Procesy		
Dominik Wróbel	21 III 2019	Czw. 17:00

Spis treści

1 Implementacja własnej powłoki

Zadanie

Ze względu na zwiększenie przejrzystości kodu, aplikacja powłoki została podzielona na trzy pliki. Proszę pobrać, skompilować i uruchomić aplikację

Pliki pobrano i skompilowano przy użyciu makefile.

Listing 1: makefile

```
shell: shell.o funcs.o
	gcc shell.o funcs.o -o shell -I.

funcs.o: funcs.c funcs.h
	gcc -c -Wall -pedantic funcs.c -I.

shell.o: shell.c funcs.h
	gcc -c -Wall -pedantic shell.c -I.
```

1.1. Zadanie 1

Zadanie

Przetestuj działanie powłoki wpisując polecenia

Listing 2: testowanie powłoki

```
@ ls

@ funcs.c  funcs.h  funcs.o  makefile  shell  shell.c  shell.o
ls -l

@ total 48
-rw-rw-r-- 1 dominik dominik  1439 mar 16 19:22 funcs.c
-rw-rw-r-- 1 dominik dominik  1780 mar 16 19:23 funcs.h
-rw-rw-r-- 1 dominik dominik  2392 mar 16 19:26 funcs.o
-rw-rw-r-- 1 dominik dominik   185 mar 16 19:26 makefile
-rwxrwxr-x 1 dominik dominik 13704 mar 16 19:26 shell
-rw-rw-r-- 1 dominik dominik  5415 mar 16 19:22 shell.c
-rw-rw-r-- 1 dominik dominik  4456 mar 16 19:26 shell.o
echo test

@ test
ps

@   PID TTY          TIME CMD
  3197 pts/0    00:00:00 bash
  3226 pts/0    00:00:00 shell
  3268 pts/0    00:00:00 ls <defunct>
  3269 pts/0    00:00:00 ls <defunct>
  3270 pts/0    00:00:00 echo <defunct>
  3271 pts/0    00:00:00 ps <defunct>

...
```

1.2. Zadanie 2

Pytanie

Dlaczego znak zachęty nie wyświetla się (dopiero) po wykonaniu procesu?

Znak zachęty @ jest wyświetlany zaraz po wpisaniu komendy, ponieważ proces wykonywania pętli powłoki nie jest przerywany gdy następuje stworzenie nowego procesu. Dzieje się tak dlatego, że komendy (programy) wpisane przez użytkownika są wykonywane w nowym procesie (tworzonym w funkcji executecmds przy pomocy funkcji fork), proces w którym znajduje się pętla nie jest jednak przerywany do czasu zakończenia wywoływanych procesów. Proces powłoki 'zatrzymuje się' dopiero gdy czeka na komendę od użytkownika, co dzieje się już po wyświetleniu znaku prompt.

1.3. Zadanie 3

Zadanie

Zmodyfikuj funkcję executecmds w taki sposób aby można było zakończyć działania powłoki poleceniem exit.

Listing 3: dodanie obsługi exit

```
int executecmds(struct cmdlist* __head)
    {
      int f, e;
      struct cmdlist* curr = __head;

    // obsluga exit
    int cmp = strcmp("exit", curr->argv[0]);
    if(cmp == 0){
      exit(RESSUCCESS);
    }


  while(curr != NULL){
  ...
```

1.4. Zadanie 4

Zadanie

Po uruchomieniu powłoki spróbuj uruchomić polecenie, które nie istnieje wpisując np. werwersdd. Następnie wpisz polecenie exit. Co się stało? Dlaczego polecenie nie działa poprawnie? Jak naprawić ten błąd?

Polecenie nie działa poprawnie, ponieważ nie kończy pracy procesu, który został utworzony przez fork, a który nie wykonał funkcji execvp z uwagi na niepoprawną komendę. W takiej sytuacji proces kontynuuje wykonanie kodu znajdującego się po execvp (wraca z execvp, gdyby komenda była poprawna, to nie wróciłby z tej funkcji), co powoduje, że pętla while nadal jest wykonywana, proces próbuje czytać dalsze polecenia i w wyniku tego dostajemy informacje o błędzie.
Błąd ten należy naprawić kończąc działanie procesu, który został stworzony w wyniku wpisania niepoprawnej komendy.

Listing 4: zakończenie procesu po wywołaniu niepoprawnej komendy

```
...
while(curr != NULL){
        f = fork();
        e = errno;

        if(f == 0){
          execvp(curr->argv[0], curr->argv);
          e = errno;
          printf("Error while executing: %s", strerror(e));
      exit(RESSUCCESS); // Zakonczenie procesu z bledna komenda
}
```

1.5. Zadanie 5

Zadanie

Zmodyfikuj funkcję executecmds w taki sposób aby oczekiwała na zakończenie się uruchomionego procesu oraz uzupełnij obsługę błędów nowoużytej funkcji. Przetestuj modyfikację jak w p. 1. i zwróć uwagę czy znak zachęty pojawia się dopiero po zakończeniu procesu.

Listing 5: dodanie obsługi czekania na proces dziecko

```
...
if(f == -1){
    printf("Fork error: %s", strerror(e));
    return RESERROR;
}

wait_ret = wait(&status);
if(wait_ret == -1){
  printf("Error while waiting for child process termination");
  exit(RESERROR);
}
...
```

Listing 6: testowanie wait

```
@ ls
funcs.c  funcs.h  funcs.o  makefile  shell  shell.c  shell.o  test

@ ls -l
total 52
-rw-rw-r-- 1 dominik dominik  1439 mar 16 19:22 funcs.c
-rw-rw-r-- 1 dominik dominik  1780 mar 16 19:23 funcs.h
-rw-rw-r-- 1 dominik dominik  2392 mar 16 19:26 funcs.o
-rw-rw-r-- 1 dominik dominik   185 mar 16 19:26 makefile
-rwxrwxr-x 1 dominik dominik 13912 mar 17 13:39 shell
-rw-rw-r-- 1 dominik dominik  5760 mar 17 13:38 shell.c
-rw-rw-r-- 1 dominik dominik  4984 mar 17 13:39 shell.o
-rw-rw-r-- 1 dominik dominik   743 mar 16 22:49 test

@
```

1.6. Zadanie 6

Zadanie

Zmodyfikuj funkcję executecmds w taki sposób aby do zmiennej int procres zapisywała wartość oznaczającą sposób zakończenia się ostatniego procesu (1 w przypadku pomyślnego zakończenia, 0 w przeciwnym wypadku) a następnie wyświetlała kod wyjścia procesu. Przetestuj zmiany.

Listing 7: dodanie zmiennej procres - wypisanie sposobu zakończenia procesu dziecka

```
...
if(f == -1){
      printf("Fork error: %s", strerror(e));
      return RESERROR;
}

wait_ret = wait(&status);
if(wait_ret == -1){
  printf("Error while waiting for child process termination");
  exit(RESERROR);
}
if(WIFEXITED(status)){
  procres = 0;
}else {
  procres = 1;
  }
printf("Process exited: %d", status);

curr = curr->next;
...
```

Listing 8: testowanie procres

```
@ echo test
test
Process exited: 0
@ ls
funcs.c  funcs.h  funcs.o  makefile  shell  shell.c  shell.o  test
Process exited: 0
@ ls hhh
ls: cannot access 'hhh': No such file or directory
Process exited: 512
@ cat aaa
cat: aaa: No such file or directory
Process exited: 256
@
```

1.7. Zadanie 7

Zadanie

Zmodyfikuj funkcję parsecmd, w taki sposób, aby poprawnie interpretowała operatory && oraz ||, a następnie zmodyfikuj funkcję executecmds, w taki sposób, aby uruchamiała procesy zgodnie z podanymi operatorami && oraz ||.

Zadanie

Sprawdź czy kolejny wyraz to operator && lub ||. Jeżeli tak to wykonaj:

1. Utwórz dynamicznie nową instancję struktury cmdlist i zapisz jej wskaźnik w polu next bieżącej.
2. Zakończ pracę z bieżącą instancją dodając NULL jako wartość ostatniego wskaźnika w tablicy argv. Użyj funkcji setupparsedcommand - pamiętaj o obsłudze błędów.
3. Ustaw nowo utworzoną instancję jako bieżącą.
4. Ustaw startowe wartości pól składowych nowo utworzonej struktury. Użyj funkcji setupnewcommand.
5. Zapisz w polu conjuction typ napotkanego operatora (patrz plik nagłówkowy).
6. Rozpocznij nową iterację pętli.

Listing 9: Funkcja parsecmd

```
/* 3. Parsing this command */
    int parsecmd(char* __buf, int __bufsize, struct cmdlist* __head)
    {
      char* cmd = __buf;                    /* String that must be parsed  */
      char* word;                           /* String between white characters  */
      struct cmdlist* curr = __head;
      struct cmdlist * new_struct;



      /* Reading next word - read strtok(3)  */
      while((word = strtok(cmd, " \t\n")) != NULL){


    // DODANIE ZNAKOW && ORAZ || ------------------------------------------------------------------
    //
    if(!strcmp("&&", word) || !strcmp("||", word)){
      // printf("Operator found");

      new_struct = malloc(sizeof *new_struct);
      curr->next = new_struct;
        if(setupparsedcommand(curr) == RESERROR){
            printf("Error while setting up parsed command.");
            return RESERROR;
          }
      curr = new_struct;
      setupnewcommand(curr);
      if(!strcmp("&&", word)){
        curr->conjuction = CONJAND;
      }
      if(!strcmp("||", word)){
        curr->conjuction = CONJOR;
      }
      continue;

    }

   // ------------------------------------------------------------------------------------------
    //

```

```
39          curr->argc++;
40          curr->argv = (char**)realloc(curr->argv, sizeof(char*)*curr->argc);    /* memory
    reallocation - needed for new argument  */
41          if(curr->argv == NULL){
42            printf("Error while allocating memory!");
43            return RESERROR;
44          }
45          curr->argv[curr->argc-1] = word;                                       /* Storing
    new argument in the argument vector in our structure */
46          cmd = NULL;
47        }
48
49        /* Setting up parsed command -- the NULL pointer at the end of the parameters list
    must added  */
50        if(setupparsedcommand(curr) == RESERROR){
51          printf("Error while setting up parsed command.");
52          return RESERROR;
53        }
54        return RESSUCCESS;
55  }
```

Zadanie

W przypadku funkcji executecmds, powinna ona uruchamiać procesy zgodnie z podanymi operatorami && oraz ||.

Listing 10: Funkcja executecmds - petla while

```
1  ...
2  while(curr != NULL){
3
4      if(flag == 0){
5
6          f = fork();
7          e = errno;
8
9          if(f == 0){
10           execvp(curr->argv[0], curr->argv);
11           e = errno;
12           printf("Error while executing: %s", strerror(e));
13       exit(RESSUCCESS);
14         }
15         if(f == -1){
16           printf("Fork error: %s", strerror(e));
17           return RESERROR;
18         }
19
20     wait_ret = wait(&status);
21     if(wait_ret == -1){
22       printf("Error while waiting for child process termination");
23       exit(RESERROR);
24     }
25     if(WIFEXITED(status)){
26       procres = 0;
27     }else {
28       procres = 1;
29     }
30     printf("Process exited: %d", status);
31     }
32
33       // OBSLUGA ZNAKOW && ORAZ ||
```

```
34      if(!procres){
35        if(curr->next->conjuction == 1){
36          printf("Conjuction OR break");
37          flag = 1;
38          curr = curr->next;
39          continue;
40        }
41      }
42      else{
43        flag = 0;
44      }
45
46
47      if(procres){
48        if(curr->next->conjuction == 2){
49          printf("Conjuction AND break");
50          flag = 1;
51          curr = curr->next;
52          continue;
53        }
54      }else{
55        flag = 0;
56      }
57
58
59      curr = curr->next;
60
61      // ------------------------------------------------------------------------------
62
63  }
64  ...
```