What is a env var?

Environment Variables

Стивенс про переменные окружения

When a program is executed, it receives information about the context in which it was invoked in two ways. The first mechanism uses the argv and argc arguments to its `main` function, and is discussed in Program Arguments. The second mechanism uses *environment variables* and is discussed in this section.

The argv mechanism is typically used to pass command-line arguments specific to the particular program being invoked. The environment, on the other hand, keeps track of information that is shared by many programs, changes infrequently, and that is less frequently used.

The environment variables discussed in this section are the same environment variables that you set using assignments and the `export` command in the shell. Programs executed from the shell inherit all of the environment variables from the shell.

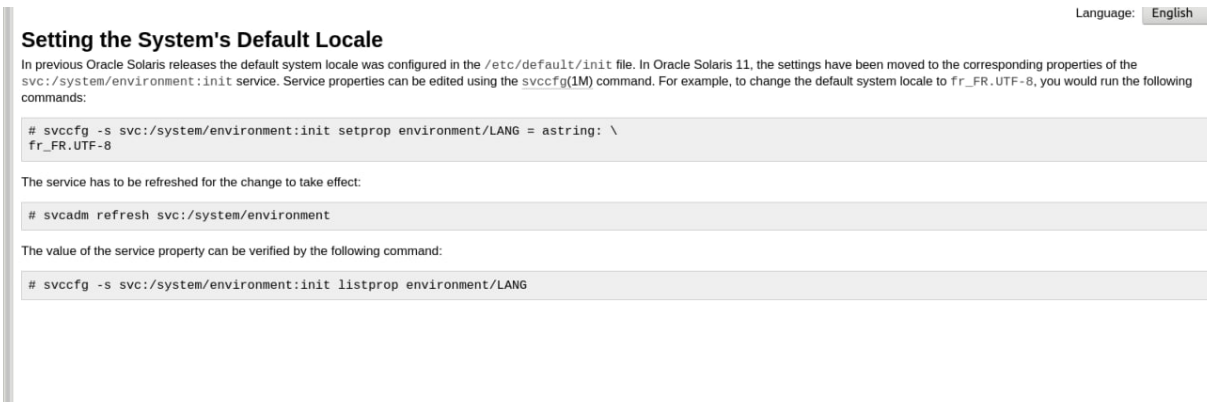
Standard environment variables are used for information about the user’s home directory, terminal type, current locale, and so on; you can define additional variables for other purposes. The set of all environment variables that have values is collectively known as the *environment*.

Names of environment variables are case-sensitive and must not contain the character '='. System-defined environment variables are invariably uppercase.

The values of environment variables can be anything that can be represented as a string. A value must not contain an embedded null character, since this is assumed to terminate the string.

> The Global environment variables of your system are stored in **/etc/default/init**

source: https://www.gnu.org/software/libc/manual/html_node/Environment-Variables.html



Setting the System's Default Locale

In previous Oracle Solaris releases the default system locale was configured in the `/etc/default/init` file. In Oracle Solaris 11, the settings have been moved to the corresponding properties of the `svc:/system/environment:init` service. Service properties can be edited using the `svccfg`(1M) command. For example, to change the default system locale to `fr_FR.UTF-8`, you would run the following commands:

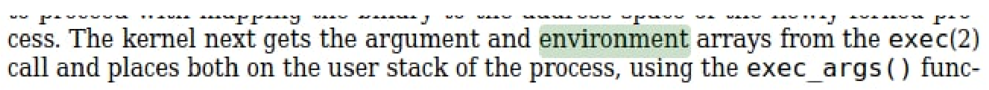
```
# svccfg -s svc:/system/environment:init setprop environment/LANG = astring: \
fr_FR.UTF-8
```

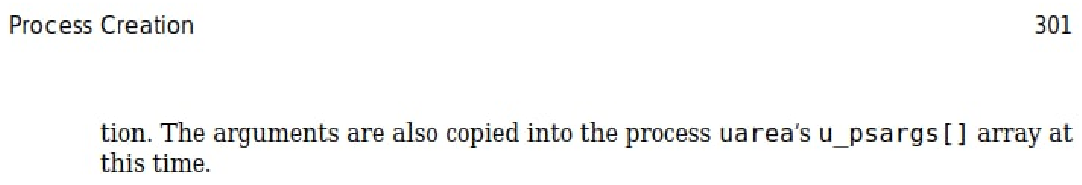
The service has to be refreshed for the change to take effect:

```
# svcadm refresh svc:/system/environment
```

The value of the service property can be verified by the following command:

```
# svccfg -s svc:/system/environment:init listprop environment/LANG
```


cess. The kernel next gets the argument and environment arrays from the exec(2) call and places both on the user stack of the process, using the exec_args() function.


Process Creation 301

tion. The arguments are also copied into the process uarea's u_psargs[] array at this time.

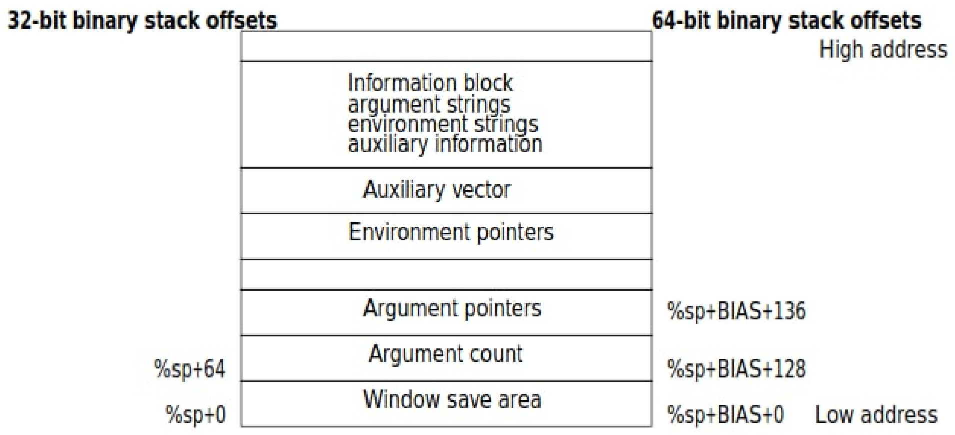


Figure 8.15 Initial Process Stack Frame

How to access?

The value of an environment variable can be accessed with the `getenv` function. This is declared in the header file <stdlib.h>.

Libraries should use `secure_getenv` instead of `getenv`, so that they do not accidentally use untrusted environment variables. Modifications of environment variables are not allowed in multi-threaded programs.
The `getenv` and `secure_getenv` functions can be safely used in multi-threaded programs.

```
Function: char * getenv (const char *name)

This function returns a string that is the value of the environment variable name.
You must not modify this string.
In some non-Unix systems not using the GNU C Library,
it might be overwritten by subsequent calls to getenv
```

```
(but not by any other library function).
If the environment variable name is not defined, the value is a null pointer.
```

```
Function: char * secure_getenv (const char *name)

This function is similar to getenv,
but it returns a null pointer if the environment is untrusted.
This happens when the program file has SUID or SGID bits set.
General-purpose libraries should always prefer this function
over getenv to avoid vulnerabilities if the library is referenced
from a SUID/SGID program.

This function is a GNU extension.
```

💡 Существует два специальных бита: SUID (Set User ID - бит смены идентификатора пользователя) и SGID (Set Group ID - бит смены идентификатора группы). Когда пользователь или процесс запускает исполняемый файл с установленным одним из этих битов, файлу временно назначаются права его (файла) владельца или группы (в зависимости от того, какой бит задан). Таким образом, пользователь может даже запускать файлы от имени суперпользователя.

Восьмеричные значения для SUID и SGID - 4000 и 2000.
Символьные: u+s и g+s.
Установка битов SUID или SGID позволит пользователям запускать исполняемые файлы от имени владельца (или группы) запускаемого файла. Например, команду chmod по умолчанию может запускать только root. Если мы установим SUID на исполняемый файл /bin/chmod, то обычный пользователь сможет использовать эту команду без использования sudo, так, что она будет выполнятся от имени пользователя root. В некоторых случаях очень удобное решение. Кстати по такому принципу работает команда passwd, с помощью которой пользователь может изменить свой пароль.

```
Function: int putenv (char *string)

The putenv function adds or removes definitions from the environment.
If the string is of the form ‘name=value’,
the definition is added to the environment.
Otherwise, the string is interpreted as the name of
an environment variable, and any definition for this
variable in the environment is removed.

If the function is successful it returns 0.
Otherwise the return value is nonzero and errno is set to indicate the error.

The difference to the setenv function is that the
exact string given as the parameter string is put
into the environment. If the user should change the
string after the putenv call this will reflect automatically
in the environment. This also requires that string not
be an automatic variable whose scope is left before
the variable is removed from the environment.
The same applies of course to dynamically allocated
variables which are freed later.

This function is part of the extended Unix interface.
You should define _XOPEN_SOURCE before including any header.
```

```
Function: int setenv (const char *name, const char *value, int replace)

The setenv function can be used to add a new definition to the environment.
The entry with the name name is replaced by the value ‘name=value’.
Please note that this is also true if value is the empty string.
To do this a new string is created and the strings name and value are copied.
A null pointer for the value parameter is illegal.
If the environment already contains an entry with
key name the replace parameter controls the action.
If replace is zero, nothing happens.
Otherwise the old entry is replaced by the new one.

Please note that you cannot remove an entry completely using this function.

If the function is successful it returns 0. Otherwise the
environment is unchanged and the return value is -1 and errno is set.

This function was originally part of the BSD library but is
now part of the Unix standard.
```

```
Function: int unsetenv (const char *name)

Using this function one can remove an entry completely from the environment.
If the environment contains an entry with the key name this whole entry is removed.
A call to this function is equivalent to a call to putenv when the
value part of the string is empty.

The function returns -1 if name is a null pointer, points to an
empty string, or points to a string containing a = character.
It returns 0 if the call succeeded.
```

```
This function was originally part of the BSD library but is now
part of the Unix standard. The BSD version had no return value, though.
```

```
Function: int clearenv (void)

The clearenv function removes all entries from the environment.
Using putenv and setenv new entries can be added again later.

If the function is successful it returns 0. Otherwise the return value is nonzero.
```

```
Variable: char ** environ
The environment is represented as an array of strings.
Each string is of the format ‘name=value’.
The order in which strings appear in the environment
is not significant, but the same name must not appear more than once.
The last element of the array is a null pointer.

This variable is declared in the header file unistd.h.

If you just want to get the value of an environment variable, use getenv.
```

source: https://www.gnu.org/software/libc/manual/html_node/Environment-Access.html

Standard Environment Variables

These environment variables have standard meanings. This doesn’t mean that they are always present in the environment; but if these variables *are* present, they have these meanings. You shouldn’t try to use these environment variable names for some other purpose.\

- `HOME`
  This is a string representing the user’s *home directory*, or initial default working directory.
  The user can set `HOME` to any value. If you need to make sure to obtain the proper home directory for a particular user, you should not use `HOME`; instead, look up the user’s name in the user database (see User Database).
  For most purposes, it is better to use `HOME`, precisely because this lets the user specify the value.

- `LOGNAME`
  This is the name that the user used to log in. Since the value in the

environment can be tweaked arbitrarily, this is not a reliable way to identify the user who is running a program; a function like `getlogin` (see Who Logged In) is better for that purpose.
For most purposes, it is better to use `LOGNAME`, precisely because this lets the user specify the value.

- `PATH`
  A *path* is a sequence of directory names which is used for searching for a file. The variable `PATH` holds a path used for searching for programs to be run.
  The `execlp` and `execvp` functions (see Executing a File) use this environment variable, as do many shells and other utilities which are implemented in terms of those functions.
  The syntax of a path is a sequence of directory names separated by colons. An empty string instead of a directory name stands for the current directory (see Working Directory).
  A typical value for this environment variable might be a string like:

  ```
  :/bin:/etc:/usr/bin:/usr/new/X11:/usr/new:/usr/local/bin
  ```

  This means that if the user tries to execute a program named `foo`, the system will look for files named foo, /bin/foo, /etc/foo, and so on. The first of these files that exists is the one that is executed.

- `TERM`
  This specifies the kind of terminal that is receiving program output. Some programs can make use of this information to take advantage of special escape sequences or terminal modes supported by particular kinds of terminals. Many programs which use the termcap library (see Find in The Termcap Library Manual) use the `TERM` environment variable, for example.

- `TZ`
  This specifies the time zone. See TZ Variable, for information about the format of this string and how it is used.

- `LANG`
  This specifies the default locale to use for attribute categories where neither `LC_ALL` nor the specific environment variable for that category is set. See Locales, for more information about locales.

- `LC_ALL`
  If this environment variable is set it overrides the selection for all the locales done using the other `LC_*` environment variables. The value of the other `LC_*` environment variables is simply ignored in this case

- `LC_COLLATE`
  This specifies what locale to use for string sorting.

- `LC_CTYPE`
  This specifies what locale to use for character sets and character classification.

- `LC_MESSAGES`
  This specifies what locale to use for printing messages and to parse responses.

- `LC_MONETARY`
  This specifies what locale to use for formatting monetary values.

- `LC_NUMERIC`
  This specifies what locale to use for formatting numbers.

- `LC_TIME`
  This specifies what locale to use for formatting date/time values.

- `NLSPATH`
  This specifies the directories in which the `catopen` function looks for

message translation catalogs.

- `_POSIX_OPTION_ORDER`
  If this environment variable is defined, it suppresses the usual reordering of command line arguments by getopt and argp_parse.