ТЕХНОСФЕРА

Общение с внешним миром

Отметьтесь на портале!

Содержание

Работа с файлами

FILEHANDLE - специальный тип

STDIN, STDOUT, STDERR - стандартные потоки

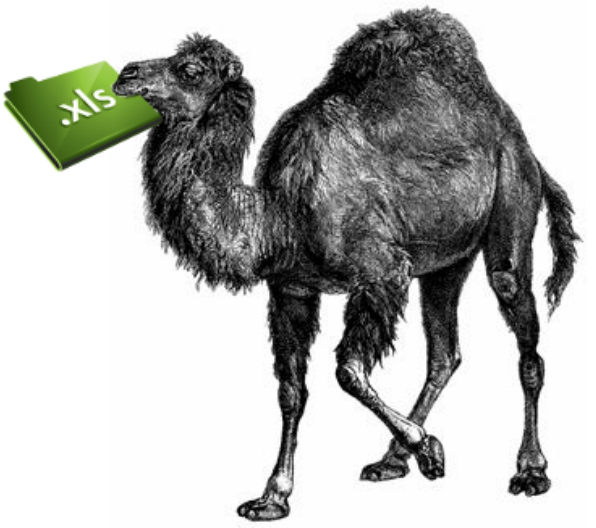
FILE_HANDLE vs my $fh

open or die

close

/dev/random + /dev/urandom - генераторы случайных/псевдослучайных чисел

/dev/null - пустое устройство



Работа с файлами

```
open( $fh, '<', 'file') vs open( $fh, '< file')
```

Режимы открытия файла

<, +<, >, +>, >>;

fopen(3) r , r+ , w , w+ , a

Указание кодировки при открытии

```
open(my $fh_cp,'<:encoding(CP-1251)','cp1251.txt');
open(my $fh_utf, '>:encoding(UTF-8)', 'utf8.txt');
while( <$fh_cp> ){
    print $fh_utf $_;
}
close($fh_utf);
close($fh_cp);
```

Работа с файлами

Чтение из файлового манипулятора

```
$input= <>

$line = <$handle>

@lines = <$handle>
```

Запись и файловый манипулятор

```
print $var;

print $fh $var;

print STDERR $var;
```

Работа с файлами

DATA - данные непосредственно в програмном модуле

```
package mypkg;

sub read_my_data {
    my @lines = <DATA>;
    return \@lines;
}

1;
__DATA__
This is data from pm file
```

Не нужно вызывать open

```
my $line = <DATA>
```

Работа с файлами

```
# для работы с двоичными данными
binmode($fh);  # open with :raw

# небуферизированная запись
syswrite($fh, $data, length($data));

# прямой вызов системного read
sysread($fh, $data, $data_size);

# чтение двоичных данных
read($fh, $data, $data_size);

# проверка на отсутствие данных
eof($fh);
```

Работа с файлами

Пример:

```
use strict;
use Digest::MD5 qw/md5/;

my $data = '';
my $data_size = 1024; # Размер записи в файле

open(my $fh, '<:raw', 'data.bin') or die $!;

while(!eof($fh)){
    read($fh, $data, $data_size) == $data_size
        or die("Неверный размер");
    print md5($data);
}

close($fh);
```

Работа с файлами

Проверка результата работы команды open

```
my $fh;
```

```
open $fh, '<', 'not_exist' || die $!;
```

```
open $fh, '<', 'not_exist' or die $!;
```

Работа с файлами

Проверка результата работы команды open

```
my $fh;
```

```
open $fh, '<', 'not_exist' || die $!;
```

```
open $fh, '<', 'not_exist' or die $!;
```

```
open($fh, '<', 'not_exist') || die $!;
```

Работа с файлами

Произвольный доступ

```
seek( $fh, $len, $type);
  # позиционирование

tell($fh);
  # текущая позиция
```

Работа с файлами

Операции проверки файлов

-r -w -x чтение, запись, исполнение

-o принадлежность файла пользователю

-e существование файла

-z файл нулевой длины

-s размер файла

-f, -d, -l, -S, -p файл, каталог, ссылка, сокет, канал

```
my $fname = 'file.txt';
my $fh;
if ( -e $fname and -f $fname and
     -r $fname and !-z $fname) {
    open($fh, '<', $fname) or die $!;
}
```

Работа с файлами

rename - переименование

unlink - удаление

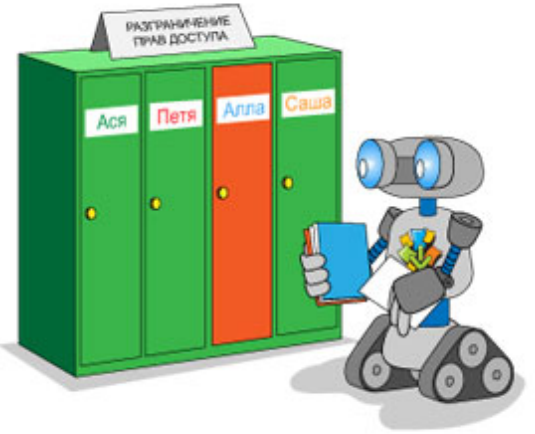
truncate - очистка

stat - информация о доступе к файлу

utime - модификация времени доступа к файлу

Работа с файлами

```
mkdir 'dir_name',    0755;
rmdir 'dir_name';
chdir 'dir_name';
```




```
use File::Path qw/make_path/;
make_path( '/full/path/to/dir',
    owner => 'user',
    group => 'group',
    mode => 0755);
```

Работа с файлами

```
opendir(my $dh, 'path_to_dir') or die $!;

my $pos;

while(my $fname = readdir $dh){
    print $fname;
    $pos = telldir $dh if $fname = 'data.bin';
}

if ($pos){
    seekdir($dh, $pos) if $pos;
    while(my $fname = readdir $dh){
        print "Second iter: $fname";
    }
}

closedir($dh);
```

Содержание

Perl io backend

:unix - использование pread/pwrite

:stdio - использование fread, fwrite, fseek/ftell

:perlio - перл буфер для быстрого доступа к данным после чтения и минимизации копирования (readline/<>)

:crlf - преобразование перевода строки

:utf8 - работа в utf-e

:encoding - перекодировка содержимого файла

:bytes - работа с однобайтовыми кодировками

:raw - binmode()

:pop - псевдослой, который позволяет убрать из цепочки верхний слой

```
% PERLIO=perlio
```

Perl io backend

```
use strict;
use PerlIO;
use Time::HiRes qw/gettimeofday/;

my $i=0;
my $start_time = gettimeofday();

while(<>){$i++}

print "Layers: "
    .join(',', PerlIO::get_layers(STDIN)).'; ';
print "lines: $i; time: "
    .(gettimeofday() - $start_time).$/;
```

Layers: unix,perlio; lines: 2932894; time: 0.410083055496216

Layers: stdio; lines: 2932894; time: 3.00101494789124

Layers: unix; lines: 2932894; time: 33.2629461288452

Perl io backend

:via - возможность подключения слоя из внешних библиотек

Например PerlIO::via::gzip

```
open( $cfh, ">:via(gzip)", 'stdout.gz' );
print $cfh @stuff;

open( $fh, "<:via(gzip)", "stuff.gz" );
while (<$fh>) {
...
}
```

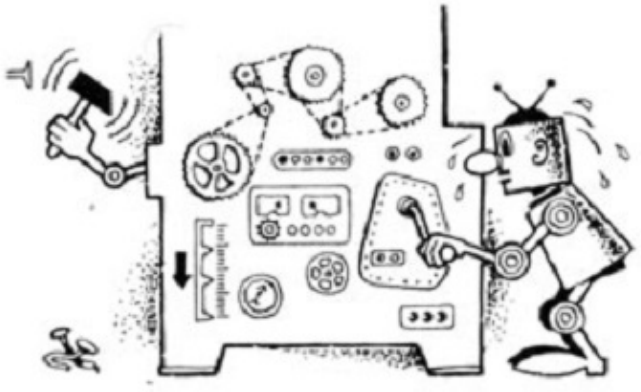
Если на CPAN нет необходимого слоя, его можно реализовать, самостоятельно определив необходимый набор функций вашего модуля.

perldoc PerlIO::via

Содержание

Взаимодействие процессов

Запуск процессов из perl программы



```
my $out = ls -l;
  # построчное чтение stdout
my @out = ls -l;
  # стандартный вывод на выходе
system('ls -l');
  # только код завершения
open(my $out, '-|', 'ls', '-l');
```

perlipc

Взаимодействие процессов

pipe - связка манипуляторов в канал

fork - порождение нового процесса (единственный способ)

exec - замена текущего процесса другим

autoflush - управление буферизацией

Взаимодействие процессов

```
use strict;
use POSIX qw(:sys_wait_h);
$|=1;

my ($r, $w);
pipe($r, $w);
if(my $pid = fork()){
    close($r);
    print $w $_ for 1..5;
    close($w);
    waitpid($pid, 0);
}
else {
    die "Cannot fork $!" unless defined $pid;
    close($w);
    while(<$r>){ print $_ }
    close($r);
    exit;
}
```

Взаимодействие процессов

Обработка сигналов

INT,CHLD,TERM,ALRM ... кроме SIGKILL, SIGSEGV, SIGABRT

Уборка зомби

$? : 16-битное число

```
my $exit_status = $? >> 8;
my $signal_num = $? & 127;
my $core_dump = $? & 128;
```

WIFEXITED - истина если процесс завершился

WEXITSTATUS - код возврата, установлен только если WIFEXITED истина

WIFSIGNALED - истина если процесс был остановлен сигналом

WTERMSIG - номер сигнала, который остановил процесс

Взаимодействие процессов

```
$SIG{CHLD} = sub {
  while( my $pid = waitpid(-1, WNOHANG)){

    last if $pid == -1;

    if( WIFEXITED($?) ){
      my $status = $? >> 8;
      print "$pid exit with status $status $/";
    }
    else {
      print "Process $pid sleep $/"
    }

  }
};
```

Взаимодействие процессов

Обработка сигналов

Игнорируем сигнал

```
$SIG{INT} = 'IGNORE';
```

Обрабатываем сигнал сами

```
$SIG{INT} = sub {...};
```

Возвращаем обработку сигнала в изначальное поведение

```
$SIG{INT} = 'DEFAULT';
```

Взаимодействие процессов

Блокировка файлов LOCK_EX LOCK_SH LOCK_UN

```
use Fcntl ':flock';
$SIG{ALRM} = sub {die "Timeout"};

alarm(10);

eval {
    flock(FH, LOCK_EX) or die "can't flock: $!";
};

alarm(0);
```

Неблокирующий вызов

```
flock(FH, LOCK_EX|LOCK_NB)
```

Взаимодействие процессов

Дополнительные модули

- IPC::Open3
- IPC::Run3
- IO::Handle

```
my($wtr, $rdr, $err);
$pid = open3($wtr, $rdr, $err, 'cmd', 'arg', ...);
```

Именованные каналы

% mkfifo /path/named.pipe

```
open( my $fifo, '<', '/path/named.pipe' );
while(<$fifo>){
    print "Got: $_";
}
close($fifo);
```

Содержание

Работа с сокетами

```
use Socket; # include <socket.h>
```

```
my $name = 'search.cpan.org';
my $addr_bin = gethostbyname($name);
my $ip = inet_ntoa($addr_bin);   # 194.106.223.155
```

```
my $ip = '207.171.7.72';
my $addr_bin = inet_aton($ip);
my $name = gethostbyaddr($addr_bin, PF_INET);
                                  # mt.perl.org
```

Работа с сокетами

Клиент

```
use strict;
use IO::Socket;
my $socket = IO::Socket::INET->new(
    PeerAddr => 'search.cpan.org',
    PeerPort => 80,
    Proto    => "tcp",
    Type     => SOCK_STREAM)
or die "Can`t connect to search.cpan.org $/";

print $socket
   "GET / HTTP/1.0\nHost: search.cpan.org\n\n";
my @answer = <$socket>;
print(join($/, @answer));
```

Работа с сокетами

Сервер

```
use strict;
use IO::Socket;
my $server = IO::Socket::INET->new(
    LocalPort => 8081,
    Type      => SOCK_STREAM,
    ReuseAddr => 1,
    Listen    => 10)
or die "Can't create server on port 8081 : $@ $/";
while(my $client = $server->accept()){
    $client->autoflush(1);
    my $message = <$client>;
    chomp( $message );
    print $client "Echo: ".$message;
    close( $client );
    last if $message eq 'END';
}
close( $server );
```

Работа с сокетами

Определение имени и порта клиента

```
use IO::Socket qw/getnameinfo/;

my $other = getpeername($client);
my ($err, $host, $service) = getnameinfo($other);
print "New connection! from $host:$service $/";
```

```
struct sockaddr_in {
    short          sin_family;
    unsigned short sin_port;
    struct in_addr sin_addr;
    char           sin_zero[8];
};
```

Работа с сокетами

Обработка нескольких соединений

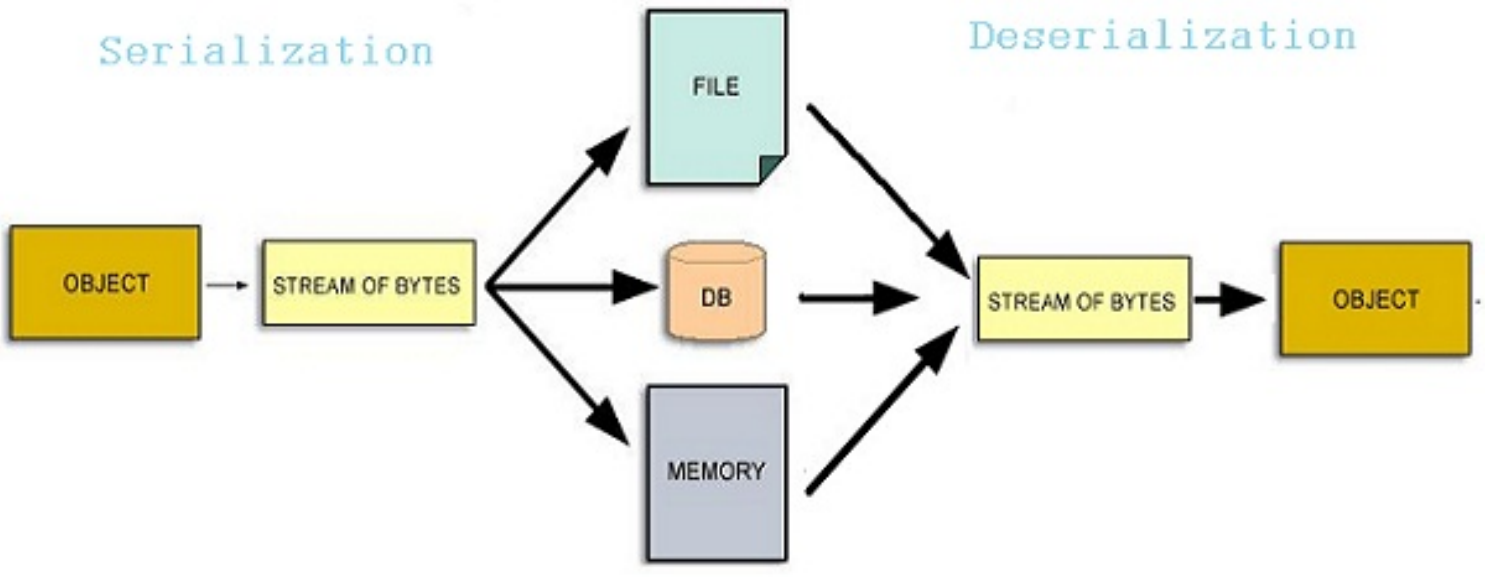
```
while(my $client = $server->accept()){
  my $child = fork();
  if($child){
    close ($client); next;
  }
  if(defined $child){
    close($server);
    my $other = getpeername($client);
    my ($err, $host, $service)=getnameinfo($other);
    print "Client $host:$service $/";
    $client->autoflush(1);
    my $message = <$client>;
    chomp( $message );
    print $client "Echo: ".$message;
    close( $client );
    exit;
  } else { die "Can't fork: $!"; }
}
```

Socket

Работа с сокетами

Содержание

Сериализация



Сериализация pack

pack - преобразование перловых типов данных в последовательность байт.

unpack - обратная операция pack

pack TEMPLATE, LIST

- **a** - строка байт, дополняемая нулями
- **A** - строка байт, дополняемая пробелами
- **b** - Битовая строка (младший бит идет первым)
- **c** - Однобайтовый символ со знаком
- **d** - Значение с плавающей запятой, двойной точности
- **f** - Значение с плавающей запятой, одинарной точности шаблона
- **h** - Строка шестнадцатиричных значений (младшие разряды идут первыми)
- **i** - Целое со знаком
- **l** - Целое со знаком типа long
- **n** - Целое 16 бит big-endian
- **v** - Целое 16 бит little-endian

Сериализация pack

```
pack "A5", "perl", "language";       # "perl "
pack "A5 A2 A3", "perl", "language";# "perl la   "

pack "H2", "31";                     # "1"
pack "B8", "00110001"                # "1"

pack "LLxLLx", 1, 2, 3, 4;
    # "\1\0\0\0 \2\0\0\0 \0 \3\0\0\0 \4\0\0\0 \0"

unpack "H*", pack "A*", "string";   # 737472696e67

unpack "(H2)*", pack "A*", "string";
    # (73,74,72,69,6e,67)
```

Сериализация pack

Специальный символ **/**

length-item/string - позволяет упаковывать строки

```
pack "Ca* ", length("Test"), "Test";# "\04Test"

pack "C/a*", "Test";             # "\04Test"
pack "L/a*", "Test";             # "\04\00\00\00Test"
pack "w/a*", "Test";             # "\04Test"
```

Сериализация JSON

```
{ "orderID": 12345,
  "shopperName": "Ваня Иванов",
  "shopperEmail": "ivanov@example.com",
  "contents": [
    {
      "productID": 34,
      "productName": "Супер товар",
      "quantity": 1
    },
    {
      "productID": 56,
      "productName": "Чудо товар",
      "quantity": 3
    }
  ],
  "orderCompleted": true
}
```

Сериализация JSON

```
use JSON::XS;
use DDP;
p JSON::XS::decode_json(
    '{"key_array":["val1", "val2", 3]}'
);
```

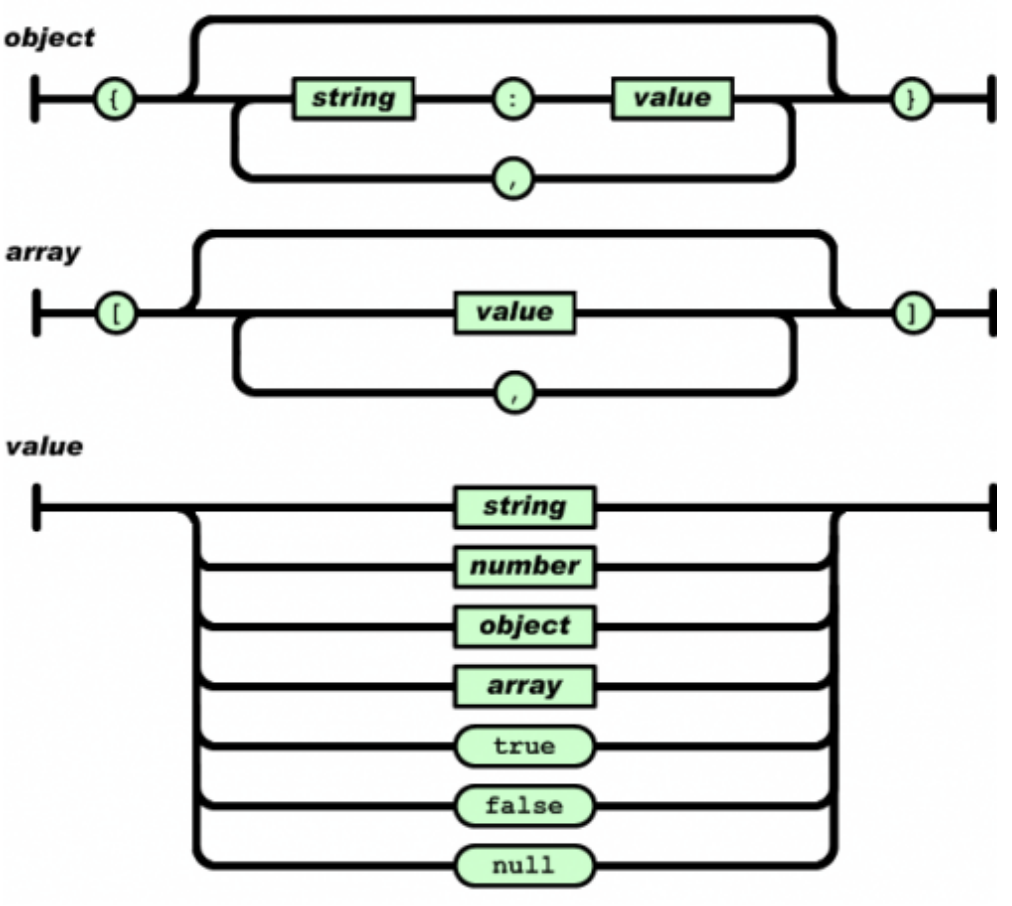
```
{"key_array":["val1","val2",3]}
\ {
    key_array   [
        [0] "val1",
        [1] "val2",
        [2] 3
    ]
}
```

Сериализация JSON

```
use strict;
use JSON::XS;
my $struct = {key1 => 3};
print "Value: ".$struct->{key1}.$/;
print JSON::XS::encode_json( $struct ).$/;
```

```
Value: 3
{"key1":"3"}
```

Сериализация JSON



Сериализация CBOR

```
use CBOR::XS;
my $cbor = CBOR::XS::encode_cbor([12,20,30]);
my $hash = CBOR::XS::decode_cbor( $cbor );
```

```
use CBOR::XS;
my $cbors = CBOR::XS::encode_cbor([12,20,30]);
$cbors   .= CBOR::XS::encode_cbor(
    ["val1","val2","val3"]
);
my @array = ();
my $cbor_obj = CBOR::XS->new();
while( length $cbors ){
  my($data, $len)=$cbor_obj->decode_prefix($cbors);
  substr $cbors, 0, $len, '';
  push @array, $data;
}
```

Сериализация MSGPACK

```
use strict;
use Data::MessagePack;
my $mp = Data::MessagePack->new();
my $packed   = $mp->pack({a => 1, b => 2, c => 3});
my $hash = $mp->unpack($packed);
```

	JSON	MessagePack
null	null	c0
Integer	10	0a
Array	[20]	91 14
String	”30”	a2 ‘3’ ‘0’
Map	{“40”:null}	81 a1 ‘4’ ‘0’ c0

Сериализация Storable

```
use Storable;
my %table = ( "key1" => "val" );
store \%table, 'file';
$hashref = retrieve('file');
```

```
use Storable qw/freeze thaw/;
my %table = ( "key1" => "val" );
my $serialized = freeze \%table;
my $hash = thaw( $serialized );
```

Сериализация XML

```
use XML::LibXML;
my $dom = XML::LibXML->load_xml(
    string => '<xml><test>1</test></xml>'
);
```

```
use XML::Parser;
my $parser = XML::Parser->new(
    Handlers => {
        Start => sub{print "New tag"},
        End   => sub{print "End tag"},
        Char  => sub{print "Data"}
    });
$parser->parse('<xml><test>1</test></xml>');
```

Сериализация

Быстродействие

```
YAML             84/s
XML::Simple     800/s
Data::Dumper   2143/s
FreezeThaw     2635/s
YAML::Syck     4307/s
JSON::Syck     4654/s
Storable       9774/s
JSON::XS      41473/s
CBOR::XS      42369/s
```

Содержание

Разбор входных параметров

Флаги

```
rm -rf
ls -l
```

Параметры

```
mkdir -m 755
perl -e ''
```

Getopt::Long

Разбор входных параметров

```
use Getopt::Long;

my $param;
GetOptions("example" => \$param);
```

Описание параметра/флага	Пример
param	--param или отсутствует
param!	--param --noparam
param=s	--param=string
param:s	--param --param=string
param=i	--param=1
param:i	--param --param=1
param=f	--param=3.14
param:f	--param --param=3.14

param|p=s

Getopt::Long

Getopt::Long + Pod::Usage

```
use Getopt::Long;
use Pod::Usage;
my $param = {};
GetOptions($param, 'help|?', 'man', 'verbose')
    or pod2usage(2);
pod2usage(1) if $param->{help};
pod2usage(-exitval => 0, -verbose => 2)
    if $param->{man};

__END__

=head1 NAME

sample - Script with Getopt::Long

=head1 SYNOPSIS
sample [options] [file ...]
Options:
-help            brief help message
-verbose         verbosity mode
```

Интерактивный режим

```
use strict;

sub is_interactive {
    return -t STDIN && -t STDOUT;
}

my $do = 1;
while( is_interactive() && $do ){
    print "Tell me anything: ";
    my $line = <>;
    print "Echo: ".$line;
    $do = 0 if $line eq "bye$/";
}

print "Goodbye$/";
```

Домашнее задание

Домашнее задание

1. lib/Local/App/GenCalc.pm — сервер который генерирует примеры для решения и отдаёт их для обработки. Одновременно может обрабатывать запрос только от одного клиента. По таймеру в 100 милисекудн генерирует новые примеры для решения и складывает их в файл. При отдаче данных клиенту выбирает из файла N запрошеных строк сверху. На каждый запрос отдельный конект.
2. lib/Local/App/Calc.pm — сервер который принимает один пример на вход и отдаёт результат. На каждого клиента создаётся отдельный процесс, в рамках одного прицесса обрабатывается много примеров пришедших на вход.
3. lib/Local/App/ProcessCalc.pm — библиотека, которая умеет ходит на сервер за пачкой новых примеров, делить эти примеры на заданное кол-во обработчиков и в каждом из них ходить на сервер для получения результата. Каждый поток складывает результаты в свой файл. После того, как все процессы отработают, мастер собирает все результаты и возвращает одной пачкой. В процессе работы каждый поток после решения любого примера должен обновлять статистику в файле статуса, он один на все потоки.

Домашнее задание

Важно!

1. Правильно обрабатывать ситуацию, когда одновременно происходит генерация новых примеров + пришёл запрос на получение пачки примеров для решения. При обновлении файла статуса необходимо следить за тем, что бы одновременно не пришли несколько потоков для обновления и не перетёрли друг друга. (flock)
2. Остановка серверов происходит по сигналу INT
3. После завершения программы все созданные файлы должны быть удалены с диска.
4. Для собственной безопастности, рекомендую предусмотреть случай переполнения диска генератором.
5. Если хоть один обработчик завершил свою работу с сигналом отличным от 0, необходимо прервать работу всех остальных и бросить исключение.

Оставьте отзыв

Спасибо за внимание!

Николай Шуляковский

Email & Agent: n.shulyakovskiy@corp.mail.ru