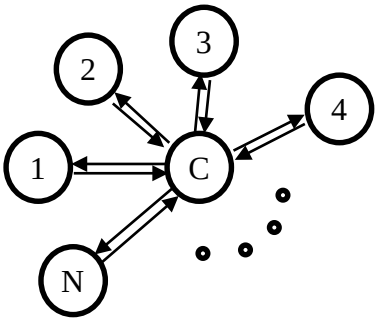
Escola Politècnica Superior d'Enginyeria de Manresa
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Examen de PCTR

12 de desembre de 2013 | Enginyeria de Sistemes TIC | 2 hores

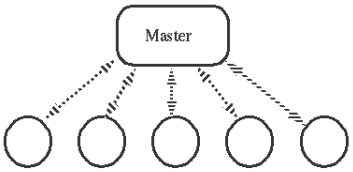
---

Exercici 1 (5 P): Escriviu una funció que inicií N processos en forma d'estel. El node central ha d'enviar M missatges a cadascun dels nodes esclaus i aquests responen amb el nº del missatge. El node central no pot enviar el següent missatge a un altre node fins que rebi la resposta del node. Quan els processos hagin rebut els M missatges han de finalitzar correctament.



Exercici 2 (5 P): En aquest problema es mostra una situació on tenim un procés (el màster) que supervisa uns altres processos (els esclaus). En un exemple real l'esclau podria, per exemple, estar controlant diferents unitats de maquinari. La feina del màster és assegurar que tots els processos esclaus estiguin actius. Si un esclau s'estavella (potser a causa d'un defecte de programari), el màster ha de reiniciar l'esclau caigut.

Escriviu un mòdul *ms* amb les següents funcions:



- start(N) - Inicia el màster i diu que comencin N processos esclaus. Registreu el procés màster amb l'àtom *master*.
- to_slave(Missatge, N) - Envia un missatge al màster i aquest el transmès a l'esclau N. L'esclau ha de finalitzar (i ser reiniciat pel màster) si el missatge és "morir" .

El màster ha de detectar la mort d'un procés esclau, reiniciar-lo i visualitzar un missatge indicant el que ha passat.
L'esclau ha d'imprimir tots els missatges que rep excepte el de "morir".

Pistes:

El màster ha d'atrapar els missatges de sortida i crear enllaços a tots els processos esclaus.
El màster hauria de tenir una llista dels identificadors dels processo esclaus i els seus números associats.

Exemple:
> ms:start(4). => true
> ms:to_slave(hola, 2). => {hola, 2} L'esclau 2 ha rebut el missatge hola
> ms:to_slave(morir, 3). => {morir, 3} El màster a reiniciat l'esclau 3

Apèndix: Llista de funcions que us poden ser útils

lists:map(Fun, List1) -> List2
Types:
Fun = fun((A) -> B)
List1 = [A] / List2 = [B] // A = B = term()
Takes a function from As to Bs, and a list of As and produces a list of Bs by applying the function to every element in the list. This function is used to obtain the return values. The evaluation order is implementation dependent.

lists:nth(N, List) -> Elem
Types:
N = integer() >= 1 / 1..length(List)
List = [T, ...] / Elem = T
T = term()
Returns the Nth element of List.

lists:seq(From, To) -> Seq // lists:seq(From, To, Incr) -> Seq
Types:
From = To = Incr = integer()
Seq = [integer()]
Returns a sequence of integers which starts with From and contains the successive results of adding Incr to the previous element, until To has been reached or passed (in the latter case, To is not an element of the sequence). Incr defaults to 1.