αφορά τα κοινά βήματα που εφαρμόσαμε σε όλους τους φλοιούς με τη δεδομένη σειρά:

1) Δεσμεύει μνήμη για την γραμμή που θα λάβει από τον χρήστη(μεγέθους 255 χαρακτήρων) και για τα args που θα χρησιμοποιηθούν για να τοποθετηθούν τα tokens στα οποία χωρίζουμε τη γραμμή.

2) 'Ελεγχος αν οι παραπάνω κλήσεις της malloc υλοποιήθηκαν ορθά

3) Σχετικό μήνυμα που υποδηλώνει τον φλοιό στον οποίο βρισκόμαστε(π.χ. auebsh1>)

4) Κάλεσμα της split_line για τον διαχωρισμό της γραμμής που εισάγει ο χρήστης σε tokes με κριτήριο το κενό που υπάρχει μεταξύ τους.

5) Τερματισμός του προγράμματος με εισαγωγή του Ctrl+d (Εάν εισαχθεί)

6) Εκτέλεση των εντολών με τη χρήση της συνάρτησης execute

7) Αποδέσμευση μνήμης που χρειάστηκε να δεσμεύσουμε

8) Διάβασμα νέας γραμμής από το χρήστη (σε περίπτωση που δεν πραγματοποιήθηκε κάποιο σφάλμα ή δεν πληκτρολογήθηκε Ctrl +D) και επανάληψη της εκάστοτε διαδικασίας για τη νέα γραμμή.

Σύντομη περιγραφή για κάθε φλοιό

Φλοιός 1: Αρχικά ο συγκεκριμένος φλοιός δεσμεύει μνήμη για την γραμμή που θα λάβει από τον χρήστη(μεγέθους 255 χαρακτήρων) και για τα args που θα χρησιμοποιηθούν για να τοποθετηθούν τα tokens στα οποία χωρίζουμε τη γραμμή. Με τη χρήση της συνάρτησης read1() διαβάζουμε την γραμμή που εισάγει ο χρήστης(για την οποία έχει πραγματοποιηθεί κατάλληλος έλεγχος ώστε να μην ξεπερνάει τους 255 χαρακτήρες.).Στη συνέχεια με την split_line τεμαχίζουμε την γραμμή σε tokens τα οποία χωρίζονται μεταξύ τους με κενό και κάθε token παίρνει από μια θέση στον πίνακα με τα args.Στο συγκεκριμένο φλοιό ο χρήστης έχει την δυνατότητα να πληκτρολογήσει μόνο απλές εντολές (τύπου ls,pwd)χωρίς τη χρήση παραμέτρων, για τον λόγο αυτό γίνεται κατάλληλος έλεγχος ώστε η γραμμή που θα πληκτρολογεί ο χρήστης να αποτελείται από μόνο ένα argument και η προσθήκη περισσότερων του ενός arguments να θεωρείται σφάλμα. Τελικά στην περίπτωση που ο χρήστης πληκτρολογήσει σωστά την εντολή καλείται από το πρόγραμμα μας η συνάρτηση execute με την οποία εκτελείται και η εκάστοτε εντολή. Για να τερματίσει ο χρήστης τον φλοιό αρκεί να πληκτρολογήσει Ctrl + D .Προσθέσαμε ακόμη έναν έλεγχο ώστε η πληκτρολόγηση του συμβόλου “>>” να θεωρείτε συντακτικό λάθος (όπως ορίζεται από τις απαιτήσεις τις εργασίας).Στην περίπτωση που δεν υπάρξει σφάλμα ή δεν πατήσει ο χρήστης Ctrl + D ο φλοιός 1 συνεχίζει να εκτελείται όσο ο χρήστης πληκτρολογεί νέες γραμμές.

Φλοιός 2: Ο φλοιός αυτός έχει τη δυνατότητα να εκτελεί όλες τις εντολές που εκτελεί και ο φλοιός 1 και επιπλέον δέχεται ανακατεύθυνση της τυπικής εισόδου/εξόδου από/προς αρχεία για τις εντολές που εκτελεί. Ωστόσο και ο συγκεκριμένος φλοιός δεν μπορεί να εκτελέσει εντολές με παραμέτρους και για αυτό το λόγο πραγματοποιούμε κατάλληλο έλεγχο. Συγκεκριμένα εφόσον οι παράμετροι των εντολών ξεκινούν με την χρήση του χαρακτήρα “-“, αποφασίσαμε να πραγματοποιήσουμε έλεγχο ο οποίος θα ελέγχει κάθε μια από τις λέξεις της γραμμής με σκοπό αν συναντήσει σε κάποια λέξη το “-“ σαν πρώτο χαρακτήρα το πρόγραμμα να εμφανίζει μήνυμα λάθους. Για το λόγο αυτό έχει χρησιμοποιηθεί η μεταβλητή “notok” η οποία ανάλογα με την τιμή της ορίζει αν υπάρχει κάποιο

invalid argument(τύπου –u ,δηλαδή παράμετρος).Αν δεν υπάρξει κάποιο σφάλμα δημιουργείται αντίγραφο με τη χρήση dup για τους fd μας (filein,fileout) , συνεχίζοντας καλείται η συνάρτηση handle ώστε να καθοριστεί αν πρόκειται για αρχείο εισόδου η εξόδου και ανάλογα με τον τύπου του αρχείου πραγματοποιούνται οι αντίστοιχες ενέργειες. Με τη χρήση dup2 επαναφέρουμε τους fd. Στην περίπτωση που δεν υπάρξει σφάλμα ή δεν πατήσει ο χρήστης Ctr + D ο φλοιός 2 συνεχίζει να εκτελείται όσο ο χρήστης πληκτρολογεί νέες γραμμές.

Φλοιός 3: Οι δυνατότητες αυτού του φλοιού είναι ίδιες με αυτές του παραπάνω φλοιού με την προσθήκη επιπλέον της δυνατότητας να δέχεται εντολές με παραμέτρους .Επομένως σε αυτή την περίπτωση έχουμε ακολουθήσει την λογική του φλοιού 2 αφαιρώντας ωστόσο τον έλεγχο που αποτρέπει την εισαγωγή παραμέτρων από τον χρήστη.

Φλοιός 4: Σε αυτόν τον φλοιό συναντάμε την μονή σωλήνωση , πάλι δημιουργούμε αντίγραφα μέσω της dup για τους fd ,όπως και στους δυο προηγούμενους φλοιούς και καλούμε την μέθοδο run4 (η ανάλυση της οποίας έγινε πιο πάνω) με κατάλληλα ορίσματα ,ώστε να επιτελέσει όλο το έργο και μετά καλούμε δυο φορές την dup2 για να κάνουμε την επιθυμητή ανάκτηση.

Φλοιός 5: Ο συγκεκριμένος φλοιός έχει τη δυνατότητα να εκτελεί ότι και όλοι οι υπόλοιπο και επιπλέον να υποστηρίζει πολλαπλή σωλήνωση. Σε προγραμματιστικό επίπεδο ακολουθούμε την ίδια διαδικασία με τον προηγούμενο φλοιό αντικαθιστώντας βέβαια τη μέθοδο τη run4 με την κλήση μιας νέας μεθόδου της run5 με τα ίδια ακριβώς ορίσματα η οποία ωστόσο δεν πραγματοποιεί έλεγχο ώστε να αποτρέπεται η χρήση παραπάνω από μία σωλήνωσης.

- ✓ Να σημειωθεί πως κάθε φλοιός έχει την δυνατότητα να επιτελέσει όχι μόνο το δικό του έργο αλλά και των από κάτω του σε ιεραρχία.
- ✓ Για την υλοποίηση της συγκεκριμένης εργασίας λάβαμε υπόψιν όλες τις υποδείξεις και απαιτήσεις που υπαγορεύονταν από την εκφώνηση της εργασίας.