Systemy operacyjne

Projekt programistyczny nr 1

„Powłoka uniksowa"

Termin oddawania: 29 grudnia 2019

Przed przystąpieniem do prac programistycznych należy zapoznać się z następującymi rozdziałami książek i dokumentacją:

- The Linux Programming Interface: 34.5 – 34.7
- Advanced Programming in the Unix Environment: 9.6 – 9.9
- The GNU C Library: Job Control[1]

Wymogi formalne

Należy pobrać ze strony przedmiotu plik «`so19_projekt-shell.tar.gz`», rozpakować go i zapoznać się z plikami źródłowymi. Zadaniem studenta jest uzupełnienie brakujących fragmentów programu oznaczonych komentarzem zawierającym napis «`TODO`».

Jeśli to tylko możliwe należy unikać modyfikacji pozostałych fragmentów kodu. Gdyby jednak okazało się, że musisz dodać nowe procedury lub zmodyfikować istniejący kod, to należy to solidnie uzasadnić w pliku tekstowym «`README.md`».

Oceniający Twój kod będzie mógł odejmować punkty, jeśli będziesz utrudniać mu pracę. Należy tak zorganizować kod, by łatwo się go czytało tj.: starannie wybierać nazwy procedur i zmiennych, pamiętać o formatowaniu kodu, dbać o przejrzystość struktury programu, dodawać komentarze do fragmentów, których zrozumienie wymaga większego wysiłku.

Wszystkie pliki źródłowe rozwiązania należy skopiować do katalogu o nazwie «`indeks_imie_nazwisko`», po czym spakować go do archiwum w formacie «`tar.gz`» o nazwie «`indeks_imie_nazwisko.tar.gz`» i umieścić w systemie SKOS. Po rozpakowaniu projekt ma mieć następującą strukturę katalogów:

```
999999_jan_nowak/
  libcsapp/
    ...
  include/
    ...
  README.md
  Makefile
  Makefile.include
  command.c
  jobs.c
  lexer.c
  shell.c
  shell.h
```

Oceniający zadania używa komputera z zainstalowanym systemem `Debian GNU/Linux 10` dla architektury `x86-64`. Ściąga z systemu SKOS archiwum dostarczone przez studenta, po czym sprawdza poprawność struktury katalogów. Następnie czyta plik `README.md` z uwagami od studenta i buduje projekt używając pliku `Makefile`. Oceniający będzie mógł odejmować punkty, jeśli będzie mieć problem z kompilacją rozwiązania.

[1] `https://www.gnu.org/software/libc/manual/html_node/Job-Control.html`

Lekser i parser

Lekser dzieli ciąg znaków na tokeny i zwraca tablicę wskaźników do ciągów znakowych. Specjalne tokeny zastępuje wskaźnikami o wartościach numerycznych (np. «T_BGJOB», «T_OUTPUT», ...). Na przykład: ciąg znaków "grep foo < in.txt | wc -l &" zostanie przetworzony do tablicy wartości typu «token_t»: {"grep", "foo", T_INPUT, "in.txt", T_PIPE, "wc", -l", T_BGJOB, T_NULL}. Tablicę tokenów można modyfikować w trakcie przetwarzania polecenia.

Implementacja parsera i interpretera wyrażeń niestety nie są od siebie oddzielone. Zakładamy, że «shell» akceptuje wyłącznie poprawnie uformowane wyrażenia według poniższej gramatyki (w formacie dla programu bison). Nie musisz się przejmować zachowaniem powłoki dla źle uformowanych wyrażeń.

```
%start program
%%
program          : pipe_sequence '&'
                 | pipe_sequence
                 ;
pipe_sequence    : command
                 | pipe_sequence '|' command
                 ;
command          : cmd_words cmd_suffix
                 | cmd_words
                 ;
cmd_words        : cmd_words WORD
                 | WORD
                 ;
cmd_suffix       : io_redirect
                 | cmd_suffix io_redirect
                 ;
io_redirect      : '<' WORD
                 | '>' WORD
                 ;
```

Zadania

Zadanie to grupa procesów utworzonych w wyniku przetwarzania pojedynczego polecenia powłoki. Zadanie może składać się z wielu procesów i wtedy nazywamy je potokiem. Zadania dzielimy na pierwszoplanowe (ang. *foreground*) i drugoplanowe (ang. *background*).

Powłoka zawsze oczekuje na zakończenie zadania pierwszoplanowego zanim będzie gotowa do wykonania kolejnego polecenia. Jednym z zadań powłoki jest przypisanie terminala sterującego do grupy procesów zadania pierwszoplanowego. Dzięki temu sterownik terminala wysyła, między innymi, sygnały «SIGINT» i «SIGTSTP» w wyniku naciśnięcia klawiszy «CTRL+C» i «CTRL+Z». Po zakończeniu zadania (stan FINISHED) lub przeniesieniu go w tło (zadanie drugoplanowe) należy oddać terminal sterujący grupie procesów powłoki. Zadanie pierwszoplanowe ma zarezerwowany slot 0 w tablicy zadań «jobs».

Zadania drugoplanowe można utworzyć na dwa sposoby: albo dopisując znak «&» na koniec polecenia, albo zatrzymując zadanie naciskając klawisze «CTRL+Z». W pierwszym przypadku polecenie jest aktywne (stan RUNNING), natomiast w drugim zatrzymane (stan STOPPED). Powłoka może nadzorować wiele zadań drugoplanowych. Z użyciem procedury «watchjobs» powłoka wyświetla zakończone zadania drugoplanowe przed pokazaniem znaku zachęty (ang. *prompt*).

Zarządzanie zadaniami

Powłoka rozpoznaje kilka wbudowanych poleceń. Procedura «builtin_command» zwraca wartość ujemną, jeśli polecenie nie zostało rozpoznane jako wbudowane. Dla nas najważniejsza jest implementacja poleceń służących do zarządzania zadaniami (ang. *job control*). Parametr w nawiasach kwadratowych jest opcjonalny. Jeśli nie zostanie podany, to zostaje wybrane zadanie o najwyższym numerze.

- `fg [n]`: zmienia zatrzymane lub uruchomione zadanie drugoplanowe na pierwszoplanowe,
- `bg [n]`: zmienia stan zadania drugoplanowego z zatrzymanego na aktywne,
- `kill %n`: uśmierca procesy należące do zadania o podanym numerze,
- `jobs`: wyświetla stan zadań drugoplanowych.

W przypadku polecenia «kill» występuje konflikt nazw z poleceniem zewnętrznym. Jeśli pierwszy argument do «kill» nie zaczyna się od znaku %, to «do_kill» zwraca wartość ujemną.

Polecenia

Powłoka wykonuje polecenia zewnętrzne i wewnętrzne w ramach zlecanych jej zadań. Poniżej widnieje kilka przykładów istotnie różnych poleceń akceptowanych przez powłokę:

- «`cd ..`»: polecenie wewnętrzne «cd»,
- «`cd .. &`»: j.w. ale jako zadanie drugoplanowe w podprocesie (możliwe, ale nie ma sensu),
- «`ls -l`»: polecenie zewnętrzne uruchomione w podprocesie i monitorowane dopóki podproces się nie zakończy lub nie zostanie zatrzymany,
- «`ls -l &`»: polecenie zewnętrzne uruchomione jako zadanie drugoplanowe, powłoka od razu przechodzi do wyświetlenia znaku zachęty,
- «`cd .. | ls -l`»: zadanie pierwszoplanowe, obydwa polecenia są uruchomione w osobnych podprocesach mimo, że pierwsze jest wewnętrzne, a drugie zewnętrzne,
- «`cd .. | ls -l &`»: zadanie drugoplanowe, reszta jak wyżej,
- «`ls -l | wc -l`»: zadanie pierwszoplanowe, procesy muszą być prawidłowo połączone potokiem.

Przekierowania

Zadaniem procedury «do_redir» jest przetworzenie i usunięcie tokenów odpowiedzialnych za przekierowania. Wartość «n» zwracana z «do_redir» to liczba tokenów, które zostaną przekazane do `execve(2)`. Argumenty «intputp» i «outputp» wskazują na miejsce, w którym zostaną zapisane deskryptory plików otwarte przez procedurę «do_redir». Ciąg tokenów {"grep", "foo", T_INPUT, "test.in", T_OUTPUT, "test.out"} (ntokens=6) zostanie przetworzony do {"grep", "foo", T_NULL, T_NULL, T_NULL, T_NULL} (n=2). W wyniku wykonania procedury «do_redir» deskryptor, na który wskazuje «inputp» i «outputp», odpowiada plikowi «test.in» otwartemu do odczytu i plikowi «test.output» otwartemu do zapisu.

W przypadku użycia przekierowania do pliku w zadaniu będącym potokiem zachowanie jest niezdefiniowane, np. «`grep foo test.in > test.out | wc -l`».

Wycieki deskryptorów plików

Po utworzeniu zadania powłoka powinna mieć otwarte tylko cztery deskryptory plików: «stdin», «stdout», «stderr» i «tty_fd», co można sprawdzić przy pomocy polecenia wydanego w zewnętrznej powłoce:

```
bash$ ls -l /proc/$(pgrep shell)/fd
(...) 0 -> /dev/pts/2
(...) 1 -> /dev/pts/2
(...) 2 -> /dev/pts/2
(...) 3 -> /dev/pts/2
```

W trakcie tworzenia przekierowań i potoków należy zadbać o to, żeby do podprocesów nie trafiały niepożądane deskryptory plików. Innymi słowy powłoka ma przekazywać do podprocesów wyłącznie deskryptory plików «stdin», «stdout» i «stderr». Można to sprawdzić wydając poniższe polecenia, przy czym deskryptor 3 to katalog otwarty przez polecenie «ls».

```
# ls -l /proc/self/fd
(usunięto)
# ls -l /proc/self/fd | cat
(usunięto)
# echo | ls -l /proc/self/fd
(usunięto)
# echo | ls -l /proc/self/fd | cat
(...) 0 -> pipe:[662015934]
(...) 1 -> pipe:[662015935]
(...) 2 -> /dev/pts/2
(...) 3 -> /proc/13200/fd
```