Programación Declarativa Avanzada

Curso 2011/201

Juan Rodríguez Hortalá

Hoja 2 de Problemas
Erlang: delegación y supervisión básica

a) Define una función `apply_timeout(Fun, Args, Timeout)` que ejecute en un nuevo proceso una llamada a la función `Fun` utilizando los argumentos de la lista `Args`, y devuelva el valor calculado si dicho valor es devuelto por el proceso hijo (trabajador) antes de `Timeout`, o devuelva el átomo `timeout` en otro caso.
Ejemplo: `apply_timeout(fun(X, Y) -> X + Y end, [1,2], 1000)` debe devolver el valor `3`.
Sugerencia: estudia la función `apply/2` del módulo `erlang`.

b) Reimplementa la función `apply_timeout/3` del ejercicio anterior como una función `safe_apply_timeout/3` que adicionalmente:

- Mate al proceso trabajador en caso de que se alcance el timeout, de manera que no se tenga un proceso en ejecución que potencialmente pueda agotar la memoria de la máquina (por ejemplo por intentar calcular `fib(200)` para la función `fib/1` de la hoja de ejercicios anterior.
- En caso de que el proceso trabajador muera ejecutando la llamada `Fun(Args)` (por ejemplo por una división entre cero) nos devuelva el motivo de que esa ejecución se abortara.

c) Define una función `par_map(Fun, List)` que realice la versión paralela de la función `lists:map/2`, es decir, que asumiendo que `Fun` es una función de aridad uno, cree un nuevo proceso para cada elemento de `List` en el que se aplique `Fun` a dicho elemento, y luego construya una lista resultado que respete el orden de los elementos en `List`. Por ejemplo `par_map(fun(X) → X + 1 end, [0,1,2])` debe devolver la lista `[1,2,3]`.

d) Reimplementa `par_map/2` como una función `par_map(Fun, List,Timeout)` que adicionalmente mate a todos los procesos trabajadores en caso de que se alcance el timeout, y que en caso de que algún proceso trabajador termine anormalmente entonces se mate a todos los otros procesos trabajadores y se devuelva el motivo de la terminación anormal junto con el elemento de la lista de entrada para el que se produjo el error.
Sugerencia: estudia la función `now/0` del módulo `erlang` y las funciones del módulo `timer`.

e) Implementa una otra versión más del map paralelo `par_map(Fun, List, NProc, Timeout)` que cree como máximo `NProc` procesos trabajadores, de forma que en el caso de que la lista sea mayor que el número de procesos trabajadores haga que cada proceso trabajador procese una sublista de la lista de entrada. Como en versiones anteriores cuando se alcance el Timeout se matará a todos los procesos trabajadores, y en caso de terminación anormal de algún proceso trabajador entonces se matará al resto de procesos trabajadores y se devolverá el segmento de la lista para el que se produjo el error.

f) Implementa una función `par_qsort/1` que realice una versión paralela del algoritmo de

ordenación de quicksort, de forma que se creen dos procesos nuevos que ejecuten cada una de las llamadas recursivas que se encargan de ordenar los elementos mayores y menores del pivote, respectivamente.
Implementa también una función `par_qsort(Lista, LimitLength)` que ordene la lista de entrada usando Quicksort paralelo, pero que caso de que la longitud de la lista sea menor que `LimitLength` entonces utilice la función `qsort/1` defina en la hoja de ejercicios anteriores que implementa ordenación mediante Quicksort secuencial. De esta forma las llamadas recursivas a `par_qsort/2` pasarán a ejecutar un código secuencial cuando el tamaño de la lista de entrada sea tan pequeño que la sobrecarga de la creación y comunicación entre procesos no compense la compartición de trabajo entre varios procesos.