PB173 Perl

10 IPC

Roman Lacko xlacko1@fi.muni.cz

2023-07-13

Obsah

Medziprocesová komunikácia

Medziprocesová komunikácia

Medziprocesová komunikácia (*Inter-Process Communication*)

Sadá techník na výmenu informácií medzi procesmi.

Typicky vyžaduje súčinnosť operačného systému:

- Súborový systém
- Pamäťový systém
- Sieťový systém
- ...

IPC: Mechanizmy

Rôzne štandardy a mechanizmy v závislosti od platformy.
Zameriame sa na POSIX a Linux.

Nízkoúrovňové mechanizmy:

- Signály
- Zámky
- Rúry
- Sokety
- Mapovaná pamäť
- IPC SysV (semafory, správy, zdieľané segmenty)

IPC: Mechanizmy

Vysokoúrovňové mechanizmy:

- Remote Procedure Call (RPC)
- Platform Communication Stack
- Distributed Object Models
- ...

IPC: Perl

Perl má zabudované niektoré nízkoúrovňové mechanizmy priamo..

Ostatné si vieme vyrobiť sami pomocou `syscall`:

```
require 'syscall.ph';
require 'sys/epoll.ph';

sub EPoll::new($class, $flags) {
        my $fd = syscall(SYS_epoll_create1(), $flags)
                or die "epoll_create: $!";
        return bless, [$fd, IO::Handle->new_from_fd($fd)], $class;
}

sub EPoll::DESTROY($self) {
        $self->[1]->close;
}
```

IPC: Perl

Pomocou ‹pack› a ‹unpack› vieme simulovať štruktúry v C:

```
# IP as list of values
my $sockaddr_in = pack "i! x![i] n x![n] C4",
        AF_INET(), 80, 147, 251, 48, 1;

# IP as a string of bytes (using version notation)
my $sockaddr_in = pack "i! x![i] n x![n] a4",
        AF_INET(), 80, v147.251.48.1;
```

`fork`

Vytvorí klon procesu.

Návratový kód

- **rodič** dostane PID potomka
- **potomok** dostane 0, PID rodiča je možné zistiť volaním `getppid`

Pri chybevráti `undef` a nový proces *nevznikne*.

POSIX `fork(2)` vráti pri chybe -1.

```
exec LIST
exec PROGRAM LIST
```

Nahradí vykonávaný obraz procesu iným obrazom zo súboru.
Použitím ‹PROGRAM› je možné zmeniť ‹argv[0]› nového procesu

```
exec "/bin/ls", "-l";           # argv[] = { "/bin/ls", "-l", NULL };
exec { "/bin/ls" } "ls", "-l";  # argv[] = { "ls", "-l", NULL };
```

‹exec› volá niektoré varianty POSIX ‹exec*(3)› (podľa argumentov).

```
wait
waitpid PID, FLAGS
```

Počká na ukončenie potomka.

Návratový kód potomka sa uloží v premennej `$?`.
Pozor, je to hodnota priamo zo systémového volania!

```
use POSIX ();           # Do not pollute the namespace

waitpid $pid, POSIX::WNOHANG
    or die "waitpid $pid: $!";

die "Child was killed" if POSIX::WIFSIGNALED($?);
die "Child failed" if POSIX::WEXITSTATUS($?) != 0;
```

```
exit [STATUS]
```

Ukončí proces s uvedeným kódom alebo 0.

IPC: Signály

Jednoduché (typicky) asynchrónne notifikácie o výnimočnom stave alebo udalosti.

- Signál je identifikovaný celým číslom.
- POSIX real-time signals navyše umožňujú preniesť jedno číslo.

```
$SIG{QUIT} = \&handler;
$SIG{CHLD} = 'IGNORE';
$SIG{USR2} = 'DEFAULT';
```

IPC: Signály

```
kill SIGNAL, LIST
```

Odošle signál ‹SIGNAL› procesom, ktorých PID sú v zozname.
Vráti zoznam PID, pre ktoré volanie uspelo.

Signály uvádzajte menom (‹KILL› alebo ‹SIGKILL›), nie číslom.

```
kill TERM => @children;          # OK
kill SIGKILL => $pid1;           # OK

kill 9, @pids;                   # WRONG: Not portable
```

POSIX Real-Time Signals

`%SIG` nepozná `RTMIN+N` syntax!

Alternatívy pre spoľahlivé signály v Perli:

- Zistiť meno signálu z `$Config{sig_name}`
- `POSIX::sigaction()`

`Config` je modul s nastaveniami, s ktorými sa prekladal interpret

IPC: Rúry

Rúra umožňuje (obvykle) jednosmerný tok bajtov z jedného konca do druhého

```
pipe READHANDLE, WRITEHANDLE
```

Vytvorí rúru a jej konce umiestni do zadaných premennýhc.

```
pipe my $rd, my $wr
    or die "pipe: $!";
```

Typicky sa používa pred ‹`fork`›, pričom po tomto volaní:

- *Rodič* zatvorí jeden koniec.
- *Potomok* zatvorí druhý koniec.

Ak chce jeden z nich zavolať ‹`exec`›,
najprv nahradí rúrou niektorý štandardný deskriptor.

IPC: Rúry

```
IO::Pipe
```

Objektová reprezentácia rúry.
Po vytvorení (`IO::Pipe->new`) a `fork` sa designácia konca nastaví metódami:

```
$pipe->reader
$pipe->writer
```

Obe metódy zatvoria nevyužitý koniec rúry.
Objekt od toho momentu je možné používať ako inštanciu `IO::Handle`.

IPC: Rúry

```
my $pipe = IO::Pipe->new;

# fork() and call parent($pipe) or child($pipe) depending on the value

sub parent($pipe) {
    $pipe->reader;
    # We can read from ‹$pipe› now.
    ...
}

sub child($pipe) {
    $pipe->writer;
    # We can write into ‹$pipe› now.
    ...
}
```

IPC: Sokety

Sokety sú podobné rúram v tom, že predstavujú abstrakciu prúdu dát medzi dvoma procesmi, pričom

- komunikácia môže byť **obojsmerná**,
- procesy môžu byť na tom istom počítači (UNIX sokety) alebo na **iných počítačoch** (sieťové sokety)

IPC: Sieťové sokety

```
use Socket;

socket SOCKET, DOMAIN, TYPE, PROTOCOL
bind SOCKET, NAME
listen SOCKET, QUEUESIZE
accept NEWSOCKET, SOCKET
```

Vysokoúrovňové rozhrania:

- `IO::Socket`
- `AnyEvent`, `Coro`, `EV`, `UV`, ...

Pokročilá téma
Sieťového programovania sa trochu dotkneme na 12. seminári.

IPC: UNIX sokety

```
use Socket qw{AF_UNIX};
socketpair SOCKET1, SOCKET2, DOMAIN, TYPE, PROTOCOL
```

Vytvorí anonymný UNIX soket.
Typicky sa používa na obojsmernú komunikáciu s potomkom.

Pomenované UNIX sokety je možné vytvoriť rovnakým mechanizmom ako sieťové, pričom `DOMAIN` sa nastaví na `AF_UNIX`.

IPC: Sokety: Objektové rozhranie

```
my $socket = IO::Socket->new(
        Domain => Socket::AF_INET,
        Type => Socket::SOCK_STREAM,
        PeerAddr => 'google.com',
        PeerPort => 'http(80)',
);
```

Konštruktor ‹IO::Socket›, ktorý kombinuje ‹socket›, ‹bind›, ‹connect›.
Pre všetky možnosti viď ‹perldoc IO::Socket›.

```
my ($s1, $s2) = IO::Socket->socketpair(AF_UNIX, SOCK_STREAM, 0);
```

Nadstavba nad ‹CORE::socketpair›.

IPC: Backticks Operator

```
`COMMAND…`
qx{COMMAND…}
```

Kombinuje `fork`, `exec` a `waitpid` podobne ako `system`.
Navyše však použije rúru a vráti to, čo program vypísal na svoj výstup. Návratový kód programu je dostupný v `$?`.

V skalárnom kontexte vráti celý výstup, v zoznamovom zoznam riadkov.
Pri chybe vráti `undef` resp. prázdny zoznam.

Používajte opatrne, ak vôbec

- Ak `COMMAND…` obsahuje metaznaky shellu, spustí sa shell.
  Nemusí byť na každej platforme rovnaký!
- Premenné v `COMMAND…` môžu podliehať interpolácii (okrem `qx''`).
  Aký je rozdiel medzi `say qx"echo $$"` a `say qx'echo $$'`?

Moduly IPC

`IPC::Open2`, `IPC::Open3`

Spustí nový proces a pripojí vstup, výstup či chybový výstup.

```
use IPC::Open2;
my $pid = open2 my $child_out, my $child_in, @COMMAND;
```

```
use IPC::Open3;
my $pid = open3 my $child_in, my $child_out, my $child_err, @COMMAND;
```

- Namiesto rúr je možné povedať aj `"<&STDIN"`, `">&STDOUT"` a `">&STDERR"`.
- Ak je `@COMMAND` jeden prvok `-`, správa sa ako `fork` s rúrami.

‹IPC::System::Simple› (CPAN)

Nahrádza ‹system› a pridáva ‹capture›. Automaticky kontroluje návratové kódy.

```
use IPC::System::Simple qw{system capture};

system(COMMAND);                # Die on error
system(COMMAND, @ARGS);         # Avoid shell

my @output = capture(@COMMAND); # Get standard output

system([0 .. 5], COMMAND);      # Allowed exit statuses
```

‹IPC::Simple› (CPAN)

Obal na jednoduchú komunikáciu s potomkom.

```
use IPC::Simple qw{spawn};

my $ssh = spawn ['ssh', "$USER\@aisa.fi.muni.cz"];
die "Cannot spawn ssh" unless $ssh->launch;

$ssh->send("ls -l");
while (my $msg = $ssh->recv) {
        die $msg if $msg->error;
        say $msg;
}
```

‹IPC::Simple::process_group()›

‹IPC::Run› (CPAN)

„Swiss Army knife of Unix I/O gizmos“

Interaktívna komunikácia s potomkom

```
use IPC::Run qw{run};

run [@COMMAND], "in.txt", "out.txt";     # Filenames
run [@COMMAND], \"input", \$output;      # Scalars
run [@COMMAND], undef, \undef;           # Throw away vs inherit
run \@A, '|', \@B, $in, $out;            # Pipes
```

... a mnoho ďalších možností.

‹IPC::Run3› (CPAN)

Osekaná verzia ‹IPC::Run›, ale s podporou pre chybový výstup.

```
use IPC::Run3;

run3 [@COMMAND], "in.txt", \$out, \$err;
```

Špeciality

Znečistený režim

Taint mode je režim interpretu zameraný na zvýšenie bezpečnosti.
V tomto režime vykonáva kontroly navyše:

- Cesty v `$ENV{PATH}` nesmú mať `o+w` právo.
- Hodnoty zo vstupov sú označené ako „špinavé“ (*tainted*).
  Nedajú sa použiť pri niektorých potenciálne nebezpečných operáciách.

Zniečistený režim sa zapína

- automaticky, ak má skript `RUID` $\neq$ `EUID` alebo `RGID` $\neq$ `EGID`,
- prepínačom `-t` (varovania) alebo `-T` (chyby).

Znečistený režim

```
#!/usr/bin/env perl -T

die "usage: $0 FILE\n" unless @ARGV == 1;

open my $fh, '>:raw', $ARGV[0]
    or die "$ARGV[0]: $!";
...
```

```
Insecure dependency in open while running with -T switch at line 5.
```

Znečistený režim

Znečistenie je možné zrušiť výberom hodnoty z regulárneho výrazu; alternatívne použitím ako kľúč v asociatívnom poli.

```
#!/usr/bin/env perl -T

die "Usage: $0 FILE\n" unless @ARGV == 1;
die "Insecure file name" unless $ARGV[0] =~ /^([a-zA-Z0-9\.]+)$/;

open my $fh, '>:raw', $1
    or die "$1: $!";
...
```