Технологично училище "Електронни системи"
Технически университет - София

Операционни системи

Задача №2: Команден интерпретатор - части I и II

---

os@elsys-bg.org

27 ноември 2017 г.

1 Условие на задачата

1.1 Основна функционалност (20 точки)

Целта на задачата е да се реализира прост команден интерпретатор `shell`.

При стартиране на програмата, тя започва да чете редове от стандартния вход и да ги интерпретира. За тази цел програмата трябва да раздели командния ред на думи, като за разделител се използва интервал (`' '`).

Например, ако на стандартния вход се напише следния ред:

```
/bin/ls -l /usr/include
```

то командния интерпретатор трябва го превърне в следния масив от думи:

```
{"/bin/ls", "-l", "/usr/include"}
```

Първата дума се интерпретира като име на файл, който трябва да се изпълни. Програмата трябва да се опита да изпълни този файл, а като аргументи от командния ред трябва се предаде масив от всички думи в командния ред.

В разгледания пример при команден ред:

```
/bin/ls -l /usr/include
```

програмата трябва да се опита да изпълни файла и да му предаде следния масив от аргументи:

```
{"/bin/ls", "-l", "/usr/include"}
```

1.2 Създаване на pipe (40 точки)

Интерпретаторът трябва да се развие като се добави възможност за създаване на pipe между две (или повече) приложения (команди).

Програмата трябва да третира по специален начин символът `|`, който се използва за създаване на pipe между приложения.

Например, ако на интерпретатора се подаде командата:

```
/bin/ls -l /usr/include | /bin/wc
```

то командния интерпретатор трябва:

- да създаде два процеса;
- да създаде pipe между двата процеса, така че стандартния изход на първия процес да бъде насочен към стандартния вход на втория процес;
- в рамките на първия процес да изпълни командата `/bin/ls -l /usr/include`;
- в рамките на втория процес да изпълни командата `/bin/wc`.

1.3 **Обработка на грешки** (20 **точки**)

Ако първата дума от командният ред не е име на файл, то програмата трябва да изведе съобщение `No such file or directory`. Например, ако на стандартния вход на програмата се въведе `./aa` и в текущата директория няма файл с такова име, то програмата трябва да формира следното съобщение:

```
./aa
./aa: No such file or directory
```

1.4 Допълнителни изисквания

За обработка на командия ред в рамките на програмата трябва да се дефинира функция `parse_cmdline`, която трябва да има следната сигнатура:

```
char **parse_cmdline( const char *cmdline );
```

Функцията `parse_cmdline()` трябва да приема като аргумент C-string и да връща масив от C-стрингове. Върнатия масив от C-стрингове трябва да бъде оформен така, че да може директно да се подаде като втори аргумент на функция.

```
execv( const char *filename, char *const argv[] );
```

За изпълнение на подадените команди трябва да се използва комбинация от обръщания към функциите

```
pid_t fork( void );
```

и

```
execv( const char *filename, char *const argv[] );
```

След завършване на работата на създадения по този начин процес, родителския процес задължително трябва да провери статуса на завършване на детето като използва някой от вариантите на функциите `wait*`, например функцията

```
pid_t waitpid( pid_t pid, int *status, int options );
```

Командния интерпретатор трябва да позволява организиране на pipe между две (или повече) приложения (команди).

За пренасочване на стандартния вход/изход трябва да се използва системната функция:

```
int dup2( int old, int new );
```

За създаване на pipe трябва да се използва системната функция:

```
int pipe( int filedesc[2] );
```

2 Изисквания към решението и оценяване

2.1 Решението на задачата трябва да бъде написано на езика C съгласно ISO/IEC 9899:2011;

2.2 Правилата за оценяване са следните. Приемаме, че напълно коректна и написана спрямо изискванията програма получава максималния брой точки - 100%.
Ако в решението има пропуски, максималният брой точки ще бъде намален съгласно правилата описани по-долу;

2.3 За работа с процеси трябва да се използва семейството от функции за работа с процеси от ниско ниво - `fork()`, `wait()`, `waitpid()`, `execve()` и т.н. Неспазването на това изискване води до оценяване на задачата с 0 точки;

2.4 Задължително към файловете с решението трябва да е приложен и `Makefile`. Изпълнимият файл, който се създава по време на компилация на решението, трябва да се казва `shell`;

2.5 При проверка на решението програмата ви ще бъде компилирана и тествана по следния начин:

```
make
./shell .
```

Предходната процедура ще бъде изпълнена няколко пъти с различни входни данни за да се провери дали вашата програма работи коректно;

2.6 Реализацията на програмата трябва да спазва точно изискванията. Всяко отклонение от изискванията ще доведе до получаване на 0 точки за съответната част от условието;

2.7 Работи, които са предадени по-късно от обявеното (или не са предадени), ще бъдат оценени с 0 точки;

2.8 Решението ви трябва да съдържа достатъчно коментари. Оценката на решения без коментари или с недостатъчно и/или мъгляви коментари ще бъде намалена с 30%;

2.9 Всеки файл от решението трябва да започва със следният коментар:

```
//---------------------------------------------------------------------
// NAME: Ivan Ivanov
// CLASS: XIa
// NUMBER: 13
// PROBLEM: #1
// FILE NAME: xxxxxx.yyy.zzz (unix file name)
// FILE PURPOSE:
// няколко реда, които описват накратко
// предназначението на файла
// ...
//---------------------------------------------------------------------
```

Всяка функция във вашия изходен код трябва да включва кратко описание в следния формат:

```
//---------------------------------------------------------------------
// FUNCTION: xxyyzz (име на функцията)
// предназначение на функцията
// PARAMETERS:
// списък с параметрите на функцията
// и тяхното значение
//---------------------------------------------------------------------
```

2.10 Лош стил на програмиране и липсващи заглавни коментари ще ви костват 30%;

2.11 Решения, които не се компилират получават 0 точки. Под "не се компилират" се има предвид произволна причина, която може да причини неуспешна компилация, включително липсващи файлове, неправилни имена на файлове, синтактични грешки, неправилен или липсващ `Makefile`, и т.н. Обърнете внимание, че в UNIX имената на файловете са "case sensitive";

2.12 Решения, които се компилират, но не работят, не могат да получат повече от 50%. Под "компилира се, но не работи" се има предвид, че вие сте се опитали да решите проблема до известна степен, но не сте успели да направите пълно решение. Често срещан проблем, който спада към този случай, е че вашият `Makefile` генерира изпълним файл, но той е именуван с име, различно от очакваното (т.е. `shell` в разглеждания случай);

2.13 Безсмислени или мъгляви решения ще бъдат оценявани с 0 точки, независимо че се компилират;

2.14 Решения, които дават неправилни или непълни резултати, или решения, в които изходът и/или форматирането се различава от изискванията ще получат не повече от 70%;

2.15 Всички наказателни точки се сумират. Например, ако вашето решение няма задължителните коментари в началото на файлове и функциите се отнемат 30%, ако няма достатъчно коментари се отнемат още 30%, компилира се, но не работи правилно - още 30%, то тогава резултатът ще бъде: (100 - 30 - 30 - 30)% = 10%;

2.16 Работете самостоятелно. Групи от работи, които имат твърде много прилики една с друга, ще бъдат оценявани с 0 точки;