

Параметры функции `main()` в языке С. Переменные среды и аргументы командной строки

У функции `main()` в языке программирования С существует три параметра, которые могут быть переданы ей операционной системой. Полный прототип функции `main()` выглядит следующим образом:

```
int main(int argc, char *argv[], char *envp[]);
```

Первые два параметра при запуске программы на исполнение командной строкой позволяют узнать полное содержание командной строки. Вся командная строка рассматривается как набор слов, разделенных пробелами. Через параметр `argc` передается количество слов в командной строке, которой была запущена программа. Параметр `argv` является массивом указателей на отдельные слова. Так, например, если программа была запущена командой

```
a.out 12 abcd
```

то значение параметра `argc` будет равно 3, `argv[0]` будет указывать на имя программы — первое слово — `"a.out"`, `argv[1]` — на слово `"12"`, `argv[2]` — на слово `"abcd"`. Так как имя программы всегда присутствует на первом месте в командной строке, то `argc` всегда больше 0, а `argv[0]` всегда указывает на имя запущенной программы.

Анализируя в программе содержимое командной строки, мы можем предусмотреть её различное поведение в зависимости от слов, следующих за именем программы. Таким образом, не внося изменений в текст программы, мы можем заставить её работать по-разному от запуска к запуску. Например, компилятор `gcc`, вызванный командой `gcc 1.c` будет генерировать исполняемый файл с именем `a.out`, а при вызове командой `gcc 1.c -o 1.exe` — файл с именем `1.exe`.

Третий параметр — `envp` — является массивом указателей на параметры окружающей среды процесса. Начальные параметры окружающей среды процесса задаются в специальных конфигурационных файлах для каждого пользователя и устанавливаются при входе пользователя в систему. В дальнейшем они могут быть изменены с помощью специальных команд операционной системы UNIX. Каждый параметр имеет вид: *переменная=строка*. Такие переменные используются для изменения долгосрочного поведения процессов, в отличие от аргументов командной строки. Например, задание параметра `TERM=vt100` может говорить процессам, осуществляющим вывод на экран дисплея, что работать им придется с терминалом `vt100`. Меняя значение переменной среды `TERM`, например на `TERM=console`, мы сообщаем таким процессам, что они должны изменить своё поведение и осуществлять вывод для системной консоли.

Размер массива аргументов командной строки в функции `main()` мы получали в качестве её параметра. Так как для массива ссылок на параметры окружающей среды такого параметра нет, то его размер определяется другим способом. Последний элемент этого массива содержит указатель `NULL`.

Посмотреть список всех переменных среды, установленных для командного интерпретатора можно с помощью команды `env`. Учтите, что выдача этой команды достаточно громоздкая.

Изменить значение переменной среды или создать новую переменную среды с установлением ей значения можно с помощью команды `export` в форме

```
export переменная=строка
```

Удалить переменную среды можно с помощью команды `unset`:

```
unset переменная
```

Команда `export` без параметров продемонстрирует список всех экспортированных переменных среды.

Изменение пользовательского контекста процесса. Семейство функций для системного вызова `exec()`

Для изменения пользовательского контекста процесса применяется системный вызов `exec()`, который пользователь не может вызвать непосредственно. Вызов `exec()` заменяет пользовательский контекст текущего процесса на содержимое некоторого исполняемого файла и устанавливает начальные значения регистров процессора (в том числе устанавливает программный счётчик на начало загружаемой программы). Этот вызов требует для своей работы задания имени исполняемого файла, аргументов командной строки и параметров окружающей среды. Для осуществления вызова программист может воспользоваться одной из шести функций: `execvp()`, `execlp()`, `execl()`, `execv()`, `execle()`, `execve()`, отличающихся друг от друга представлением параметров, необходимых для работы системного вызова `exec()`. Взаимосвязь указанных выше функций изображена на рисунке.



Функции изменения пользовательского контекста процесса

Прототипы функций

```
#include <unistd.h>
int execlp(const char *file, const char *arg0, ... const char *argN, (char *)NULL)
int execvp(const char *file, char *argv[])
int execl(const char *path, const char *arg0, ... const char *argN, (char *)NULL)
int execv(const char *path, char *argv[])
int execl(const char *path, const char *arg0, ... const char *argN, (char *)NULL,
char *envp[])
int execve(const char *path, char *argv[], char *envp[])
```

Описание функций

Для загрузки новой программы в системный контекст текущего процесса используется семейство взаимосвязанных функций, отличающихся друг от друга формой представления параметров.

Аргумент `file` является указателем на имя файла, который должен быть загружен. Файл должен находиться в директории из переменной `PATH`.

Аргумент `path` – это указатель на полный путь к файлу, который должен быть загружен. Аргументы `arg0`, ..., `argN` представляют собой указатели на аргументы командной строки.

Заметим, что аргумент `arg0` должен указывать на имя загружаемого файла.

Аргумент `argv` представляет собой массив из указателей на аргументы командной строки.

Начальный элемент массива должен указывать на имя загружаемой программы, а заканчиваться массив должен элементом, содержащим указатель `NULL`.

Аргумент `envp` является массивом указателей на параметры окружающей среды, заданные в виде строк «переменная=строка».

Последний элемент этого массива должен содержать указатель `NULL`.

Поскольку вызов функции не изменяет системный контекст текущего процесса, загруженная программа унаследует от загрузившего ее процесса следующие атрибуты:

- идентификатор процесса;
- идентификатор родительского процесса;
- групповой идентификатор процесса;
- идентификатор сеанса;
- время, оставшееся до возникновения сигнала `SIGALRM`;
- текущую рабочую директорию;
- маску создания файлов;
- идентификатор пользователя;
- групповой идентификатор пользователя;
- явное игнорирование сигналов;
- таблицу открытых файлов (если для файлового дескриптора не устанавливался признак «закрывать файл при выполнении `exec()`»).

В случае успешного выполнения возврата из функций в программу, осуществившую вызов, не происходит, а управление передается загруженной программе. В случае неудачного выполнения в программу, инициировавшую вызов, возвращается отрицательное значение.

Замечание для функций `execlp()` и `execvp()`: если в первом параметре — имени файла — отсутствует символ слэш, то соответствующий файл будет искаться в директориях, список которых задается переменной среды `PATH`.

Поскольку системный контекст процесса при вызове `exec()` остаётся практически неизменным, большинство атрибутов процесса, доступных пользователю через системные вызовы (`PID`, `UID`, `GID`, `PPID` и другие, смысл которых станет понятен по мере углубления ваших знаний на дальнейших занятиях), после запуска новой программы также не изменяется.

Важно понимать разницу между системными вызовами `fork()` и `exec()`. Системный вызов `fork()` создаёт новый процесс, у которого пользовательский контекст совпадает с пользовательским контекстом процесса-родителя. Системный вызов `exec()` изменяет пользовательский контекст текущего процесса, не создавая новый процесс.

Пример 2: 03-2.c

Задачи на семинар

Задача 1 (5 баллов):

Модифицируйте программу, вычисляющую квадратный корень по алгоритму Герона, из семинара 2 так, чтобы число, из которого вычисляется корень, вводилось не с клавиатуры, а задавалось как слово в командной строке.

Предполагается, что передаваемый аргумент всегда является числом, однако возможен запуск программы без аргументов, который нужно обработать отдельно.

Задача 2 (5 баллов):

Модифицируйте программу, вычисляющую квадратный корень по алгоритму Герона, из семинара 2 так, чтобы число, из которого вычисляется корень, вводилось не с клавиатуры, а задавалось как значение некоторой переменной среды.

Предполагается, что значение переменной всегда является числом, но сама переменная может отсутствовать. Этот случай также нужно обработать отдельно.

Использование функции `getenv()` запрещено, по сути, надо реализовать ее функционал самостоятельно, используя полученные на семинаре знания.

Задача 3 (5 баллов):

Модифицируйте программу, созданную при выполнении задачи 2 с семинара 3 так, чтобы порождённый процесс запускал на исполнение новую (любую) программу. Не забудьте обработку ошибки в случае возвращения из `exec`.