

Implementacja własnej powłoki

Ze względu na zwiększenie przejrzystości kodu, aplikacja powłoki została podzielona na trzy pliki:

shell.c - główny plik aplikacji, w tym pliku wprowadzamy modyfikacje w celu wykonania ćwiczeń.

funcs.h - plik nagłówkowy zawierający definicję struktury, stałych oraz deklaracje funkcji. Tego pliku nie trzeba modyfikować wystarczy się z nim zapoznać.

funcs.c - plik z funkcjami których nie trzeba modyfikować aby zrealizować ćwiczenia, jednak trzeba się z nim zapoznać.

Zagadnienia

Funkcje systemowe:

waitpid(2) -

fork(2) -

Funkcje operujące na stringach

strcmp(3) -

strtok(3) -

strchr(3) -

Funkcje zarządzające pamięcią:

realloc(3) -

free(3) -

Inne funkcje biblioteczne:

exec(3) -

execvp(3) -

Ćwiczenia

1. Przetestuj działanie powłoki wpisując polecenia np.:

```
@ ls
```

```
@ ls -l
```

```
@ echo test
@ ps
@ top
```

2. Dlaczego znak zachęty nie wyświetla się dopiero po wykonaniu procesu?

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

4. 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?

5. 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.

6. Zmodyfikuj funkcję `executecmds` w taki sposób aby do zmiennej `int proces` 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 wpisując np.:

```
@ echo test - powinno wyświetlić 0
@ ls - powinno wyświetlić 0
@ ls hhh - powinno wyświetlić wartość różną od 0 (o ile folder hhh nie istnieje)
@ cat aaa - powinno wyświetlić wartość różną od 0 (o ile plik aaa nie istnieje lub nie da się go przeczytać)
```

7. 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 `||`.

Podpowiedzi:

W przypadku funkcji `parsecmd`, powinna ona tworzyć listę instancji struktury `struct cmdlist`, np. polecenie:

```
ls -l && echo test || cat a.txt b.txt
```

powinno zostać przetworzone w następujący sposób:

Instance	argv	argc	conjunction	next
1	[0]=ls [1]=-l [2]=NULL	3	CONJUD	2

2	[0]=echo [1]=test [2]=NULL	3	CONJAND	3
3	[0]=cat [1]=a.txt [2]=b.txt [3]=NULL	4	CONJOR	NULL

Następnie przetestuj, czy interpretacja działa poprawnie: do tego celu może odkomentować funkcję `printparsedcmds` wywołaną w funkcji `main`.

Powyższe zadanie można wykonać implementując następujący algorytm:

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

7.1. Utwórz dynamicznie nową instancję struktury `cmdlist` i zapisz jej wskaźnik w polu `next` bieżącej.

7.2. Zakończ pracę z bieżącą instancją dodając `NULL` jako wartość ostatniego wskaźnika w tablicy `argv`. Użyj funkcji `setupparsedcommand` - pamiętaj o obsłudze błędów.

7.3. Ustaw nowo utworzoną instancję jako bieżącą.

7.4. Ustaw startowe wartości pól składowych nowo utworzonej struktury. Użyj funkcji `setupnewcommand`.

7.5. Zapisz w polu `conjunction` typ napotkanego operatora (patrz plik nagłówkowy).

7.6. Rozpocznij nową iterację pętli.

W przypadku funkcji `executecmds`, powinna ona uruchamiać procesy zgodnie z podanymi operatorami `&&` oraz `||`. Można to przetestować przy pomocy np. takich wywołań:

```
@ ls || echo OK - wynik: lista plików bez napisu OK
```

```
@ ls && echo OK - wynik: lista plików oraz napis OK
```

```
@ ls || echo OK && ps - wynik: lista plików bez napisu OK oraz lista procesów
```

```
@ ls && echo OK || ps - wynik: lista plików oraz napis OK bez listy procesów
```

```
@ ls nieistnieje && echo OK || ps - wynik: błąd ls oraz lista procesów
```

```
@ ls nieistnieje || echo OK && ps - wynik: błąd ls, napis OK oraz lista procesów
```

```
@ ls || echo OK && ps || wc aaa - wynik: lista plików oraz procesów
```

```
@ ls || echo OK && wc aaa || ps - wynik: lista plików, błąd wc oraz lista procesów
```