Working with UNIX Process

6. Resource Limit / 7. Environment of Processes

今日学ぶこと

- プロセスにはリソースの制限がある
  - 制限の見つけ方
    - ソフトリミットの引き上げ、制限
- プロセスには環境がある
  - 環境変数の参照の仕方

6章

プロセスのリソース制限

プロセスにはリソースの制限がある

- リソースが閉じられないとファイルディスクリプタの番号が増加し続ける

- 1プロセスあたりどれくらいのファイルディスクリプタを持てるのか？

- => カーネルによって1プロセスごとにリソースの制限が設定されている！

制限を見つける

```
p Process.getrlimit(:NOFILE)

=> [256, 9223372036854775807]
```

- :NOFILEシンボルを引数にする (オプション的な)
- [ファイルディスクリプタ数のソフトリミット, ~のハードリミット]
- http://rurema.clear-code.com/1.9.3/method/Process/m/getrlimit.html

ソフトリミットとハードリミット

- ソフトリミットはソフト側で規定した上限値
- 変更できる
- 実体はハードリミットで、こっちはとんでもなく大きな数値
- *sysctl(8)*
- http://www.usupi.org/sysad/240.html

ソフトリミットを引き上げる

```
Process.setrlimit(:NOFILE, 4096)

p Process.getrlimit(:NOFILE)

=> [4096, 4096]
```

- 第３引数で新しいハードリミットも指定できる

ソフトリミットを引き上げる

```
Process.setrlimit(:NOFILE, Process.getrlimit(:NOFILE)[1])
```

- ソフトリミットをハードリミットの値まで引き上げる

制限を超えたとき

```
# オープンできるファイルの最大数を3に設定
# 標準ストリームでファイルディスクリプタを3つ使うため
# すでに上限に達している状態になる
Process.setrlimit(:NOFILE, 3)
File.open('/dev/null')
Errno::EMFILE: Too many open files - /dev/null
```

- ソフトリミットを超えた場合にはErrno::EMFILE例外を発生する

その他のシステムリソースの制限の確認・変更

```
# プロセスのユーザが作成できる最大プロセス数
Process.getrlimit(:NPROC)
# プロセスが作成できるファイルサイズの最大値
Process.getrlimit(:FSIZE)
# プロセススタックの最大サイズ
Process.getrlimit(:STACK)
```

- Process.getrlimitに渡せるオプションのリストはドキュメント参照
- http://ruby-doc.org/core-1.9.3/Process.html#method-c-setrlimit

実用例

- システムリソースの制限の変更が必要になるプログラムはあんまない…。けど特別な用途のツールではとても重要になる
- *ex.* プロセスが同時に数千のネットワークコネクションを扱う必要がある
- *http*パフォーマンスツールの*httperf(1)*
    - *httperf –hog –server www –num-conn 5000*
    - => 同時に*5000*のコネクションを作成しようとする
- ソフトリミットがデフォルト値だと明らかにまずいので、*httperf(1)*は実行前に自身のソフトリミットを引き上げる

実用例 (2)

- 制限する例
- 第三者のコードの実行時に制約をつける
- プロセスは自身に許されている以上のリソースは使えなくなる

6章のシステムコール

- *getrlimit(2)*
  - *Process.getrlimit*
- *setr-limit(2)*
  - *Process.setrlimit*

7章

プロセスの環境

プロセスには環境がある

- 「環境」＝「環境変数」
- *key*と*value*の組み合わせ => プロセスで使えるデータを保持するもの
- 全ての子プロセスは親プロセスから環境変数を引き継ぐ
- 環境変数はプロセスごとに存在し、それぞれのプロセスではグローバルにアクセスできる

example

```
$ MESSAGE=‘wing it’ ruby -e “puts ENV[‘MESSAGE’]”
```

```
ENV[‘MESSAGE’] = ‘wing it’

system “echo $MESSAGE”
```

これってハッシュ？

```
puts ENV[‘EDITOR’]
=> vim
puts ENV.has_key?(‘PATH’)
=> true
puts ENV.is_a?(Hash)
=> false
```

- ハッシュオブジェクトではない (mergeがないとか) => ハッシュライクにアクセスできるAPIが用意されている

実用例

```
$ RAILS_ENV=production rails server
$ EDITOR=mate bundle open actionpack
$ QUEUE=default rake resque:work
```

- コマンドラインツールに入力を渡す方法としてよく採用される
- この方がコマンドラインから与えられたオプションを解析するよりもラクというメリットもある

7章のシステムコール

- *setenv(3), getenv(3)*
  - **環境変数を扱うC言語の関数**
- *environ(7)*

今日学ぶこと

- プロセスにはリソースの制限がある
  - 制限の見つけ方
    - ソフトリミットの引き上げ、制限
- プロセスには環境がある
  - 環境変数の参照の仕方