Λειτουργικά Συστήματα

Αναφορά δεύτερης άσκησης

Κωνσταντίνος Μπουραντάς (23 6145)
bourantas@ceid.upatras.gr

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής, Πανεπιστήμιο Πατρών

31 Οκτωβρίου 2017

Κεφάλαιο 1

Documentation

Στην δεύτερη άσκηση θα πραγματοποιήσουμε την υλοποίηση ενός shell γραμμένο σε γλώσσα προγραμματισμού C. Υλοποιήθηκαν όλα τα ερωτήματα της άσκησης επιτυχώς . Ας ξεκινήσουμε λοιπόν την ανάλυση της υλοποίησης ξεκινώντας από την συνάρτηση main . Ξεκινάμε διαβάζοντας το input από τον χρήστη διαβάζοντας όλη την γραμμή την οποία έδωσε με την χρήση της συνάρτηση read_user_input .

```
printf("$ ");
    line =  read_user_input();

    args  = (char**)calloc(100, sizeof(char*));

    if (!args) {

      fprintf(stderr, "Allocation error : args\n");
      exit(EXIT_FAILURE);
    }
```

Έπειτα αφού αποθηκεύσουμε την γραμμή το πρόγραμμα μας μπαίνει σε ένα while-loop στο οποίο σε κάθε επανάληψη αντιγράφουμε το περιεχόμενο της γραμμής σε μια καινούργια μεταβλητή μέχρι να εντοπίσουμε , εάν τυχόν υπάρχει , ερωτηματικό . Στην περίπτωση ερωτηματικού αντιγράφουμε ότι βρήκαμε πριν από αυτό στην μεταβλητή newline . Σε κάθε επανάληψη κρατάμε την θέση στην οποία σταματήσαμε να αντιγράφουμε την γραμμή του χρήστη έτσι ώστε να συνεχίσουμε στην επόμενη εντολή.

```
 while (index < line_length) {
      i = 0;
      //printf("%d\n", line_length);
      newline = (char*)calloc(line_length , sizeof(char*));

      if (!newline) {

        fprintf(stderr, "Allocation error : newline\n");
        exit(EXIT_FAILURE);

      }

      while (line && (index < line_length) && (line[index] != ';')) {
        newline[i] = line[index];
        //printf("%c\n",newline[i] );
```

```
16
17          i++;
18          index++;
19        }
20
21        if (line[index] == ';') {
22          index++;
23        }
```

Στην ουσία αυτό που κάνουμε είναι να σπάμε την γραμμή του χρήστη σε ξεχωριστές γραμμές κάθε φορά και να τις εκτελούμε μια ανά φορά , έτσι το πρόγραμμα εκτελεί την μια εντολή μετά την άλλη αγνοώντας την ύπαρξη του ερωτηματικού.Στην συνάρτηση prepare_line_4exec(newline) επεξεργαζόμαστε το περιεχόμενο της κάθε προς εκτέλεσης γραμμή , μετρώντας εάν υπάρχουν pipes στην εντολή μας , το πλήθος των pipes , τον αριθμό των παραμέτρων κάθε εντολής πριν και μετά το pipe κλπ. και ενημερώνουμε κάποιες global μεταβλητές που θα μας χρησιμεύσουν στην μετέπειτα εκτέλεση των εντολών του χρήστη. Τέλος , κάνουμε σπάμε την γραμμή μας σε tokens και προχωράμε στην εκτέλεση των εντολών μας. Αν έχουμε μόνο μια εντολή χωρίς pipe τότε καλούμε μια πιο απλή συνάρτηση για την εκτέλεση , την launch , διαφορετικά καλούμε την launch_pipe . Σε περίπτωση της εντολής cd έχουμε μια ειδική συνάρτηση για της κλίση της.

```
1       if (strlen(line) > 1 && strlen(newline) > 1) {
2
3         args =  split_line(newline);
4
5         if (!strcmp(args[0], "exit")) {
6           exit(1);
7         }
8         else if (!strcmp(args[0], "cd")) {
9           change_dir(args);
10        }
11        else if (!strcmp(args[0], "about")) {
12          about_me();
13        }
14        else {
15          if (pipes_counter == 0) {
16            status = launch(args);
17          }
18          else {
19            status = launch_pipe(args);
20          }
21        }
```

Στην συνάρτηση launch κάνουμε fork μια διεργασία και έπειτα εάν το pid της είναι μηδέν καλούμε την execvp για την εκτέλεση της εντολής μας. Ο πατέρας της διεργασίας περιμένει μέχρι η διεργασία παιδί τελειώσει την εκτέλεση της. Στην συνάρτηση launch_pipe τα πράγματα είναι λίγο πιο σύνθετα. Αρχικά , καλούμε την συνάρτηση pipe ανάλογα με τον αριθμό των pipes που περιέχονται στην κάθε εντολή .

```
1  for (i = 0; i < pipes_counter; i++) {
2     if (pipe(pipes_fd + i * 2) < 0) {
3       fprintf(stderr, "Pipe[%d] failed!\n", i + 1);
4       exit(EXIT_FAILURE);
```

```
    }
  }
```

Αφού δημιουργήσουμε την είσοδο της εντολής execvp κάνουμε fork() για να δημιουργήσουμε ένα καινούργιο process έτσι ώστε να εκτελεστή η κάθε εντολή . Για κάθε εντολή δημιουργούμε ένα καινούργιο process μέσα στην επανάληψη. Στην συνεχεία εκτελούμε την εντολή dup2(pipes_fd[j – 2]) εάν η εντολή μας δεν είναι η πρώτη στην σειρά για να διαβάσει από το pipe της προηγούμενης εντολής. Διαφορετικά εάν είναι η πρώτη γραφεί στο ανάλογο pipe. Αφού η κάθε εντολή διαβάσει ότι πρέπει κλείνουμε όλα τα ανοιχτά φιλε descriptors και στην συνεχεία εκτελούμε την εντολή execvp για την εκτέλεση της κάθε εντολής. Τέλος κλείνουμε όλα τα pipes και το γονικό process περιμένει για όλα του τα παιδιά να τελειώσουν να εκτελούνται.

```
for (i = 0 ; i < total_cmds + 1 ; i++) {

   int loop_counter = 0;
   int index = seen_args;

   while (loop_counter < input_numb[i]) {

     exec_inputs[loop_counter] =  malloc(strlen(cmd[index]) * sizeof(char));
  //bug
     if (!exec_inputs[loop_counter]) {

       fprintf(stderr, "Allocation error\n");
       exit(EXIT_FAILURE);

     }

     strcpy(exec_inputs[loop_counter], cmd[index]);

     index++;
     loop_counter++;

   }

   exec_inputs[loop_counter] = NULL;

   pid = fork();

   if (pid == 0) {

     if (j != 0 ) {
       if (dup2(pipes_fd[j - 2], 0) < 0) {
         fprintf(stderr, "dup2 failed!\n");
         exit(EXIT_FAILURE);
       }
     }

     if (i < pipes_counter) {
       if (dup2(pipes_fd[j + 1], 1) < 0) {
         fprintf(stderr, "dup2 failed!\n");
```

```
39            exit(EXIT_FAILURE);
40          }
41        }
42
43        for (q = 0; q < 2 * pipes_counter; q++) {
44          close(pipes_fd[q]);
45        }
46
47        if (execvp(exec_inputs[0], exec_inputs ) < 0 ) {
48          fprintf(stderr, "MyShell: %s: command not found!\n",exec_inputs[0]);
49          exit(EXIT_FAILURE);
50        }
51      }
```

Η συγκεκριμένη υλοποίηση παραδόξως δεν παρουσίασε προβλήματα όσον αφορά το piping , όπως ήταν αναμενόμενο παρά μόνο ορισμένων λογικών λαθών, αλλά στην διαχείριση της μνήμης και των παρά πολλών pointers . Λόγω έλλειψης αρχικοποίησης των θέσεων μνήμης των μεταβλητών , οι μεταβλητές περιείχαν σκουπίδια . Οι τυχαίες τιμές αυτές όμως στο δικό μου σύστημα δεν ήταν μηδενικές οπότε όταν γινόταν έλεγχος για το memory allocation δεν υπήρχε κανένα λάθος .

```
1  args  = (char**)malloc(100, sizeof(char*));
2
3     if (!args) {
4
5       fprintf(stderr, "Allocation error : args\n");
6       exit(EXIT_FAILURE);
7
8     }
```

Όταν όμως έτρεχε στο σύστημα του grader οι τιμές των σκουπιδιών ήταν κυρίως μηδέν οπότε έπαιρνα ως αποτέλεσμα ότι όλη η υλοποίηση ήταν λανθασμένη. Το γεγονός αυτό με οδήγησε να ψάχνω για λάθη στο piping , δαπανώντας πολύ χρόνο . Μόλις ανακάλυψα το συγκεκριμένο bug, το έλυσα με την χρήση της συνάρτησης calloc αντί της malloc η οποία προσφέρει επιπλέον από την malloc και αρχικοποίηση της μνήμης .