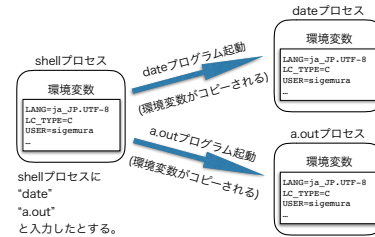


オペレーティングシステムの機能を使ってみよう 第 8 章 環境変数

オペレーティングシステムの機能を使ってみよう

1 / 26

環境変数



- シェルが管理する変数
- シェルからプログラムにコピーされる。
- プログラムは実行時に環境変数の値を調べることができる。
- 同じプログラムで複数の言語に対応すること等ができる。

オペレーティングシステムの機能を使ってみよう

2 / 26

環境変数と使用例 (1)

macOS や UNIX でよく使用される環境変数

SHELL=/bin/bash	# 使用中のシェル
TERM=xterm-256color	# 使用中のターミナルエミュレータ
USER=sigemura	# 現在のユーザ
PATH=/usr/bin:/bin:/usr...	# シェルがコマンドを探すディレクトリー一覧
PWD=/Users/sigemura	# カレントディレクトリのパス
HOME=/Users/sigemura	# ユーザのホームディレクトリ
LANG=ja_JP.UTF-8	# ユーザが使用したい言語 (ja_JP.UTF-8(日本語))
LC_TIME=C	# ユーザが日時の表示に使用したい言語(C 言語標準)
TZ=Japan	# どの地域の時刻を使用するか (日本)
CLICOLOR=1	# ls コマンド等がカラー出力する (yes)

- 本当はもっとたくさんの環境変数がある。
- ここでは「名前=値」形式で一覧を表示している。
- 次頁は LC_TIME 環境変数と TZ 環境変数を変更して実行した例

オペレーティングシステムの機能を使ってみよう

3 / 26

環境変数と使用例 (2)

```
$ printenv LC_TIME          # 環境変数 LC_TIME の値を確認する
C                            # C 言語標準 (米国英語表記) を使用する
$ date
Tue Jul 5 08:33:53 JST 2016  # 英語表記, 日本時間の現在時刻
$ ls -l Makefile
-rw-r--r--  1 sigemura  staff  355 Jun 26 23:02 Makefile
$ LC_TIME=ja_JP.UTF-8      # LC_TIME に日本語表記を表す値をセットして試す
$ date
2016 年 7 月 5 日 火曜日 08 時 34 分 13 秒 JST
$ ls -l Makefile
-rw-r--r--  1 sigemura  staff  355  6 26 23:02 Makefile
$ export TZ=Cuba           # TZ 環境変数を作ってキューバ時間を表す値をセット
$ date
2016 年 7 月 4 日 月曜日 19 時 34 分 29 秒 CDT
$ ls -l Makefile
-rw-r--r--  1 sigemura  staff  355  6 26 10:02 Makefile
$
```

- LC_TIME 環境変数は日時の表示形式を決める。
- TZ 環境変数はどの地域の時刻を表示するか決める。

オペレーティングシステムの機能を使ってみよう

4 / 26

環境変数を誰が決めるか

- (1) システム管理者
システム管理者はユーザがログインした時の初期状態を決める。
UNIX や macOS では管理者が作成したスクリプトが初期化を行う。
管理者は全ユーザに共通の初期化処理をここに書いておく。
- (2) ユーザの設定ファイル
ユーザは自分のホームディレクトリのファイルに初期化手順を書く。
初期化スクリプト (.bash_profile) の例を示す。

```
PATH="/usr/local/bin:$PATH:$HOME/bin:."
export LC_TIME=C
export CLICOLOR=1
```

- (3) ユーザによるコマンド操作
シェルのコマンド操作で環境変数进行操作することができる。
影響範囲は操作したウィンドのシェルのみである。
次のログイン時には操作結果の影響は残らない。

オペレーティングシステムの機能を使ってみよう

5 / 26

環境変数の操作 (1)

環境変数を表示するコマンド (printenv)

書式 name は環境変数の名前である。

```
printenv [name]
```

解説 name を省略した場合は、全ての環境変数の名前と値を表示する。name を書いた場合は該当のする環境変数の値だけ表示する。該当する環境変数がない場合は何も表示しない。

実行例 macOS 上での printenv コマンドの実行例を示す。環境変数の名前を省略して実行した場合は、全ての環境変数について「名前=値」形式で表示される。

```
$ printenv
SHELL=/bin/bash          <--- 「名前=値」形式で表示
TERM=xterm-256color
USER=sigemura
...
$ printenv SHELL          <--- SHELL 環境変数を表示する
/bin/bash                (「値」だけ表示される)
$ printenv NEVER          <--- 何も表示されない
$
```

オペレーティングシステムの機能を使ってみよう

6 / 26

環境変数の操作 (2)

環境変数を新規作成する手順 (その1) - sh の場合 -

書式 次の2ステップで操作を行う。

```
name=value
export name
```

解説 1行で、一旦、シェル変数を作る。
2行でシェル変数を環境変数に変更する。

実行例 1行は MYNAME 環境変数が存在するか確認している。
(MYNAME 環境変数は存在しないので何も表示されない。)
2, 3行で値が sigemura の MYNAME 環境変数を作った。
4行で MYNAME 環境変数を確認する。
(値が sigemura になっていることが分かる。)

```
1 $ printenv MYNAME          <--- MYNAME は存在しない
2 $ MYNAME=sigemura         <--- シェル変数 MYNAME を作る
3 $ export MYNAME            <--- MYNAME を環境変数に変更する
4 $ printenv MYNAME
sigemura                    <--- 環境変数 MYNAME の値
5 $
```

オペレーティングシステムの機能を使ってみよ

7 / 26

環境変数の操作 (3)

環境変数を新規作成する手順 (その2) - bash の場合 -

書式 次の1ステップで環境変数を作ることができる。

```
export name=value
```

解説 一旦、シェル変数を作ることなく環境変数を作ることができる。

実行例 次のように動作確認ができる。

```
$ printenv MYNAME          <--- MYNAME 環境変数は存在しない
$ export MYNAME=sigemura
$ printenv MYNAME          <--- MYNAME 環境変数ができていた
sigemura
$
```

オペレーティングシステムの機能を使ってみよ

8 / 26

環境変数の操作 (4)

環境変数の値を変更する手順

書式 name は環境変数の名前, value は新しい値である。

```
name=value
```

解説 「環境変数の変更」と「シェル変数の作成」は書式だけでは区別が付かない、変数名を間違った場合、間違った名前で新しいシェル変数が作成されエラーにならないので注意が必要である。

実行例 MYNAME 環境変数が既に存在している場合の実行例を示す。

```
$ printenv MYNAME          <--- 値を表示する
sigemura
$ MYNAME=yosinaga         <--- 値を変更する
$ printenv MYNAME
yosinaga                  <--- 変更されている
$
```

オペレーティングシステムの機能を使ってみよ

9 / 26

環境変数の操作 (5)

環境変数の値を参照する手順 (1)

書式 name は環境変数の名前である。

```
$name
```

解説 \$name は変数の値に置き換えられる。

実行例 1 PATH 環境変数の値にディレクトリを追加する例。

```
$ printenv PATH            # PATH の初期値を確認
/bin:/usr/bin
$ PATH=$PATH:.             # カレントディレクトリを追加
$ printenv PATH
/bin:/usr/bin:.
$ PATH=$PATH:$HOME/bin    # ホームの bin を追加
$ printenv PATH
/bin:/usr/bin:../User/sigemura/bin
$
```

オペレーティングシステムの機能を使ってみよ

10 / 26

環境変数の操作 (6)

環境変数の値を参照する手順 (2)

実行例 2 環境変数 i の値をインクリメントする例。

```
$ export i=1               # 環境変数 i を作る
$ printenv i
1
$ i=`expr $i + 1`          # クォートはバッククォート
$ echo $i
2
$
```

- expr は式の計算結果を表示するコマンド。
- バッククォートの内部は実行結果と置き換わる。
- printenv i の代わりに echo \$i でも値を表示できる。

オペレーティングシステムの機能を使ってみよ

11 / 26

環境変数の操作 (7)

環境変数を削除する手順

書式 name は変数の名前である。

```
unset name
```

解説 存在しない変数を unset してもエラーにならない。
変数名を間違ってもエラーにならないので注意が必要である。

実行例 MYNAME 環境変数が既に存在している場合の実行例を示す。

```
$ printenv MYNAME
yosinaga
$ unset MYNAME
$ printenv MYNAME
# MYNAME は存在しない
$
```

オペレーティングシステムの機能を使ってみよ

12 / 26

環境変数の操作 (8)

env コマンドを用いて環境変数を一時的に変更する手順

書式 変数=値を代入が続いた後にコマンドが続く。

```
env name1=value1 name2=value2 ... command
```

解説 最初の代入形式を環境変数の変更（作成）指示とみなす。
代入形式ではないものを以降の実行すべきコマンドとみなす。

実行例 ロケールとタイムゾーンを変更して date を実行する。
LC_TIME 環境変数は日時表示用のロケールを格納する。
TZ 変数はタイムゾーンを格納する。

```
$ date
Sun Jul 3 08:35:42 JST 2016          <--- 普通は日本時間, 英語表記
$ env LC_TIME=ja_JP.UTF-8 TZ=Cuba date
2016年 7月 2日 土曜日 19時36分01秒 CDT <--- キューバ時間, 日本語表記
$ date
Sun Jul 3 08:36:05 JST 2016          <--- 後のコマンドに影響はない
$
```

オペレーティングシステムの機能を使ってみよう

13 / 26

ロケール (ユーザの言語や地域を定義する)

LANG 環境変数や LC_TIME 環境変数にセットする値をロケール名と呼ぶ。
ロケール名は次の組み合わせで表現される。
「言語コード」、「国名コード」、「エンコーディング」

- **言語コード**は ISO639 で定義された 2 文字コードである。
(日本語は"ja")
- **国名コード**は ISO3166 で定義された 2 文字コードである。
(日本は"JP")
- **エンコーディング**は、使用する文字符号化方式を示す。
(macOS では UTF-8 方式が使用される.)。
- 使用可能なロケールの一覧は locale -a コマンドで表示できる。

macOS で日本語を使用する場合のロケール名は次の通り。
ja_JP.UTF-8 (日本語_日本.UTF-8)

オペレーティングシステムの機能を使ってみよう

14 / 26

タイムゾーン (時差が同じ地域)

どの地域時間で時刻を表示するかを環境変数で制御できる。

- 日本時間は協定世界時 (UTC) と時差がマイナス 9 時間
- TZ 環境変数にタイムゾーンを表す値をセットする。
- OS の内部の時刻は協定世界時 (UTC)
- 時刻を表示する時に TZ を参照して現地時間に変換する。
- 日本時間は TZ=JST-9 となる。
 - /usr/share/zoneinfo/ディレクトリのファイル名でも指定できる。
 - Cuba ファイルが存在するので TZ=Cuba と指定できる。
 - Japan ファイルも存在するので TZ=Japan も指定できる。
 - Asia/Tokyo ファイルが存在するので TZ=Asia/Tokyo も可。

TZ 環境変数が定義されていない時は、OS のインストール時に選択した標準のタイムゾーンが用いられる。

オペレーティングシステムの機能を使ってみよう

15 / 26

課題 No.7

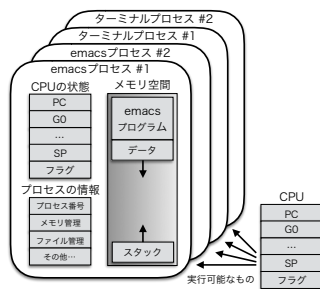
1. ここまでの実行例を試してみなさい。
2. 囲み記事を参考に、LC_TIME 環境変数や TZ 環境変数を色々試してみる。例えば、「モスクワ時間、ロシア語表記」で現在時刻を表示するにはどうしたらよいか？

オペレーティングシステムの機能を使ってみよう

16 / 26

環境変数の仕組み (0)

参考：プロセスの構造 (6章で紹介したもの)

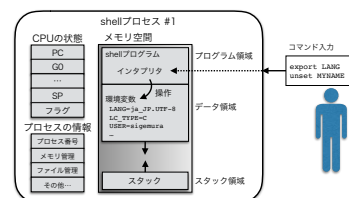


オペレーティングシステムの機能を使ってみよう

17 / 26

環境変数の仕組み (1)

シェルによる管理



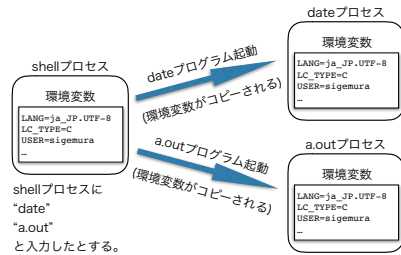
- 環境変数はシェルプロセスのメモリ空間に記憶されている。
- コマンドが入力されるとシェルのインタプリタが意味を解釈する。
- 環境変数进行操作するコマンドならメモリ空間进行操作する。
- 環境変数进行操作するコマンドは**内部コマンド**

オペレーティングシステムの機能を使ってみよう

18 / 26

環境変数の仕組み (2)

プロセスへのコピー



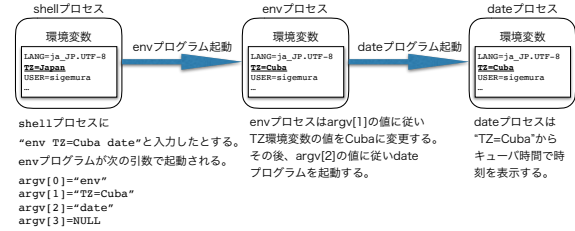
- シェルは子プロセスとして**外部コマンド**を起動する。
- 外部コマンドの起動時に子プロセスに環境変数をコピーする。
- 子プロセスはコピーされた環境変数を参照・変更・削除できる。

オペレーティングシステムの機能を使ってみよ

19 / 26

環境変数の仕組み (3)

変更した上でのコピー



- 他のプログラムを起動する時に環境変数をコピーする。
- `env` コマンドは他のプログラムを起動するプログラムの例。

1. `env` コマンドは自身の環境変数を変更する。
2. `env` コマンドは指定されたプログラムを起動する。
3. `env` コマンドは、この際、環境変数をコピーする。

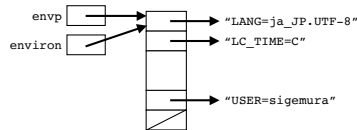
オペレーティングシステムの機能を使ってみよ

20 / 26

プログラムからの環境変数アクセス (1)

読み出し (envp 仮引数, environ 変数を用いる)

データ構造 メモリ内で環境変数は次のようなデータ構造

プログラム例 全ての環境変数を `name=val` 形式で印刷する。

```

1 #include <stdio.h>
2 extern char **environ; // 外部で定義されている
3 int main(int argc, char *argv[]) { // 今回は envp は不要
4     for (int i=0; environ[i] != NULL; i++) { // NULL が見つかるまで
5         printf("%s\n", environ[i]); // 環境変数を印刷
6     }
7     return 0; // 必ず正常終了
8 }
```

オペレーティングシステムの機能を使ってみよ

21 / 26

プログラムからの環境変数アクセス (2)

読み出し (getenv 関数を用いる)

書式 `getenv` 関数に変数名を与えると値が返る。

```

#include <stdlib.h>
char *getenv(char *name);
```

プログラム例 LANG 環境変数の値を表示する。

```

1 #include <stdio.h>
2 #include <stdlib.h> // getenv() のために必要
3 int main(int argc, char* argv[]) {
4     char *val = getenv("LANG"); // LANG 環境変数の値を調べる
5     if (val != NULL) { // 見つかったら
6         printf("LANG=%s\n", val); // 値を表示
7     } else { // 見つからない時は
8         printf("LANG does not exist.\n"); // エラーメッセージを表示
9     }
10    return 0; // 正常終了
11 }
12 /* 実行例
13 $ ./a.out
14 LANG=ja_JP.UTF-8
15 */
```

オペレーティングシステムの機能を使ってみよ

22 / 26

プログラムからの環境変数アクセス (3)

作成と値の変更 (setenv 関数)

書式 変数名 (name), 値 (val), フラグ (overwrite) を与える。

```

#include <stdlib.h>
int setenv(char *name, char *val, int overwrite);
```

解説 `overwrite=0` で上書き禁止になる。
 返り値は、正常時 0, エラー時 -1 である。
 上書き禁止時、既に変数が存在するとエラーになる。

使用例 MYNAME 環境変数の値を sigemura にする。

```
setenv("MYNAME", "sigemura", 1);
```

この例は上書き許可の場合。

オペレーティングシステムの機能を使ってみよ

23 / 26

プログラムからの環境変数アクセス (4)

作成と値の変更 (putenv 関数)

書式 `name=val` 形式の文字列 (string) を与える。

```

#include <stdlib.h>
int putenv(char *string);
```

解説 `name=val` 形式以外の文字列を与えるとエラーになる。
 返り値は正常時 0, エラー時 -1 である。
`putenv` 関数は常に上書き許可になる。

使用例 MYNAME 環境変数の値を sigemura にする。

```
putenv("MYNAME=sigemura");
```

次の `setenv` と同じ。

```
setenv("MYNAME", "sigemura", 1);
```

オペレーティングシステムの機能を使ってみよ

24 / 26

プログラムからの環境変数アクセス (5)

削除 (`unsetenv` 関数)

書式 削除する変数の名前 (`name`) を与える.

```
#include <stdlib.h>
int unsetenv(char *name);
```

解説 名前 (`name`) を指定して環境変数を削除する.
名前の変数が無いなどのエラー時-1 が返る.
正常時は 0 が返る.

使用例 MYNAME 環境変数を削除する.

```
unsetenv("MYNAME");
```

課題 No.8

1. 外部コマンド `printenv` の仕様を調べる
オンラインマニュアル (`man 1 printenv`) を読んだり, `printenv` を実際に実行したりして, `printenv` コマンドの仕様を調べなさい.
2. `myprintenv` プログラム
外部コマンド `printenv` と同様な働きをする `myprintenv` プログラムを作成しなさい. なるべく本物と同じ動作をするように作ること.