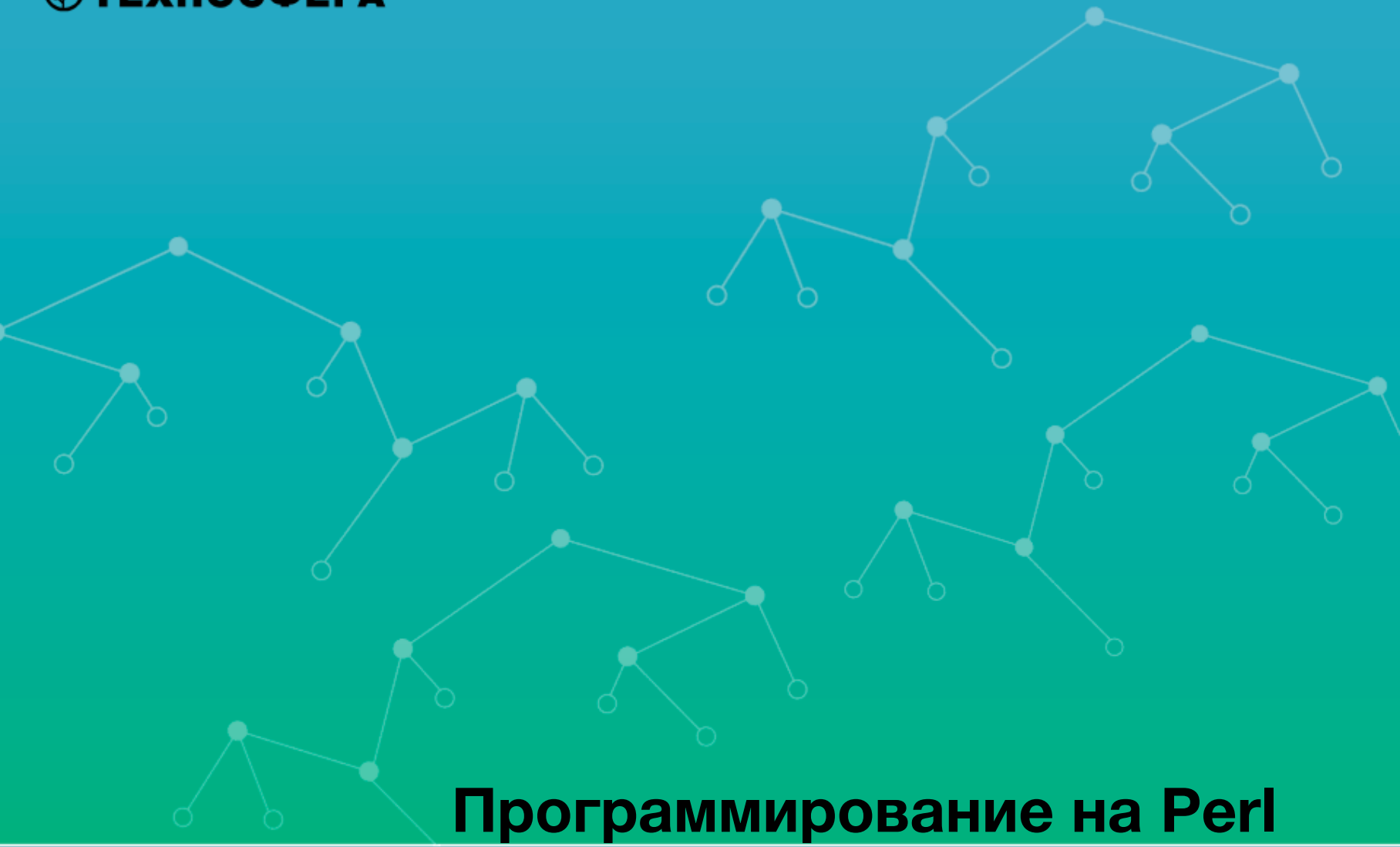
ТЕХНОСФЕРА



Программирование на Perl

ТЕХНОСФЕРА

Асинхронно-событийное программирование

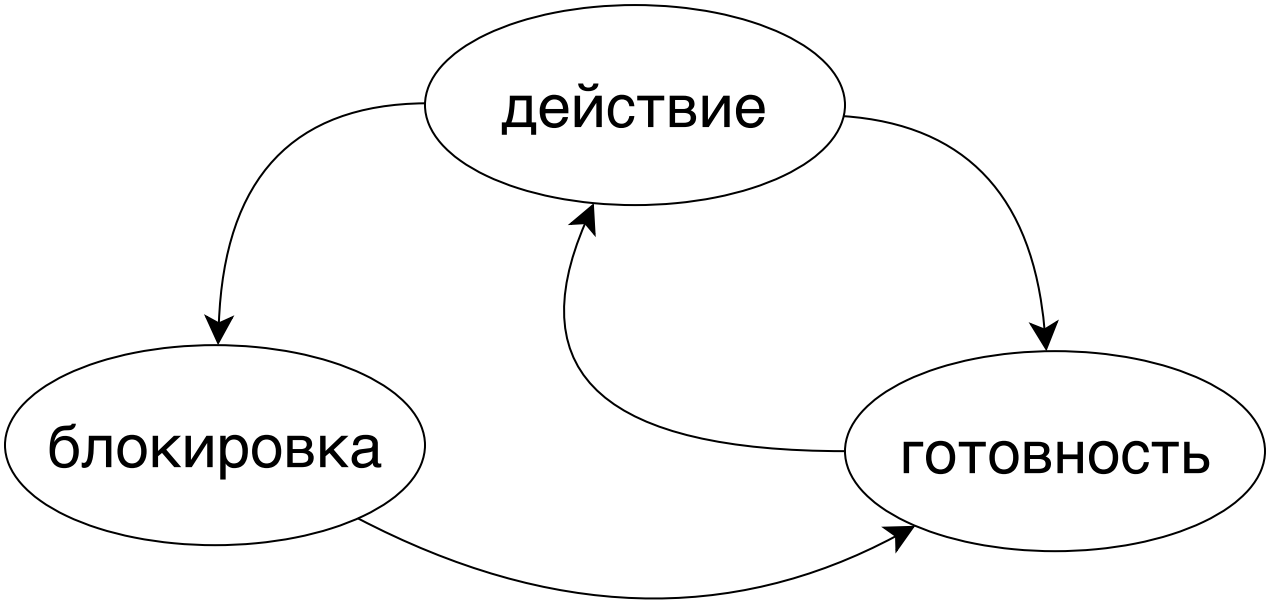
Содержание

Что такое процесс?

Что такое паралеллизм?

Что такое псевдопаралеллизм?

Состояние процесса



Переключение контекста

Context switch

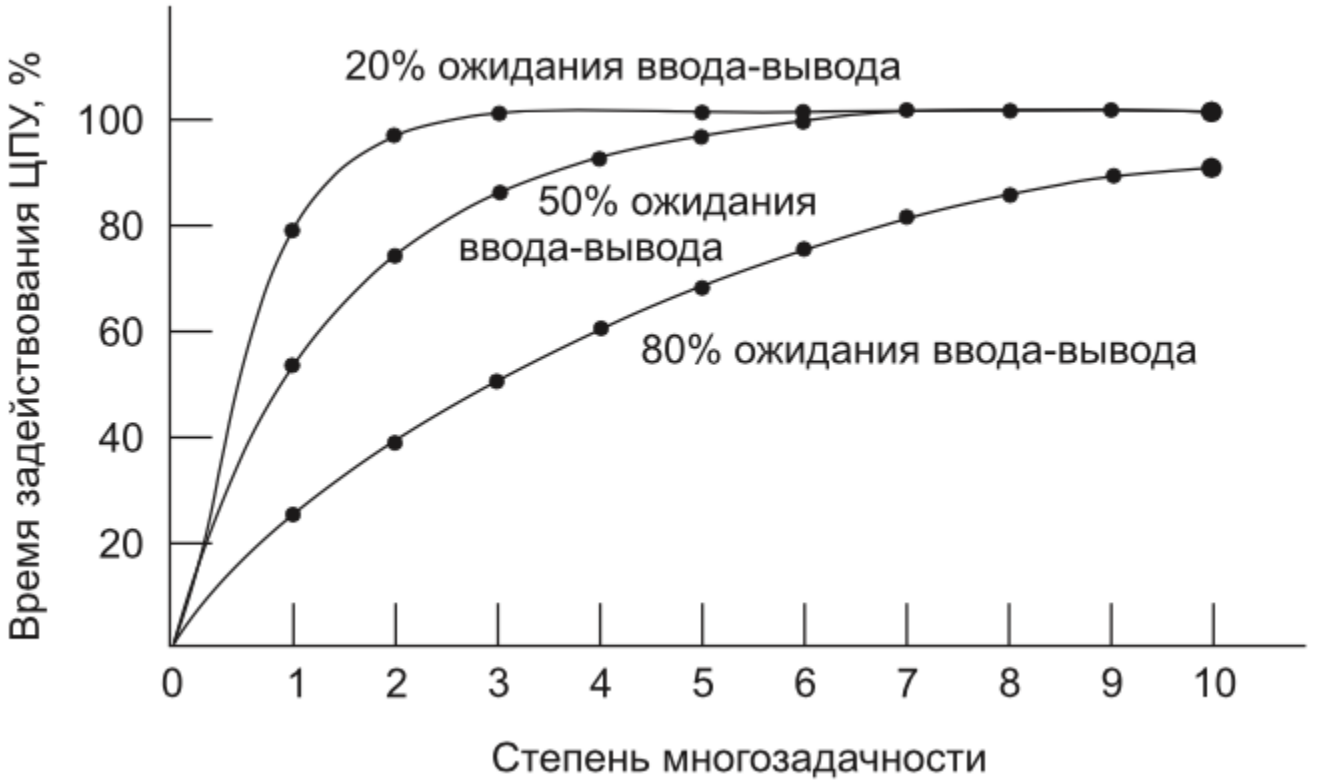
- обработать прерывание таймера
- сохранить регистры
- сохранить стек
- выбрать новый процесс
- загрузить регистры
- загрузить стек

CONFIG_HZ_100

CPU Usage

CPU = (1 − p^n) * 100%

p: часть времени ожидания ввода-вывода



Системный вызов

просьба ОС выполнить привилегированную операцию

- Прерывание
- передача управления ос
- проверка привилегий
- работа с оборудованием
- загрузка данных для программы
- возврат управления

Системный вызов ввода-вывода

- Прерывание
- передача управления ос
- проверка привилегий
- работа с оборудованием
- установка обработчика прерывания
- context switch
- ...
- context switch
- загрузка данных для программы
- возврат управления исходному процессу

Простой tcp сервер

```
while (1) {
    accept
    fork
        read
        write
        read
        write
        ...
        close
        exit
}
```

Ожидание IO: 80%

$CPU = 1 - p^n$

p = 80%

1 процесс - 20% CPU

2 процесса - 36% CPU

3 процесса - 49% CPU

10 процессов - 89% CPU

20 процессов - 99% CPU

Ожидание IO: 0.01%

$CPU = 1 - p^n$

$p = 0.01\%$

1 процесс - 0.01% CPU

10 процессов - 0.1% CPU

100 процессов - 1% CPU

1000 процессов - 10% CPU

10000 процессов - 63% CPU

Ожидание IO: 0.01%

$CPU = 1 - p^n$

p = 0.01%

1 процесс - 0.01% CPU

10 процессов - 0.1% CPU

100 процессов - 1% CPU

1000 процессов - 10% CPU

10000 процессов - 63% CPU

1Mb/process = 10Gb RAM

+ 10000 CWS

Блокирующее IO

```
read(...) -> SUCCESS

read(...) -> FATAL ERROR
```

Неблокирующее IO

```
read(...) -> SUCCESS

read(...) -> TEMPORARY ERROR

read(...) -> FATAL ERROR
```

```
while (1) {
    for my $fh (@fds) {
        my $res = read($fh, ...);
        if ($res) {
            # do work
        }
        elsif ($! ~~ FATAL_ERROR) { # pseudocode
            # close fh, remove from @fds
        }
        else {
            # wait
        }
    }
}
```

select

```
($found,$timeleft) =
    select(
        $readable, # vec
        $writable, # vec
        $errors,   # vec
        $timeout   # in fractional seconds
    )
```

```
vec($readable, fileno(STDIN),  1) = 1;
vec($readable, fileno(STDOUT), 1) = 1;
vec($readable, fileno(STDERR), 1) = 1;

say unpack "B*", $readable; # 00000111
```

IO::Select

```
use IO::Select;

$s = IO::Select->new();

$s->add(\*STDIN);
$s->add($fd);

@ready = $s->can_read($timeout);
```

O_NONBLOCK

```
use Fcntl qw(F_GETFL F_SETFL O_NONBLOCK);

$flags = fcntl($fd, F_GETFL, 0)
    or die "Can't get flags for the socket: $!\n";

$flags = fcntl($fd, F_SETFL, $flags | O_NONBLOCK)
    or die "Can't set flags for the socket: $!\n";
```

EAGAIN, EINTR, EWOULDBLOCK

```
use Errno qw(EAGAIN EINTR EWOULDBLOCK);

my $read = sysread($fd, my $buf, SOMELENGTH);

if ($read) { # read >= 0
    # work with data in buf
}
elsif (defined $read) { # read == 0
    # socket was closed
}
elsif ( $! ~~ [ EAGAIN, EINTR, EWOULDBLOCK ] ) {
    # socket not ready for reading
}
else {
    # socket was closed with error $!
}
```

Event loop

```
use IO::Select; my $s = IO::Select->new();

my $timeout = 1;
# prepare program...

while () {
    my @ready = $s->can_read($timeout);
    for (@ready) { # do reads }

    my @ready = $s->can_write($timeout);
    for (@ready) { # do writes }
}
```

Замыкание

```
{
    my $var = rand();

    my $sub = sub {
        print $var;
    }
}
```

Замыкание

```
{
    my $var = rand();

    my $sub = sub {
        print $var;
    }
}
```

```
{
    my $var = rand(); # .42;
    my $sub = sub {
        # my $var = .42;
        print $var;
    }
}
```

```
sub decorator {
    my $decor = shift;
    return sub {
        return $decor."@_".$decor;
    }
}

my $dq = decorator "'";
my $dd = decorator '"';
my $ds = decorator '/';

say $dq->('test');  # 'test'
say $dd->('test');  # "test"
say $ds->('test');  # /test/
```

```
my @subs;

for my $var (1..10) {
    my $sub = sub {
        return $var + $_[0];
    };
    push @subs, $sub;
}

for my $sub (@subs) {
    say $sub->(2);
}
# 3 4 5 6 7 8 9 10 11 12

for my $sub (@subs) {
    say $sub->(10);
}
# 11 12 13 14 15 16 17 18 19 20
```

```
my $fd = socket...
wait_socket_readable($fd, sub {
    read($fd, ...)
})

# ...

our %waiters;
sub wait_socket_readable {
    my ($fd,$cb) = @_;
    $select->add($fd);
    push @{ $waiters{$fd} }, $cb;
}
# Event loop:
while () {
    # ...
    for my $fd (@ready) {
        for my $cb ( @{ $waiters{$fd} } ) {
            $cb->();
        }
    }
}
```

```
my $fd = socket...

wait_socket_readable($fd, sub {
    sysread($fd, ...);

    wait_socket_writable($fd, sub {
        syswrite($fd, ...);

        wait_socket_readable($fd, sub {
            sysread($fd, ...);

            # ...
        });
    });
});

# ...
        my @wait = @{ $waiters{$fd} };
        @{ $waiters{$fd} } = ();
        for my $cb ( @wait ) {
```

```
wait_timeout 1, sub { ... };

our @deadlines;
sub wait_timeout {
    my ($t,$cb) = @_;
    my $deadline = time + $t;
    @deadlines =
        sort { $a->[0] <=> $b->[0] }
        @deadlines, [ $deadline, $cb ];
}

# Event loop:
while () {
    #...
    while ($deadlines[0][0] <= time) {
        my $next = shift(@deadlines);
        my $cb = $next->[1];
        $cb->();
    }
}
```

```
wait_timeout 1, sub {
    wait_timeout 0, sub {
        wait_timeout 0, sub {
            wait_timeout 0, sub {
                wait_timeout 0, sub {
                    wait_timeout 0, sub {
                        wait_timeout 0, sub {
                            ...
                        };
                    };
                };
            };
        };
    };
};
```

```
wait_timeout 1, sub {
    my $sub; $sub = sub {
        wait_timeout 0, $sub;
    }; $sub->();
};
```

```
wait_timeout 1, sub {
    my $sub; $sub = sub {
        wait_timeout 0, $sub;
    }; $sub->();
};
```

```
my $deadline = time + $t;
unshift @deadlines, [$deadline, $cb];
# ...
    while ($deadlines[0][0] <= time) {
        my $next = shift(@deadlines);
        my $cb = $next->[1];
        $cb->();
    }
```

```
wait_timeout 1, sub {
    my $sub; $sub = sub {
        wait_timeout 0, $sub;
    }; $sub->();
};
```

```
my $deadline = time + $t;
unshift @deadlines, [$deadline, $cb];
# ...
    while ($deadlines[0][0] <= time) {
        my $next = shift(@deadlines);
        my $cb = $next->[1];
        $cb->();
    }
```

```
our $now;
our @deadlines;

sub wait_timeout {
    my ($t,$cb) = @_;
    my $deadline = $now + $t;
    @deadlines =
        sort { $a->[0] <=> $b->[0] }
        @deadlines, [ $deadline, $cb ];
}
# Event loop:
while () {
    $now = time;
    #...
    my @exec;
    push @exec, shift @deadlines
        while ($deadlines[0][0] <= $now);
    for my $dl (@exec) {
        $dl->[1]->();
    }
}
```

Обобщённый интерфейс

```
io( $fd, READ | WRITE, $cb);

timer( $timeout, $cb );

runloop();
```

AnyEvent

```
AE::io( $fd, $flag, $cb );

AE::timer( $after, $interval, $cb );

AE::signal( $signame, $cb );

AE::idle( $cb );

AE::now();
```

AE::io

```
AE::io \*STDIN, 0, sub {
    # stdin is readable;
    my $line = <STDIN>;
    AE::io \*STDOUT, 1, sub {
        # stdout is writable
        print $line;
    };
};
```

```
AnyEvent->io( fd=>\*STDIN, poll=>'r',
    cb => sub {
        my $line = <STDIN>;
        AnyEvent->io( fd=>\*STDOUT, poll=>'w',
            cb => sub {
                print $line;
            }
        );
    }
);
```

Guard

```
my $guard = guard { # same as guard(sub { ... })
    say '$guard was unrefed';
};
say "Before...";
undef $guard;
say "After";
```

```
Before...
$guard was unrefed
After
```

Guard

```
sub Guard::DESTROY {
    my $self = shift;
    $self->[0]->() if $self->[0];
}

sub Guard::cancel {
    $_[0][0] = undef;
}

sub guard(&) {
    my $cb = shift;
    bless [$cb], 'Guard';
}
```

Guard

```
use Guard;

sub delayed_action {
    my ($smth,$cb) = @_;
    my $state = ...

    # ...

    return guard {
        cancel_action($state);
    }
}


my $w = delayed_action(..., sub { ... });
#  $w is a guard
# ...
undef $w; # cancels action
```

AE::io

```
my ($r,$w);

$r = AE::io \*STDIN, 0, sub {
    # stdin is readable;
    my $line = <STDIN>;

    $w = AE::io \*STDOUT, 1, sub {
        # stdout is writable
        print $line;

        undef $w; # not interesting
                  # in write anymore
    };
};

AE::cv->recv; # Run loop
```

AE::timer (after, period)

```
my $w; $w = AE::timer 1, 0, sub {
    undef $w;
    say "Fired after 1s";
};

my $p; $p = AE::timer 0, 0.1, sub {
    state $counter = 0;
    return undef $p if ++$counter > 5;
    say "Fired $counter time";
};

AE::cv->recv; # Run loop
```

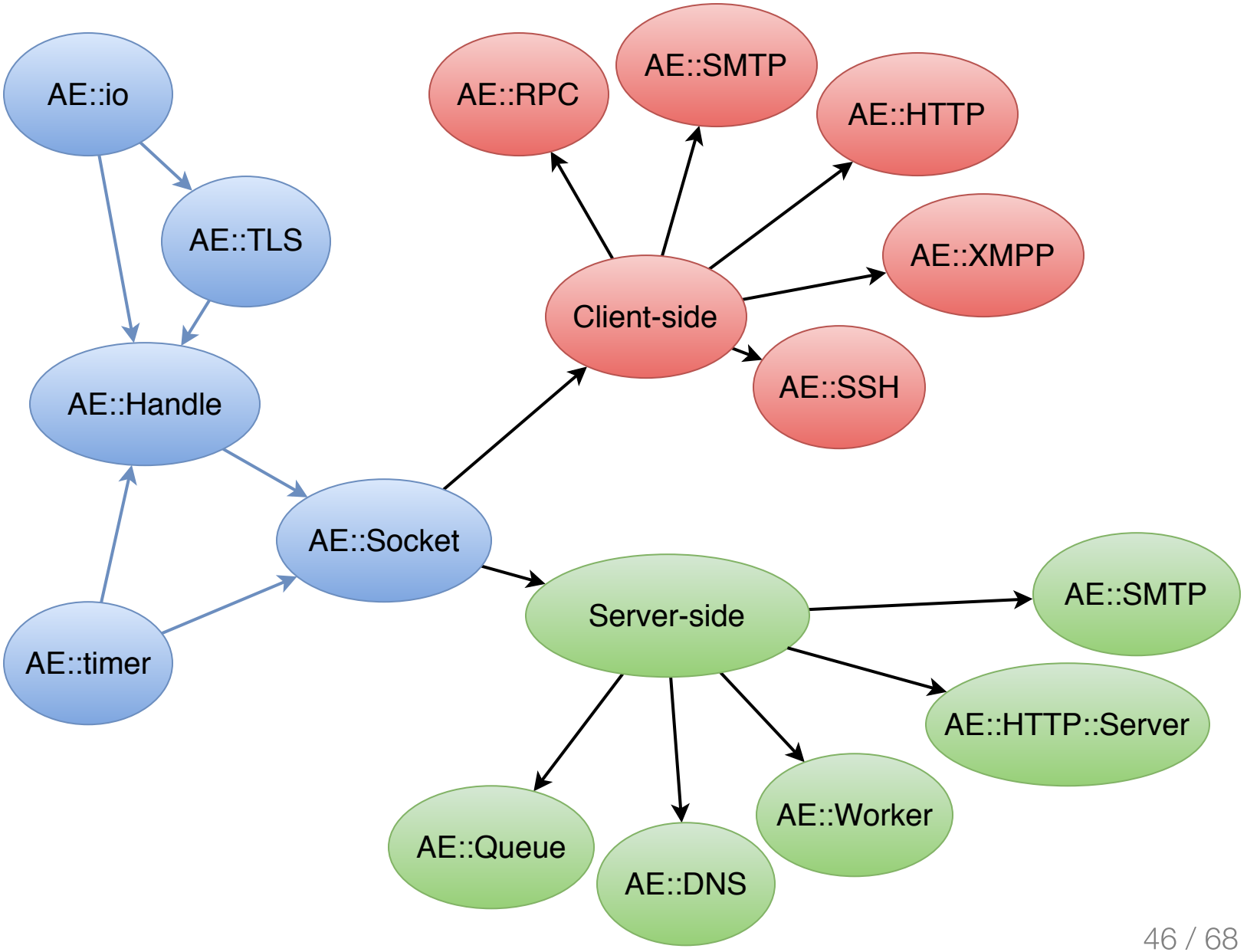
AE::signal

```
my $s;$s = AE::signal INT => sub {
    warn "Received SIGINT, exiting...\n";
    exit(0);
};

AE::cv->recv; # Run loop
```

AE::idle, AE::now

```
my $i = AE::idle sub {
    printf "now: %f, idle...\n",AE::now();
};
```

```
while () {
    $now = time;

    if (@ready) {
        # ...
    }
    elsif(@timers) {
        # ...
    }
    else {
        call_idle();
    }
}
```


AE::io
AE::TLS
AE::Handle
AE::timer
AE::Socket
AE::RPC
AE::SMTP
AE::HTTP
AE::XMPP
Client-side
AE::SSH
Server-side
AE::SMTP
AE::HTTP::Server
AE::Worker
AE::Queue
AE::DNS

AE::cv (condvar)

```
my $cv = AE::cv(); # create condvar

my $p; $p = AE::timer 0, 0.1, sub {
    state $counter = 0;
    if (++$counter > 5) {
        undef $p;
        $cv->send;
        return;
    };
    say "Fired $counter time";
};

$cv->recv;
```

AE::cv (condvar)

```
my $cv = bless {}, 'condvar';

sub condvar::recv {
    my $self = shift;

    $self->_one_loop
        while !$self->{sent};

    return @{ $self->{args} };
}

sub condvar::send {
    my $self = shift;

    $self->{sent} = 1;

    $self->{args} = [ @_ ];
}
```

AE::cv (begin/end)

```
my $cv = AE::cv;

$cv->begin;
my $w1;$w1 = AE::timer rand(),0, sub {
    undef $w1;
    say "First done";
    $cv->end;
};

$cv->begin;
my $w2;$w2 = AE::timer rand(),0, sub {
    undef $w2;
    say "Second done";
    $cv->end;
};

$cv->recv;
```

AE::cv (begin/end)

```
sub condvar::begin {
    my $self = shift;
    $self->{counter}++;
}

sub condvar::end {
    my $self = shift;

    $self->{counter}--;

    if ($self->{counter} == 0) {
        $self->send();
    }
}
```

AE::cv (begin/end/cb)

```
my $cv = AE::cv {
    say "cv done"
};

$cv->begin;
my $w1;$w1 = AE::timer rand(),0, sub {
    undef $w1;
    say "First done";
    $cv->end;
};
$cv->begin;
my $w2;$w2 = AE::timer rand(),0, sub {
    undef $w2;
    say "Second done";
    $cv->end;
};

$cv->recv;
```

AE::cv (begin/end/cb)

```
my $cv = AE::cv;

$cv->begin;
my $w1;$w1 = AE::timer rand(),0, sub {
    undef $w1;
    say "First done";
    $cv->end;
};
$cv->begin;
my $w2;$w2 = AE::timer rand(),0, sub {
    undef $w2;
    say "Second done";
    $cv->end;
};

$cv->cb(sub {
    say "cv done";
});
$cv->recv;
```

```
sub AE::cv(;&) {
    my $self = bless {}, 'condvar';
    $self->{cb} = shift;
    return $self;
}

sub condvar::cb {
    my $self = shift;
    $self->{cb} = shift;
}

sub condvar::send {
    my $self = shift;

    $self->{sent} = 1;

    $self->{args} = [ @_ ];

    if ($self->{cb}) { $self->{cb}->() };
}
```

Simple async function

```
sub async {
    my $cb = pop;

    my $w;$w = AE::timer rand(0.1),0,sub {
        undef $w;

        $cb->();
    };

    return;
}
```

Параллельное выполнение

```
my $cv = AE::cv;
my @array = 1..10;

for my $cur (@array) {
    say "Process $array[$cur]";
    $cv->begin;
    async sub {
        say "Processed $array[$cur]";
        $cv->end;
    };
}

$cv->recv;
```

Параллельное выполнение

```
my $cv = AE::cv; $cv->begin;
my @array = 1..10;

for my $cur (@array) {
    say "Process $array[$cur]";
    $cv->begin;
    async sub {
        say "Processed $array[$cur]";
        $cv->end;
    };
}

$cv->end; $cv->recv;
```

Последовательное выполнение

```
my $cv = AE::cv;
my @array = 1..10;

my $i = 0;
my $next; $next = sub {
    my $cur = $i++;
    return if $cur > $#array;
    say "Process $array[$cur]";
    async sub {
        say "Processed $array[$cur]";
        $next->();
    };
}; $next->();

$cv->recv;
```

Параллельное исполение с ограничением

```
my $cv = AE::cv;
my @array = 1..10;

my $i = 0;
my $next; $next = sub {
    my $cur = $i++;
    return if $cur > $#array;
    say "Process $array[$cur]";
    async sub {
        say "Processed $array[$cur]";
        $next->();
    };
}; $next->() for 1..5;

$cv->recv;
```

```
Process 1
Processed 1
Process 2
Processed 2
Process 3
Processed 3
Process 4
Processed 4
Process 5
Processed 5
Process 6
Processed 6
Process 7
Processed 7
Process 8
Processed 8
Process 9
Processed 9
Process 10
Processed 10
```

```
Process 1
Process 2
Process 3
Process 4
Process 5
Processed 5
Process 6
Processed 2
Process 7
Processed 4
Process 8
Processed 3
Process 9
Processed 6
Process 10
Processed 1
Processed 9
Processed 8
Processed 10
Processed 7
```

```
my $cv = AE::cv; $cv->begin;
my @array = 1..10;

my $i = 0;
my $next; $next = sub {
    my $cur = $i++;
    return if $cur > $#array;
    say "Process $array[$cur]";
    $cv->begin;
    async sub {
        say "Processed $array[$cur]";
        $next->();
        $cv->end;
    };
}; $next->() for 1..5;

$cv->end; $cv->recv;
```

Stack

```
while ()
    -> process_fds
        -> $cb
```

```
http_request 1..., sub {
    http_request 2..., sub {
        http_request 3..., sub {
            http_request 4..., sub {
                $done->();
            }
        }
    }
}
```

```
use Async::Chain;

chain
sub {
    my $next = shift;
    http_request 1..., sub { $next->() },
},
sub {
    my $next = shift;
    http_request 2..., sub { $next->() },
},
sub {
    my $next = shift;
    http_request 3..., sub { $next->() },
},
sub {
    my $next = shift;
    http_request 4..., sub { $done->() },
};
```

Coro

```
use Coro;

async { # create new stack
    say 2;
    cede; #
    say 4;
};

say 1;
cede;
print 3;
cede;

# 1 2 3 4
```

AnyEvent

- - no stack
- ~ линейный код неудобен
- + параллельный код легко
- + стек нити не ограничен
- + дедлок невозможен

Coro

- + есть стек
- + линейный код удобен
- - параллельный неудобно
- - стек ограничен
- - возможен дедлок

Домашнее задание

Необходимо написать краулер с использованием `AnyEvent` или `Coro`
Требования к роботу:

- Собрать с сайта все уникальные страницы
- Для каждой страницы запомнить её размер
- Если страниц более `10000`, собрать максимум `10000` уникальных ссылок
- Не уходить с сайта на другие сайты
- Вывести `Top-10` страниц по размеру и суммарный размер всех страниц

Модули, которые могут помочь в решении: `AnyEvent::HTTP`, `Coro::LWP`, `Web::Query`. `Web::Query` допустимо использовать только как парсер документов, но не как инструмент для скачивания

```
$AnyEvent::HTTP::MAX_PER_HOST = 100;
```

__END__