2020-11-17

Занятие #11.

Задача 11.1.

В вашем распоряжении есть файл dumbsh_v1.c, прилагаемый к лекции.
Почему именно команды, содержащие более одного слова, терпят неудачу? Ответ следует искать в том, как мы выполняем преемника, используя API execlp(3). Напомню, что для execlp мы должны передать имя программы (поиск PATH, конечно, будет выполняться автоматически) и все аргументы, начиная с argv[0]. В этой простой реализации мы просто не передаем ничего, кроме первого аргумента argv[0]. Все дело именно в этом.
Как заставить оболочку работать с командами с любым количеством аргументов? Это потребует обработки строк: вам нужно будет преобразовать аргументы в отдельные строки, инициализировать массив argv указателей на них и использовать этот argv через execv[pe] API.
Подсказка: библиотека C предоставляет API для токенизации строк; strtok(3), strtok_r(3).
Вывод вашей программы должен выглядеть примерно так. Обратите внимание на сообщения.

```
$ ./dumbsh_v2 -v
>> ps
Parent process (2095) issuing the wait...
Child process (2096) exec-ing cmd "ps" now..
PID TTY TIME CMD
1078 pts/0 00:00:00 bash
2095 pts/0 00:00:00 dumbsh_v2
2096 pts/0 00:00:00 ps
Child (2096) status changed:
normal termination: exit status: 0
>> quit
Parent process (2095) exiting...
$
```

(запуск программы в verbose режиме (-v). У дочернего процесса ps(1) произошло изменение статуса: он терминировался нормально, статус выхода равен нулю, т.е. успех.)

Задача 11.2.

Убедитесь, что родительский процесс действительно ожидает оба дочерних процесса (см. fork2.c в материалах лекции). Чтобы родитель ждал всех возможных потомков, вызовите API wait как условие цикла while. Пока ожидаемые дочерние процессы существуют, он будет блокировать и возвращать положительный результат. В момент отсутствия ожидающих потомков wait возвращает -1; проверьте это как условие выхода из цикла. Обратите внимание, однако, что существуют сценарии, требующие установки неблокирующего ожидания.