UDC 004.424

NEW PARADIGM OF PROCESSING OF COMMAND LINE ARGUMENTS

OLEG SUKHORUKOV, OKSANA GOLUBEVA, VIKTOR KURLOV
Polotsk State University, Belarus

The article describes the author's method of command line arguments processing. The main features of the method, in contrast to the existing analogues, is the versatility, natively-friendly interface that allows quick start at first use and does not require deep study of documentation.

Programming technologies are not static; both, the programming paradigm itself that defines not only a set of ideas and concepts, and the style of writing computer programs [1]. Going to a higher level of abstractions, the code, according to the author, is becoming more intuitive, readable. The development of computer technology expands the style of software writing from the code "how it will be convenient for the compiler, processor" up to the code "how it will be convenient to a person". If one considers the development of the libraries used in the creation of software – firstly, the development is not only based on the maximum speed of its methods' work, and secondly, with the creation of maximum comfort, quick start of their usage, even to the detriment of the first one.

Functions of the «getopt ()» and the «getopt_long ()», being today the de facto standard tool for processing of command line arguments in the UNIX / Linux environment, were developed in 1980s [2, p. 43]. Originally, the functions were created for the «C» language, but almost for any language today there are special libraries that implement their functioning or work on their base. For example, the "Argp" created for the line of the "C" languages [3], "the getopts" for "the bash" [4], the "optparse" for the "Python" [5] and others.

Functions of the «getopt ()» and the «getopt_long ()», as well as their derivatives, seem the author archaic today, even "greeting from the 80's." For such a trivial, far from being the main, but extremely important task, as processing of command line arguments [6], it is needed, perhaps, more than one man-hour to figure out how to use them properly. At the same time, through the author's personal experience, if you fill the «Argp» structures once, and understand how to do it, you will not have to deal with it again, using it for the second time.

The problem described, forms a clear goal: in the development of a new method of command line arguments processing, the priority is - to provide maximum comfort, natively-friendly interface, quick start at first use which does not require deep study of documentation.

Not only to the author of the article the interface of the methods described seems too complicated, outdated, or having insufficient functioning. There were written numerous articles on this topic, have been repeatedly taken and are still taken successful attempts to create a simpler interface, while extending the functioning of existing methods. A good example would be the library «shflags», which uses the same function of the «getopt ()», but in addition to parsing of options, it can also check their values and even independently calls the variables for options according to their long name [7]. Similar successful analogues of standard functions can be found in almost any language, for example, a more advanced library «argparse» for the «Python» [8], replacing an «optparse» library [5], which has already become, in some cases, standard.

Behind the apparent way out, lies the problem, not obvious at first glance. Functions of the «getopt ()» and the «getopt_long ()» were universal, as they were the only ones, and therefore one did not have to delve into the nuances and differences of numerous methods. Now it turns out that in every programming language, one way or another, eliminating the drawbacks of the original functions, there is its own implementation of the command line arguments processing, with its library and a set of methods that with each successive version is increasingly removed from the original functions of 1980s.

The author, positioning himself as a system programmer-administrator, often needs to write small support programs, at the same time quickly switching from one language to another. These can be all sorts of scripts written in a script language, as well as programs written with the line of "C" languages. Processing of the command line, considering all the possible «POSIX» agreements, may not be a trivial task [6], which, in its turn, affects complexity of the application, known to the author processing methods. Often, according to the author's experience, the code of command line processing may be longer and even more complicated than the code of the main program, for example, of a short script. At the same time constant switching between the methods of options processing, different programming languages, only aggravates the situation additionally taking away time and efforts.

It is therefore necessary to develop not only simple, but also definitely a universal method of command line arguments processing, which does not require switching between languages.

One of the easiest ways to create a universal method is if a separate application is responsible for the processing of options (see Fig. 1). In this case, the "program" transmits to the "handler", in the form of a configu-

ration file or line, the structure of options used, as well as a command line that requires processing. The "handler" parses the command line according to the transmitted structure of options, stores the received data to a temporary file and completes its work. As needed the "program" sends requests to the "handler" – the "handler" compares them with the file and sends an appropriate response to the "program".

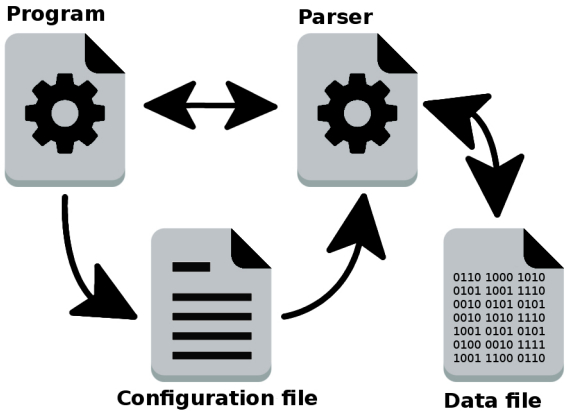

Fig. 1. General processing circuit

The circuit with the handler, in the form of a separate program, is the price that one needs to pay for versatility. Such a method cannot be called rapid, but the command line arguments processing is a procedure which usually does not require quick computations. The number of requests can be reduced to the minimum, up to the receipt of all "formalized" data in response to the transference of the command line. If the speed of information exchange between the "program" and the "handler" is important, then, the following example of the database, instead of single calls, to open session is enough, at which the "handler" will be finishing his work not after working off another request, but after a certain interval of time. Sleep interval is prolonged while the expectation of a new request. Under such a scheme of work, the "handler" does not require, at every request, to read or reset the data to a temporary file, he will do it only after a sleep interval. The data file is removed by the "handler", on request, at the end of the program life cycle.

If the scheme with the handler, in the form of a separate program, is responsible for versatility of the method, therefore the comfort and natively-friendly interface will be responsible for the structure of the configuration file. As the configuration file, one can choose any format, from «YAML» [9], «XML» [10], and «JSON» [11] in respect of their own developments. According to the author, the best solution for the formation of the configuration file, will be the use of «JavaScript" language [12].

Although the format of the "YAML" configuration file, for example, may be called "user-friendly", however, it cannot be called standard. Separation of syntactic elements is produced by a sequence of dash signs ending with a special symbol [9]. This may be unaccustomed for people not familiar with the format, which means that it can bring additional inconvenience in their work. The format «XML» has a redundant marking, behind which, eventually, the basic data are lost [10]. «JSON», unlike «XML», has a minimalistic marking that is actually unostentatious [11]. Moreover, the extended format – «JSON5», supports comments, has a simpler format allowing the existence of minor errors (for example, it is allowed to have a comma after the last element in an object or list), etc. [13].

The "JSON" is based on the «JavaScript» [11] and its extended format, in principle, is already ideally suitable for the use in the "program-handler" scheme. However, using, instead of the format «JSON5», the programming language in which the «JavaScript» was based [12], we do not limit ourselves to the format, but obtain all the available functions of the programming language, which opens up new opportunities. The "handler", in this case, will act in the role of interpreter, producing line analysis, handling and program execution, described in the configuration file.

Using the scenario programming languages in the configuration files is not a new method, but proved perfectly. A good example would be the configuration file of the domain «XEN», that is, in fact, the script in the language of «Python». This means that in the configuration file can be used any structure of this language, making it flexible and efficient. For example, you can combine configuration files of several machines into one, and to run any of them using this file with a parameter, indicating the domain with the help of which a machine should start and so on [14]. At the standard use, for example, initialization of variables, programming languages, the method of application is not different from the method used in simple configuration files, for example, in the «INI» file, distributed in «MS Windows» [15], and therefore does not complicate the code.

REFERENCES

1. The Great Encyclopedia of Oil and Gas [Bolshaya EHnciklopediya Nefti I Gaza] // Development – programming [Razvitie – programmirovanie]: site, 2016. – Mode of access: http://www.ngpedia.ru/id360477p1.html.
2. Robbins, A. (2005), Linux: programming in examples: Trans. from Eng. [Linux: programmirovanie v primerah], KUDIC-OBRAZ, Moscow, 656 c.
3. GNU Operating System // Parsing Program Options with Argp: site, 2015. – Mode of access: http://www.gnu.org/software/libc/manual/html_node/Argp.html#Argp.
4. CIT Forum // getopts - parse of command options [getopts - razbor opcij komandy]: site, 2016. – Mode of access: http://citforum.ru/operating_systems/manpages/GETOPTS.1.shtml.
5. Python // optparse — Parser for command line options: site, 2016. – Mode of access: https://docs.python.org/dev/library/optparse.html.
6. GNU Operating System // GNU Coding Standards: site, 2015. – Mode of access: http://www.gnu.org/prep/standards.
7. Habrahabr // Administering → bash script with support for long (gnu-style) options [Administrirovanie → bash skript s podderzhkoj dlinnyh (gnu-style) opcij]: site, 2016. – Mode of access: https://habrahabr.ru/post/133860.
8. Python // Argparse Tutorial: site, 2016. – Mode of access: https://docs.python.org/3/howto/argparse.html.
9. The Offical YAML Web Site // YAML 1.2: site, 2016. – Mode of access: http://yaml.org/.
10. XML.com // XML: site, 2014. – Mode of access: http://www.xml.com/.
11. JSON // Introducing JSON: site, 2016. – Mode of access: http://www.json.org/.
12. JavaScript // JavaScript: site, 2016. – Mode of access: http://www.javascript.com/.
13. JSON5 // JSON5: site, 2016. – Mode of access: http://www.json5.org/.
14. XGU // Configuration file XEN [Konfiguracionnyj fajl XEN]: : site, 2009. – Mode of access: http://xgu.ru/wiki/Конфигурационный_файл_Xen.
15. non GNU // INI formats: : site, 2016. – Mode of access: http://www.nongnu.org/chmspec/latest/INI.html.