Programación Concurrente

Paso de mensajes

El intercambio de mensajes se puede efectuar asíncronamente o síncrono, en el cual el que envía el mensaje se bloque hasta que recibe el mensaje.

Paso asíncrono

Puede ser confiable o no confiable (send and pray).

Interacción mediante paso de mensajes

Los mensjaes recibidos son solicitudes y son puestos en una cola para ser manejados uno por uno.

El proceso que envía no para de trabajar mientras espera una respuesta a su mensaje y sigue atendiendo a los demás en la fila.

Peligros:

* No elimina las condiciones de carrera.
* Tampoco elimina Deadlocks, cuando un proceso se queda en espera de la respuesta de un mensaje.

Procesos y Concurrencia en Erlang

Las funciones son asíncronas, por lo que no se espera a que se obtenga un resultado para continuar con el resto de las instrucciones.

En Erlang, la Concurrencia se implementa mediante la creación y comunicación de procesos.

* Proceso: unidad de cómputo separada, es una función en ejecución, que se ejecuta concurrentemente con otros procesos en el sistema.
* Los procesos de Erlang no comparten memoria (datos) con otros procesos.
* Los procesos se comunican mediante el paso de mensajes (modelo de programación concurrente)

Procesos

Los procesos pertenecen al lenguaje y no al sistema operativo.

* Spawn: crear procesos
* Send: mandar mensajes
* Receive: recibir mensajes

Creación de Procesos

Spawn/1 o spawn/3, crean un nuevo proceso concurrente y regresa su identificador.

Pid = **spawn**(función)

Pid = **spawn**(modulo, función, listaArgs)



Envío de mensajes

Un mensaje se envía a otro proceso mediante el primitivo **!** (send), como Pid ¡ mensaje.

* El send regresa el mensaje que envías.
* El envío de mensaje es asíncrono
  + El que envía el mensaje continúa con lo que estaba haciendo, **no espera.**
  + El sistema no informa al que envía si el mensaje se entregó.
* El mensaje puede ser cualquier término Erlang válido.
* El valor de retorno de ! es el mensaje que envía, es asociativo por la derecho
  + Pid1 ¡ Pid2 ¡ Mensaje
  + Es equivalente a Pid1 ¡ (Pid2 ¡ Mensaje)



Si el receptor no ha terminado, todos los mensajes le son entregados en el mismo orden

en el que se le envían.

Recepción de Mensajes

El primitivo **receive** se usa para recibir mensajes

Receive

Patron 1 when Guardia1 – Acciones 1

…

End

Cada proceso tiene su propio buzón de correo.

Todos los mensajes que se le envían a un proceso se almacenan en su buzón en el orden en la que se reciben.

Receive regresa el resultado de la ultima acción ejecutada.

Cuando un mensaje es recibido, el sistema trata de empatarlo secuencialmente.

* Si un mensaje empata, se elimina del buzón y se evalúan las acciones relacionadas.
* Si no empata con ningún patrón, permanece en el buzón para su procesamiento posterior y el proceso continua con el siguiente mensaje en su buzón.

El proceso que evalúa un receive es suspendido hasta que un mensaje sea empatado.

Los mensajes que arriban a un proceso no pueden bloquear otros mensajes para ese proceso.

El buzón se puede llenar con mensajes que no empatan con los patrones.

Es responsabilidad del programador asegurarse que el buzón no se llene.

Mensajes de procesos específicos

Cuando se quiere recibir mensajes de un proceso específico, el que envía debe mandar su propio pid en el mensaje.

Tiempos de espera

Recordemos que reciece supende la ejecuci’on de la función mientras el proceso no empate con alguno de los mensajes.

Receive puede incluir tiempos de espera (timeouts) para no bloquear al proceso para siempre si no recibe un mensaje.

Receive

After T -> true

End.

Las acciones de espera se evalúan si no se selecciona un mensaje antes que termine el tiempo de espera.

Ejemplo: suspender un procose T milisegundos

Ejemplo: detectar dobles clicks

Receive

{mouse, click} ->

Receive {mouse, click} -> doublé\_click

After

Double\_click\_interval() -> single\_click

End

End

Tiempos especiales de espera

1. Infinity, especifica una espera que nunca ocurrirá.
2. 0 especifica que la espera se acaba inmediatamente
   1. Antes, el sistema trata todos los mensajes actualmente en el buzón.

Recordar: Los mensajes se procesan en el orden que se reciben.

Registro de Procesos

* El PID de un proceso se requiere para mandarle un mensaje
  + Esto es muy seguro, pero inconveniente porque el procsoe le tiene que enviar su PID a todos los otros procesos que se quieran comunicar con el.
  + Erlang tiene un método para publicar los PIDs para que cualquier proceso en el sistema les pueda enviar mensajes => **Registo de procesos.**
* Primitivos:
  + Register(Alias, Pid) – registra el proceso Pid con el nombre Alias (un atomo).
  + Unregister(Alias) – remueve cualquier registro con el nombre Alias.
  + Whereis(Alias) -> Pid | undefined – determina si el nombre Alias esta registrado.
  + Registered() – regresa una lista con todos los procesos registrados en el sistema.

Ejemplo de Registro de procesos

A picture containing diagram, text

Description automatically generated

A picture containing text

Description automatically generated