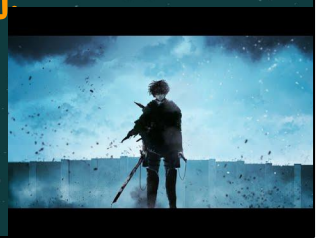
Erlang

Clase 2

Bienvenides a la segunda clase de Erlang.

Warm-up: 10 minutos. Programar un servicio de Broadcast.

```
-export([iniciar/0, finalizar/1]).
-export([broadcast/2, registrar/1]
-export([loopBroadcast/1]).
```



Un servicio de Broadcast lo podemos pensar como un objeto concurrente que acepta dos funciones:

- Registrar() función que registra el proceso que invoca la función
- Broadcast(Msg) que envía un mensaje a los procesos que se han registrado.

Clase 2

- Receive(Timeouts)
- Registración de Procesos
- Revisitamos Modelo Cliente/Servidor
- Mención al Scheduler de Procesos.
- Manejo y propagación de Errores

En esta clase entraremos en algunos detalles más sobre el lenguaje.
Reforzaremos el concepto de buzón de mensajes, y veremos una técnica para darle prioridad a ciertos mensajes.
Veremos como asignarles nombres a ciertos procesos, facilitando la implementación de librerías, etc.
Revisitaremos el modelo cliente/servidor desde el punto de vista de Erlang.
Revisaremos el concepto de scheduler y veremos como reducir la prioridad de procesos en Erlang.
Finalmente veremos como hacer un manejo buen manejo de errores en Erlang.

Receive

```
receive
  Patrón1 -> Cuerpo1
  Patrón2-> Cuerpo2
  .....
  PatrónN -> CuerpoN
end
```

La primitiva de concurrencia **receive** nos permite buscar mensajes en el buzón.
El proceso de Pattern Matching se basa en buscar en el orden de llegada de los mensajes, el primer mensaje que pueda ser matcheado con alguno de los patrones.

Receive nos permite hacer pattern matching con los mensajes que estén en el buzón. El proceso de pattern matching es entonces en orden. Esto es, en el orden en que llegaron los mensajes, se comparan con los patrones en el orden en el que son escritos y en la primer coincidencia, se instancian las variables necesarias y se ejecuta el cuerpo correspondiente.
Debido a que dependemos en qué orden vinieron los mensajes, el orden de las guardas **no nos garantizan ninguna prioridad**.

Receive con Timeout

```
receive
  Patrón1 -> Cuerpo1
  Patrón2-> Cuerpo2
  .....
  PatrónN -> CuerpoN
after
   ExprTO -> CuerpoT
end
```

Al agregar la cláusula **after** podemos estipular un tiempo de espera, que puede ser calculado en tiempo de ejecución.

Hay dos tiempos (átomos) que son útiles:

- 0
- infinity

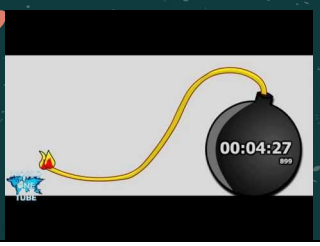
Fue sutilmente presentado en la práctica, y ahora lo revisamos para ver su utilidad. La expresión `ExprTO` es una expresión que evalúa a un entero interpretado en milisegundos (la precisión es dependiente de la implementación de erlang y el sistema op).

Veremos 3 ejemplos concretos

+ `sleep/1` bloquea por cierto tiempo
+ `procesos_prioridad/0`.
+ `empty_mailbox/0` vacía el buzón de un proceso [Actividad de la siguiente slides]

Actividad 1: 5 minutos.

Programar una función `empty_mailbox/0` que vacíe el buzón de un procesos



Registración de Procesos

Toda la comunicación entre procesos en Erlang es a través del identificador del proceso, pero no siempre es práctico (ni deseable)

- Servidores Globales (e intercambiables)
- Cuestiones de Seguridad

Puede cambiar el proceso que actualmente está siendo el que espera a los clientes.
Tiene sentido tener una entidad un poco más abstracta y tener nombres.

BIFs Registración de Procesos

- register(Name,Pid) : registra al identificador de proceso Pid con el nombre Name
- unregister(Name) : borra el nombre Name del registro
- whereis(Name) : retorna el identificador de proceso asociado a Name
- registered() : retorna una lista de los procesos registrados en el sistema

Notar que es `whereis` y no `whois`. El PID nos da información de DóNDE ESTÁ EL PROCESO!

Actividad: 5 minutos.

Implementar un servicio de Broadcaster, registrar el proceso como broadcaster y propagar el cambio:

```
-export([iniciar/0, finalizar/0]).
-export([broadcast/1, registrar/0]).
```

Donde `iniciar/0` ya no dará el PId del proceso encargado de administrar los mensajes, sino que registrará ese proceso con nombre `broadcaster`.

Broadcast lo que hace es simplemente estar a la espera de mensaje

Revisitamos el modelo cliente/servidor, y para esto nos viene bien el uso de nombres!!

Definamos mejor el modelo cliente servidor.
Lo podemos descomponer en 3:

+ Servidor: **un** proceso que ofrece algún servicio (esto ya lo teníamos)
+ Protocolo: la forma y procedimiento utilizado para comunicarnos con el servidor
+ Libraría de acceso: las funciones que nos permiten interactuar con el servidor (funciones que implementan el protocolo).

El servidor y el protocolo es algo que ya vimos en C. Veamos mejor entonces lo que sería la librería de acceso.

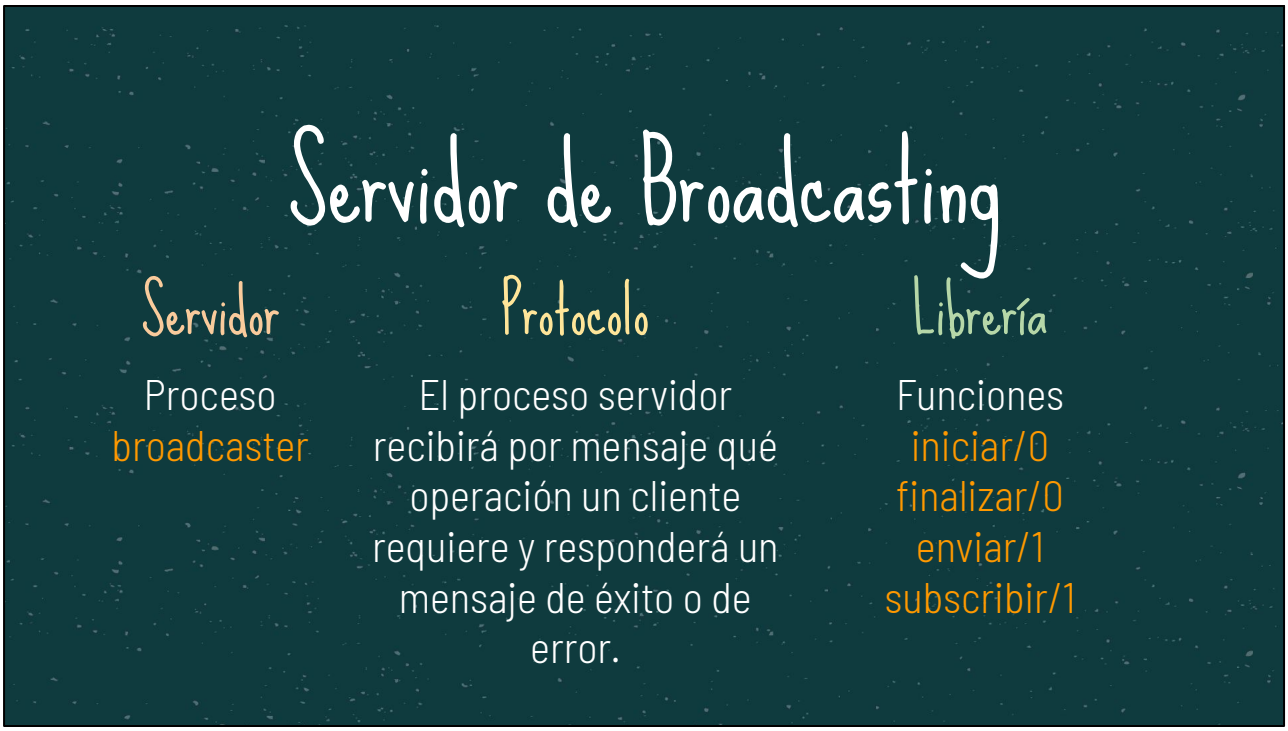
La idea es abstraer lo más posible al cliente (o usuario de nuestro servicio) de los protocolos internos de comunicación con el Servidor.
De esta manera podemos desconectar lo más posible al cliente:

+ Oculta las componentes internas del servidor: Donde está, quien es, pueden ser incluso múltiples servidores, etc
+ Nos permite además cambiar la implementación libremente: Básicamente como una librería.

Actividad en Conjunto
Repasemos Broadcaster

Repasemos el Broadcaster entre todos.

Recordar implementar un Broadcaster confiable, es decir, con acuse de recepción por parte del servidor.



Revisar broadcasting utilizando lo que sabemos. Registración de procesos, y modelo cliente servidor.

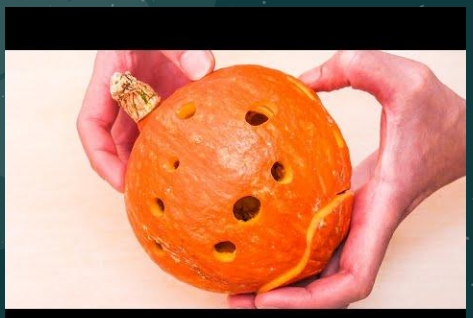
La distribución de la unidad de cómputo en erlang es responsabilidad del que implementa la máquina virtual. Recordemos que Erlang ejecuta sobre una máquina virtual y **no** sobre el sistema operativo directamente. Esto nos trae un montón de beneficios que son los que ya vimos.
Pero el se le pide que la repartición de tareas sea:

+ Justa: todo proceso que pueda ser ejecutado será ejecutado realmente (Fair Scheduling)
+ Ningún proceso se puede apropiar del cpu indefinidamente

De hecho hay una unidad llamada `time slice` (porción de tiempo) que suele ser de `500` reducciones (llamado de funciones).

Los procesos se ejecutan con prioridad `normal` por defecto.
¿Quien se encargaba de asignarles tiempo de ejecución antes?

Descanso de 16 minutos.



Veremos entonces unos mecanismos para el manejo de errores en Erlang.
Como se esperaran al ser un lenguaje con tipado dinámico tendremos errores de runtime (como en C).

Excepciones

Las excepciones se utilizan para indicar detener la ejecución de un proceso e indicar un estado de error. El mecanismo consiste de dos partes, una que captura la excepción y otra que la dispara.

El lector atento habrá visto el chiste :D.
La capturación de excepciones cobrará sentido cuando vean cómo disparar una excepción.

La idea de throw es simplemente lanzar una excepción, y enviarla “hacía atrás” hasta encontrar un catch que la reciba

¡Live Coding!

Mini ejemplo con la división por cero

+ Error badmatch : `1 = 3` lanza la excepción {'EXIT', PId, badmatch}
+ Error badarg: Una bif se llamó con un tipo incorrecto,
+ Error case_clause: M = 3, case M of
  1 -> true;
  2 -> false
  end.
+ Error if_clause: lo mismo que case, ninguna guarda se cumple
+ Error undef: llamar a una función que no existe, `foo(1)` meintras que foo no se declaró en ninguna lado. O no se exportó!
+ Error badarith: mala operación aritmética, como dividr por 0.

Antes de ver los errores, veamos como termina correctamente un proceso en Erlang.

Todo proceso que tiene un error en ejecución (runtime error):

+ División por cero
+ Error en el pattern matching
+ Invocar a una función que no existe

etc. Termina de forma **abnormal** invocando a la función exit(Razón), siendo Razón distinta al átomo normal

Ejemplo entre ping/pong para motivar el problema. Ping muere y pong no sabe

Linkeo de Procesos

Los procesos pueden indicar cuando terminan de forma abnormal mediante un sistema de links.
Lo creamos con la primitiva link(PIdOtro)

La idea es básicamente que Erlang al crear procesos para todo, estos pueden fallar y otros procesos no estar ni enterados que hubo un error en el sistema.
La idea de generar un `link` entre dos procesos es para evitar dicho problema, y cuando un proceso falla, el otro también.
`link` crea una conexión bidireccional entre el proceso que la invoca y el que es pasado como argumento.
Cuando un proceso termina, le envía una señal a todos los procesos a los que estaba conectado con la razón por la que terminó.
Por defecto el programa que recibe la señal `normal` la ignora.
En el caso que sea `abnormal` u otra razón, puede pasar lo siguiente:

+ Enviar todos los mensajes del buzón de un proceso al otro
+ Morir
+ Propagar el comportamiendo a los procesos linkeados por el receptor.

Resolver el problema de que muera ping o pong pero el otro no se enteró.
La solución es que pong muere también

Terminación Abnormal

Detectar la señal

el proceso que recibe la señal la detecta como un proceso más

Matar al proceso

Se mata el proceso que recibe la señal

Propagar la señal

Se propaga la señal a los procesos que están linkeados a éste último

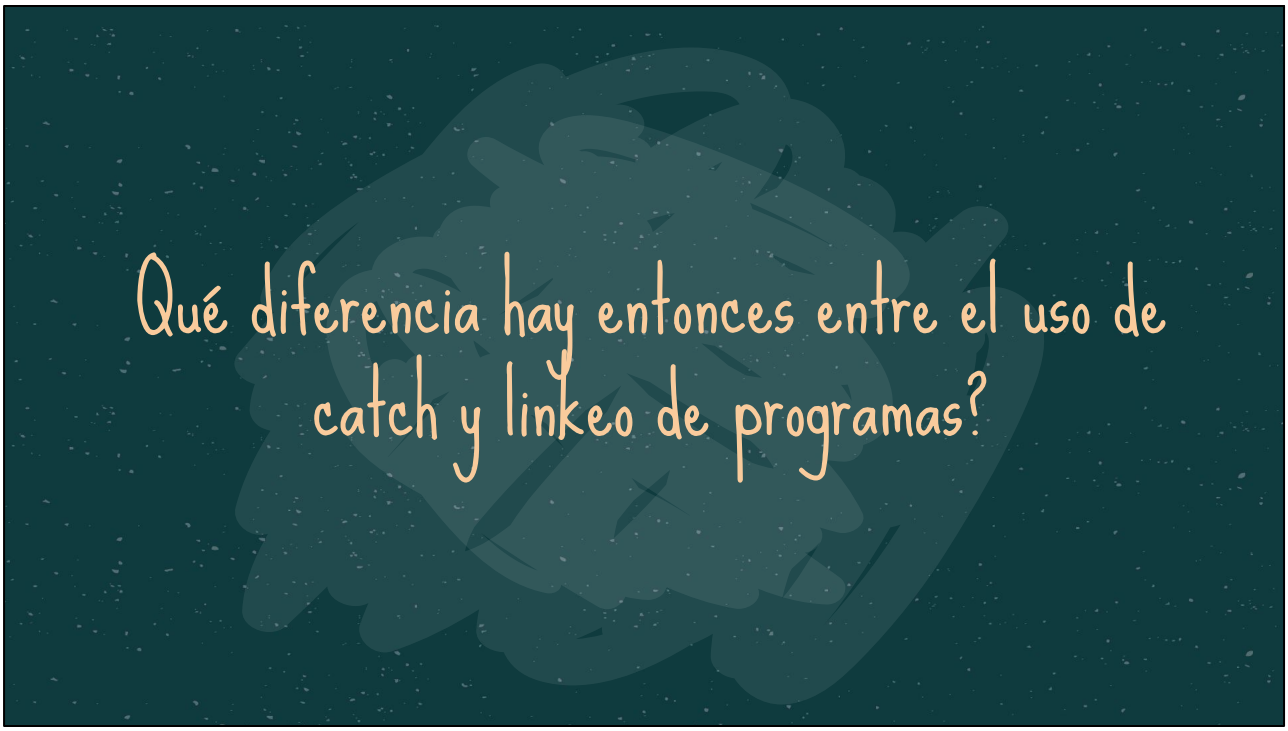
Cuando se produce una terminación con una razón distinta a `normal` se puede.
De esta manera uno puede conectar los procesos, y si uno falla por algún motivo, el resto se puede enterar y:

+ Tratar de solucionar el error
+ Abandonar el barco silenciosamente
+ Avisarle al resto y morir juntos

¡Live Coding!

BIF útil: spawn_link
Para poder detectar la señal hay que indicarselo a Erlang con:
process_flag(trap_exit, true)

con `spawn_link` creamos un proceso y generamos la conexión al mismo tiempo.
con `process_flag(trap_exit, true)` activamos la capturación de la señal de muerte.
Y viene de la siguiente forma `{'EXIT', From, Reason}`
Entonces ping muere, pong se entera y decide terminar elegantemente.



La diferencia radica en el uso. Las excepciones y catch están pensadas para que las use un proceso, y el linkeo para propagar errores a través de la red de procesos. Ambas funciones permiten implementar software robusto, un proceso puede recuperarse de un errores, y un sistema puede recuperarse en el caso que un proceso fallé.
Por ejemplo, podemos tener un proceso llevando cuenta del sistema, y mirando que el proceso que brinda el servicio esté vivo. Si el servidor falla, el monitor puede revivirlo.

Descanso de 10 mins

Descanso de 10 minutos

Descanso de 5 mins

Descanso de 10 minutos