

Procesy		
Dominik Wróbel	21 III 2019	Czw. 17:00

Spis treści

1	Implementacja własnej powłoki	2
2	Ustawiania limitów procesu	5

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

```

1 shell: shell.o funcs.o
2     gcc shell.o funcs.o -o shell -I.
3
4 funcs.o: funcs.c funcs.h
5     gcc -c -Wall -pedantic funcs.c -I.
6
7 shell.o: shell.c funcs.h
8     gcc -c -Wall -pedantic shell.c -I.

```

Zadanie

Przetestuj działanie powłoki wpisując polecenia

Listing 2: testowanie powłoki

```

1 @ ls
2
3 @ funcs.c funcs.h funcs.o makefile shell shell.c shell.o
4 ls -l
5
6 @ total 48
7 -rw-rw-r-- 1 dominik dominik 1439 mar 16 19:22 funcs.c
8 -rw-rw-r-- 1 dominik dominik 1780 mar 16 19:23 funcs.h
9 -rw-rw-r-- 1 dominik dominik 2392 mar 16 19:26 funcs.o
10 -rw-rw-r-- 1 dominik dominik 185 mar 16 19:26 makefile
11 -rwxrwxr-x 1 dominik dominik 13704 mar 16 19:26 shell
12 -rw-rw-r-- 1 dominik dominik 5415 mar 16 19:22 shell.c
13 -rw-rw-r-- 1 dominik dominik 4456 mar 16 19:26 shell.o
14 echo test
15
16 @ test
17 ps
18
19 @
20 PID TTY          TIME CMD
21 3197 pts/0    00:00:00 bash
22 3226 pts/0    00:00:00 shell
23 3268 pts/0    00:00:00 ls <defunct>
24 3269 pts/0    00:00:00 ls <defunct>
25 3270 pts/0    00:00:00 echo <defunct>
26 3271 pts/0    00:00:00 ps <defunct>
27 ...

```

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.

Zadanie

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

Listing 3: dodanie obsługi `exit`

```

1 int executecmds(struct cmdlist* __head)
2 {
3     int f, e;
4     struct cmdlist* curr = __head;
5
6     // obsługa exit
7     int cmp = strcmp("exit", curr->argv[0]);
8     if(cmp == 0){
9         exit(RESSUCCESS);
10    }
11
12
13    while(curr != NULL){
14        ...

```

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

```

1 ...
2 while(curr != NULL){
3     f = fork();
4     e = errno;
5
6     if(f == 0){
7         execvp(curr->argv[0], curr->argv);

```

```

8     e = errno;
9     printf("Error while executing: %s", strerror(e));
10    exit(RESSUCCESS); // Zakonczenie procesu z bledna komenda
11 }

```

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

```

1 ...
2 if (f == -1){
3     printf("Fork error: %s", strerror(e));
4     return RESERROR;
5 }
6
7 wait_ret = wait(&status);
8 if (wait_ret == -1){
9     printf("Error while waiting for child process termination");
10    exit(RESERROR);
11 }
12 ...

```

Listing 6: testowanie wait

```

1 @ ls
2 funcs.c  funcs.h  funcs.o  makefile  shell  shell.c  shell.o  test
3
4 @ ls -l
5 total 52
6 -rw-rw-r-- 1 dominik dominik 1439 mar 16 19:22 funcs.c
7 -rw-rw-r-- 1 dominik dominik 1780 mar 16 19:23 funcs.h
8 -rw-rw-r-- 1 dominik dominik 2392 mar 16 19:26 funcs.o
9 -rw-rw-r-- 1 dominik dominik 185 mar 16 19:26 makefile
10 -rwxrwxr-x 1 dominik dominik 13912 mar 17 13:39 shell
11 -rw-rw-r-- 1 dominik dominik 5760 mar 17 13:38 shell.c
12 -rw-rw-r-- 1 dominik dominik 4984 mar 17 13:39 shell.o
13 -rw-rw-r-- 1 dominik dominik 743 mar 16 22:49 test
14
15 @

```

Zadanie

Zmodyfikuj funkcję `executecmds` w taki sposób aby do zmiennej `int` `procrs` 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 `procrs` - wypisanie sposobu zakończenia procesu dziecka

```

1 ...
2 if (f == -1){
3     printf("Fork error: %s", strerror(e));
4     return RESERROR;
5 }

```

```
6 wait_ret = wait(&status);
7 if(wait_ret == -1){
8     printf("Error while waiting for child process termination");
9     exit(RESEERROR);
10 }
11 if(WIFEXITED(status)){
12     procses = 0;
13 }else {
14     procses = 1;
15 }
16 printf("Process exited: %d", status);
17 curr = curr->next;
18 ...
19 ...
20 ...
```

Listing 8: testowanie proceses

```
1 @ echo test
2 test
3 Process exited: 0
4 @ ls
5 funcs.c funcs.h funcs.o makefile shell shell.c shell.o test
6 Process exited: 0
7 @ ls hhh
8 ls: cannot access 'hhh': No such file or directory
9 Process exited: 512
10 @ cat aaa
11 cat: aaa: No such file or directory
12 Process exited: 256
13 @
```

2 Ustawianie limitów procesu
