POE

POEtry ... Panel Of Experts ... Parallel Object Executor ... Parcel Out Execution ... Parenthetically Over-Engineered ... Parity Of Evil ... Part Of Elephant ... Particles Of Eternity ... Party On, Ebenezer ... Passed Out from Excitement ... Pathetically Over-Engineered ... Peace On Earth ... Penes Over-Emphasized ... Perfect Orange Eater ... Perfectly Oblique Egg-plant ... Periodically Orbits Earth ... Perl Obfuscation Engine ... Perl Object Environment ... Perl: Objectively Excellent ... Perl Objects for Enterprises ... Perl Objects for Events ... Perl On Extasy ... Perl Operating Environment ... Perl Operator Extravaganza ... Perl Over Easy ... Perl Over Ethernet ... Perl Overdrive Engine ... Perlmud Offers Expansions ... Perpetual Orgone Energy ... Persistent Object Environment ... Persnickity Oblong Erudition ... Phallic Overture Ejaculate ... Philanthropic Organization Enterprises ... Physician Order Entry ... Piece Of Eight ... Pigs, Owls, and Elephants ... Piles Of Eugh ... Pious Object Excelsior ... Plain Old English ... Plastic Orbs Everywhere ... Platonic Object Engine ... Plenty Of Everything ... Plucky Object Engine ... Poe Organizes Everything ... Poe Over Earth ... Point Of Entry ... Polyolester ... Pompous Oversexed Engineers ... Porn Operating Environment ... Porn of Esoterics ... Port of Embarkment ... Portal Of Evil ... Possibly Over-Engineered ... Post-Occupancy Evaluation ... Potatoes Of Eternity ... Potentially Omnipotent Entity ... Pow! Oof! Eek! ... Power Over Ethernet ... Power Operating Environment ... Practical Over Extraction ... Practically Overengineered Environment ... Preponderance of Evidence ... Preserve Our Essences ... Pretty Obelisk, Excavated ... Pretty Obfuscation Engine ... Pretty Obtuse Engine ... Pretty Odd Environment ... Price Of Entity ... Probable Obese Elephant ... Product Of Experts ... Products Of Eccentricity ... Prognosis: Over-Engineered ... Program Office Estimate ... Programming Over Easy ... Proliferation Of Events ... Prolifically Over-Eaten ... Purity Of Essence ... Princess On Ecstasy ... Pissed Off Elephants ... Purveyor Of Everything ... Edgar Allan POE ... Perversely Oriented Entities ... Piracy Over Ethernet ... Poe Or Environment ? ... Piece Of Eden ...

Agenda

- Principes généraux
- Fonctionnement de POE
- Un peu de pratique
- Utilisation avancée
- Interfaces graphiques et POE
- POE distribué
- Conclusion

Principes généraux

- Multitâche... coopératif
- Ni `fork()`, ni threads
- Evénements
- Domaines de prédilection
- Performances

Multitâche... coopératif (1/2)

- Ce que ça veut dire
  - plusieurs tâches en même temps
  - chaque tâche doit laisser les autres s'exécuter

- (Mauvais) Souvenirs :
  - Windows 3.x, 95
  - MacOS

Multitâche... coopératif (2/2)

- Problèmes pour un système d'exploitation
  - blocage si un processus ne rend pas la main (prog bloqué)
  - partage ressources (cpu, mémoire, i/o) inefficace et injuste
  - n'importe quel logiciel peut être lancé
    - fonctionnement +/- stable

- Appliqué à une application
  - un seul développeur / équipe réduite : couplage fort
  - (relativement) facile de respecter l'environnement

→ réduction des inconvénients du multitâche coopératif

→ révèle la puissance d'un environnement multitâche

Ni `fork()`, ni threads

- Multitâche mais...
  - un seul processus
  - un seul thread

- Applis facilement portables
  - même si appel système `fork()` non disponible
  - même sur les perls non threadés
    - ce qui inclut la majorité des distributions Perl

- Applis faciles à écrire
  - pas d'IPC : tout est partagé et accessible
  - pas de verrous : une seule tâche en cours, tout est atomique

Evénements

- Application POE
  - suite de traitements d'événements
  - les tâches s'envoient des événements
  - les gestionnaires sont des callbacks

- Evénement = « il s'est passé quelque chose d'intéressant »
  - un fichier est prêt
  - une alarme a expiré
  - une nouvelle connexion est arrivée

Domaines de prédilection

- Applications réseau
  - nature multitâche et événementielle
  - composants de haut niveau disponibles
  - application complexe en un minimum de lignes

- Interfaçage facile avec toolkits graphiques
  - Tk, Gtk, Curses
  - nature événementielle

Performances

- Framework avancé
  - perte d'efficacité / procédural
  - reste performant
  - suffit pour la majorité des besoins
  - avantages nombreux

- Applications exigeantes : évaluation nécessaire

- Parlons chiffres...
  - 1000 messages / seconde : POE convient sans hésiter
  - +10000 messages / seconde : POE ne suffit plus
  - entre les deux : à étudier (selon traitements)

Fonctionnement de POE

- Le noyau
- Les sessions
- Les alias
- 3 niveaux d'abstraction

Le noyau

- Implémenté dans `POE::Kernel`

- Le kernel :
  - achemine les événements
  - appelle le code devant les gérer
  - vérifie des conditions et lance des événements
    - alarmes
    - fichiers prêts à être traités

- Méthode `run()`
  - boucle principale du programme
  - ne `return` pas tant qu'il y a des tâches

Programme POE de base

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

use strict;
use warnings;
use POE;

# initialisation et création des tâches POE...

POE::Kernel->run();
exit;

# déclaration des fonctions gérant les événements.
```

Les sessions (1/2)

- Tâches
  - appelées sessions en POE
  - objets `POE::Session`
- Une session a des ressources privées :
  - descripteurs de fichiers
  - espace de stockage privé (la **heap**)
  - ses propres événements
  - peut même avoir des sessions filles
- Analogie OS
  - session = processus
  - POE s'assure que les données d'une session sont séparées des autres

Les sessions (2/2)

- Principes
  - reste en vie tant qu'il faut (!)
  - choses appartenant à une session
    - requièrent qu'elle fasse quelque chose
    - la gardent donc en vie
  - s'arrête quand plus rien à faire
  - arrêter une session : s'assurer qu'elle ne possède plus rien

- Exemple
  - socket qui a besoin d'être gérée
  - session reste en vie tant qu'elle possède la socket
  - si socket fermée + libérée : POE libérera la session

Les alias (1/2)

- Session ID
  - référence une session
  - alloué par le kernel lors de sa création
  - permet d'envoyer un message à la session
  - ... mais pas pratique

- Alias
  - nom symbolique pour une session
  - autant qu'on veut
  - accessible aux autres sessions

Les alias (2/2)

- Effet de bord
  - tout le monde peut envoyer des événements à un alias
  - POE ne peut le deviner à l'avance
  - sessions aliasées donc gardées en vie

- Cependant...
  - si seuls les alias gardent toutes les sessions en vie
  - POE envoie le signal `IDLE`
  - si pas de réveil : POE envoie le signal `ZOMBIE`
    - qui n'est pas gérable par les sessions...
    - ... qui vont donc mourir

3 niveaux d'abstraction

- Degrés d'abstraction différents
- Balance différente entre effort développeur / contrôle
- Flexibilité accrue pour chaque tâche
- Possibilité de mélanger les composants de différents niveaux

Composants de haut niveau

- `POE::Component::*`
- Puissants composants, peu d'effort
- Serveurs ou clients TCP
  - web, syslog, ...
  - générique
- Interfaces à des applis / bibliothèques
  - pcap, oggenc, mpd, ...
- Gestion du temps évoluées
  - cron, ...

Toolkit de niveau intermédiaire

- `POE::Wheel::*`
- Quand les POCO ne sont pas appropriés
- Réutilisation de « roues » standard
- Surveillance des lignes d'un fichier (tail -f)
- Gestion d'I/O bufferisées non bloquantes
  - y compris sockets !
- Lancement et contrôle de processus

Fonctions bas niveau de POE

- Quand même les roues de POE sont trop avancées
- Code moins évolué
  - mais contrôle plus complet
- Gestion des alarmes
- Gestion des signaux
- Interfaces à l'appel système `select`

Un peu de pratique

- Création de session
- Hello, world!
- Passage de paramètres
- Passage de messages

Création de session

- Exemple

```
POE::Session->create(
    inline_states => {
        _start     => \&on_start,
        file_ready => \&on_file_ready,
    }
);
```

- Evénements
  - prédéfinis : `_start`, `_stop`
  - autres : libres, API de la session

Hello, world!

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

use warnings;
use strict;
use POE;

POE::Session->create(
   inline_states => {
      _start => sub { print "Hello, world!\n"; }
   },
);

POE::Kernel->run();
exit;
```

Passage de paramètres

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

use warnings;
use strict;
use POE;

POE::Session->create(
   inline_states => {
      _start => \&on_start,
   },
   args => \@ARGV,
);

POE::Kernel->run();
exit;

sub on_start {
   my @args = @_[ARGO .. $#_];
   print "Hello, @args!\n";
}
```

Passage de messages (1/3)

- Exemple
  - logger : réagit aux messages debug
    - ok, pas très intéressant
    - même pas besoin d'une session
  - main : utilisera logger
    - réveil toutes les secondes
    - envoi de 10 messages
    - puis arrêt

Passage de messages : logger (2/3)

```
POE::Session->create(
   inline_states => {
       _start => \&on_logger_start,
       debug  => \&on_logger_debug,
   },
);

sub on_logger_start {
   my $kernel = $_[KERNEL];
   $kernel->alias_set( 'logger' );
}

sub on_logger_debug {
   my @args = @_[ARGO .. $#_];
   warn @args;
}
```

Passage de messages : alarm (3/3)

```
POE::Session->create(
   inline_states => {
      _start => \&on_alarm_start,
      tick   => \&on_alarm_tick,
   },
);

sub on_alarm_start {
   $_[HEAP]->{nb} = 0;
   $_[KERNEL]->delay_set( 'tick', 1 );
}

sub on_alarm_tick {
   my ($k,$h) = @_[KERNEL, HEAP];
   $k->post( 'logger', 'debug', 'this is a test: msg ', $h->{nb} );
   return if ++$h->{nb} > 9;
   $k->delay_set( 'tick', 1 );
}
```

Utilisation avancée

- Modulariser le code
- Où sont stockées les sessions ?
- Envoi de messages à soi-même
- Remplacement de la heap
- L'événement `_start`
- Evénements non gérés
- Gestion des signaux
- Le bon citoyen POE

Modulariser le code

- 1 session = 1 module
  - encapsulation
  - ré-utilisation
  - convention : `Foo::Bar->spawn()`
- 1 session, gérée par des objets différents

```
my $logger = Logger->new( ... );
my $error  = Error->new( ... );
POE::Session->create(
    inline_states => {
        _start => \&on_start,
    },
    object_states => [
        $logger => [ 'debug', 'warning' ],
        $error  => {
            critical  => 'graceful_exit',
            emergency => 'emergency_exit',
        },
    ],
);
```

Où sont stockées les sessions?

- Jamais par l'utilisateur
- POE s'en occupe tout seul

- Raisons
  - créer une session est banal
    - sauvegarder + détruire à la main serait lourd
  - POE utilise bcp de références circulaires
    - tel fichier appartient à telle session, telle session gère ce fichier
    - fuites de mémoire garanties si suivi pas attentif
  - cela permet à POE de savoir quand une session n'est plus nécessaire
    - passage au ramasse-miettes

Envoi de messages à soi-même

- Utilisation de `$_[SESSION]`

```
$_[KERNEL]->post( $_[SESSION], $event, @args );
```

- `yield()` plus simple

```
$_[KERNEL]->yield( $event, @args );
```

- Messages traités dans un ordre FIFO

Remplacement de la heap

- Comment
  - Au moment de la création
  - par n'importe quel scalaire

```
POE::Session->create(
     # ...
     heap => My::Heap->new( ... ),
     # ...
);
```

- Meilleure encapsulation

L'événement `_start`

- Envoyé automatiquement par POE
- Notifie à une session qu'elle a démarré
- Gestionnaire `_start`
  - initialise la session
  - crée des choses
    - si rien n'est géré après `_start`...
    - ... ramasse-miettes...
    - ... arrêt immédiat

Evénements non gérés

- ie, non définis lors de la création de la session
  - discrètement ignorés
  - pas une erreur (exemple : événements de débug)
- Peut être embêtant
  - typo, oubli du handler... difficile de savoir !
  - assertions pour cas marginaux de POE

```
sub POE::Kernel::ASSERT_DEFAULT () { 1 }
use POE;
```

```
a 'unknown' event was sent from unknown.pl at 8 to session 2(POE::Session=ARRAY(0x808b8e8))
but session 2 (POE::Session=ARRAY(0x808b8e8)) has neither a handler for it nor one for _default
```

  - ralentit POE !
- Sinon, trappé par `_default`
  - `ARG0` = nom du message, `ARG1` = [liste de paramètres]
  - gestionnaire fourre-tout, dispatch personnalisé

Gestion des signaux

- Signaux POSIX
  - enregistrement auprès du noyau

```
$_[KERNEL]->sig( USR1 => 'usr1' );
```

  - `ARG0` = nom du signal

- Cas des signaux terminaux
  - HUP, INT, KILL, QUIT et TERM
  - arrêt du programme si non gérés
  - utilisation de `sig_handled()` pour indiquer qu'ils ont été traités

Le bon citoyen POE (1/2)

- Composants doivent coopérer

- Bannir sleep()
  - met l'ensemble du programme en pause
  - utiliser `delay_set()` / `alarm()`

```
$_[KERNEL]->delay_set( 'tick', 10, @args );
$_[KERNEL]->alarm_set( 'alarm', $epoch, @args );
```

- Terminer le programme
  - bannir `exit` ou `die` !
    - termine toutes les sessions en cours
  - envoyer des messages de fin

Le bon citoyen POE (2/2)

- Couper les longues boucles
  - pour laisser une chance aux autres sessions de s'exécuter

```
sub long_loop_start {
    print "start...\n";
    $_[HEAP]->{i} = 0;
    $_[KERNEL]->yield('long_loop');
}
sub long_loop {
    my $h = $_[HEAP];
    if ( $h->{i} < 1_000_000 ) {
        # ...
        $h->{i}++;
        $_[KERNEL]->yield('long_loop');
    } else { print "done.\n"; }
}
```

  - ou 100 par 100 (plus efficace)

```
sub long_loop {
    my $h = $_[HEAP]; my $i = 0;
    while ( $i++ < 100 && $h->{i} < 1_000_000 ) {
        # ...
        $h->{i}++;
    }
    if ( $h->{i} < 1_000_000 ) {
        $_[KERNEL]->yield('long_loop');
    } else { print "done.\n"; }
}
```

Interfaces graphiques et POE

- Boucles d'événements
- Spécificités GUI
- Exemple simple
- Ajout de commandes
- Avantages

Boucles d'événements (1/2)

- POE fournit sa boucle d'événements
  - basée sur select
  - 100% pur Perl
- Adaptation aux autres boucles
  - toolkits graphiques fournissent la leur
    - interfaçage avec Gtk, Glib (Gtk2), Tk, Wx, Curses, ...
  - non-graphiques
    - Event.pm, IO::Poll (potentiellement plus efficace)
- API de POE reste la même
  - indépendamment de la boucle d'événements

Boucles d'événements (2/2)

- En chargeant le module souhaité avant POE

```
use Gtk;
use POE;
```

- En chargeant la boucle souhaitée avec POE

```
use POE qw[ Loop::Gtk ];
```

- En forçant la boucle voulue

```
use POE::Kernel { loop => "Gtk" };
```

Spécificités GUI

- Tk refuse de s'exécuter si aucun widget
  - POE en crée un par défaut
  - disponible via `$poe_main_window`

- Surveillance d'un widget
  - enregistrement via signal_ui_destroy()
  - signal UIDESTROY

```
$heap->{gtk_toplevel_window} = Gtk::Window->new('toplevel');
$kernel->signal_ui_destroy( $heap->{gtk_toplevel_window} );
```

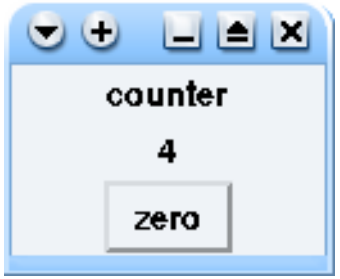
Exemple

```
#!/usr/bin/env perl
use strict;
use warnings;
use Tk;
use POE;

POE::Session->create(
   inline_states => {
      _start => \&on_start,
      tick   => \&on_tick,
   }
);
POE::Kernel->run();
exit;

sub on_start {
   my ($k, $h) = @_[KERNEL, HEAP];
   $poe_main_window->Label( -text => 'counter' )->pack;
   $poe_main_window->Label( -textvariable => \$h->{counter} )->pack;
   $k->delay_set('tick', 1);
}

sub on_tick {
   $_[HEAP]->{counter}++;
   $_[KERNEL]->delay_set('tick', 1);
}
```

Ajout de commandes

- Différence avec Tk
  - pas un callback direct
    - non intégré à POE
    - ne respecte pas l'ordonnancement POE
  - session se poste un événement POE
    - utilisation de `postback()`

```
sub on_start {
    my ($k, $s, $h) = @_[KERNEL, SESSION, HEAP];
    # ... création des labels
    $poe_main_window->Button(
        -text    => 'zero',
        -command => $s->postback('zero')
    )->pack;
    # ... fin de l'initialisation
}

sub on_zero { $_[HEAP]->{counter} = 0; }
```



Avantages

- Facilité de programmation
  - fenêtres différentes...
    - gérées par sessions dédiées
  - sessions non-graphiques
    - gèrent en arrière-plan connexions réseau
    - ou toute autre tâche n'ayant rien à voir avec la GUI
- Modularité
- Applications complexes faciles à coder

POE distribué

- IKC
- Service logger
- Client IKC
- Aller plus loin

IKC

- Distribution d'une appli
  - sur plusieurs machines
  - ou même machine mais processus différents

- Communication inter-kernel
  - Inter-Kernel Communication (IKC)
  - `POE::Component::IKC`

Service logger

- Logger vu ci-dessus
  - implémenté en tant que service
  - accessible depuis n'importe quelle session IKC
  - `POE::Component::IKC::Server`

- Création du serveur IKC (en fait un composant POE avec sa session)

```
POE::Component::IKC::Server->spawn(
    ip   => 'localhost',
    port => 17000,
    name => 'my-server',
);
```

- Publication de l'interface acceptée via IKC

```
sub on_start {
    my $k = $_[KERNEL];
    $k->alias_set('logger');
    $k->call( IKC => publish => 'logger', ['debug'] )
}
```

Service logger complet

```
#!/usr/bin/env perl
use strict;
use warnings;
use POE;
use POE::Component::IKC::Server;

POE::Component::IKC::Server->spawn(
    ip   => 'localhost',
    port => 17000,
    name => 'my-server',
);
POE::Session->create(
    inline_states => {
        _start => \&on_start,
        debug  => \&on_debug,
    }
);
POE::Kernel->run();
exit;

sub on_start {
    my $k = $_[KERNEL];
    $k->alias_set('logger');
    $k->call( IKC => publish => 'logger', ['debug'] )
}

sub on_debug {
    warn @_[ARG0 .. $#_];
}
```

Client IKC

- Connexion au serveur

```
POE::Component::IKC::Client->spawn(
    host => 'localhost',
    port => 17000,
    name => "client-$$",
    on_connect => \&on_connect,
);
```

- Souscription au service IKC publié
  - `poe://<kernel>/<session>/<événement>`
  - jokers autorisés : `poe://<kernel>/logger/*`

```
$k->post('IKC', 'subscribe', [ 'poe://my-server/logger/debug' ]);
```

- Utilisation du service

```
$k->post('poe://my-server/logger', 'debug', $h->{nb}++ );
```

  - tout est sérialisé / dé-sérialisé automatiquement
    - via `Data::Dumper` / `eval` (attention à la sécurité)

Client IKC complet

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

use strict;
use warnings;
use POE;
use POE::Component::IKC::Client;

POE::Component::IKC::Client->spawn(
    host => 'localhost',
    port => 17000,
    name => "client-$$",
    on_connect => \&on_connect,
);
POE::Kernel->run();
exit;


sub on_connect {
    POE::Session->create(
        inline_states => {
            _start => \&on_start,
            tick   => \&on_tick,
        }
    );
}

# - to be continued ...
```

```
# - continued

sub on_start {
    my $k = $_[KERNEL];
    $_[HEAP]->{nb} = 0;
    $k->delay_set('tick', 1);
    $k->post(
        'IKC', 'subscribe',
        [ 'poe://my-server/logger/debug' ]
    );
}


sub on_tick {
    my ($k, $h) = @_[KERNEL, HEAP];
    $k->post(
        'poe://my-server/logger',
        'debug', $h->{nb}++
    );
    $k->delay_set('tick', 1);
}
```

IKC : aller plus loin

- Passage d'événements retour
  - avec ou sans RSVP
    - (retour différé dans le temps)
  - protocole de type RPC
- Communication peer to peer
  - un kernel serveur à l'origine des communications
  - plusieurs clients de type « esclave » (ou worker)
  - distribution des tâches par un scheduler
- Surveillance intégrée des kernels distants
  - lorsqu'un kernel se dé / connecte
- perldoc `POE::Component::IKC`

Conclusion

- Bref survol
- Framework puissant et bien pensé
  - facilité de programmation
  - composants de haut niveau
  - attention au code spaghetti
- Merci Rocco Caputo
- Bientôt la version 1 (actuellement 0.9999)

Essayez-le !