

# LAB3

Jordi Calatayud Álvarez

Fèlix Andrés Navarro

## Worker

Procés que vol enviar un missatge multicast a un grup en intervals aleatoris.

Espera fins que rep el seu missatge abans d'enviar un altre per a prevenir un overflow de missatges en el sistema.

Està connectat al process gui, que es simplement una finestra coloreada.

La finestra serà inicialment negra, en RGB {0,0,0}, es el estat inicial del worker.

Cada missatge enviat al worker es un nombre sencer N dins d'un interval, de 1 a 20. Canvia el seu estat i per tant canvia de color

Per exemple:

Inicial	->	Següent
{R, G, B}	N	{G, B, (R+N) rem 256}

La seqüència de missatges, per tant, es important que tingui el mateix ordre per a que tots tinguin el mateix ordre de colors.

Podem canviar:

- El modul de multicast
- Temps de sleep. Temps en milisegons que ha de esperar fins que puga enviar el següent missatge
- Temps de Jitter

## Experiments

Set up the basic multicast system, and use the following test program to experiment with different values for Sleep and Jitter. Does it keep workers synchronized? Justify why. Note that we are using the name of the module (i.e. basic) as a parameter to the start procedure. We will easily be able to test different multicast implementations. Sleep stands for the average number of milliseconds the workers should wait until the next message is sent. Jitter stands for the average number of milliseconds of network delay

**-Compilem els moduls toty, worker, basic i gui:**

```
1> c(toty).
{ok,toty}
2> c(worker).
worker.erl:10:18: Warning: erlang:now/0 is deprecated; see the "Time and Time Correction in Erlang" c
tion
% 10|      {A1,A2,A3} = now(),
%   |      ^
worker.erl:11:5: Warning: random:seed/3 is deprecated; use the 'rand' module instead
% 11|      random:seed(A1, A2, A3),
%   |      ^
worker.erl:48:16: Warning: random:uniform/1 is deprecated; use the 'rand' module instead
% 48|      Msg = {Id, random:uniform(?change)},
%   |      ^
{ok,worker}
3> c(basic).
basic.erl:8:18: Warning: erlang:now/0 is deprecated; see the "Time and Time Correction in Erlang" cha
on
% 8|      {A1,A2,A3} = now(),
%  |      ^
basic.erl:9:5: Warning: random:seed/3 is deprecated; use the 'rand' module instead
% 9|      random:seed(A1, A2, A3),
%  |      ^
basic.erl:36:35: Warning: random:uniform/1 is deprecated; use the 'rand' module instead
% 36|      timer:sleep(random:uniform(Jitter)),
%   |      ^
{ok,basic}
```

```
4> c(gui).
{ok,gui}
```

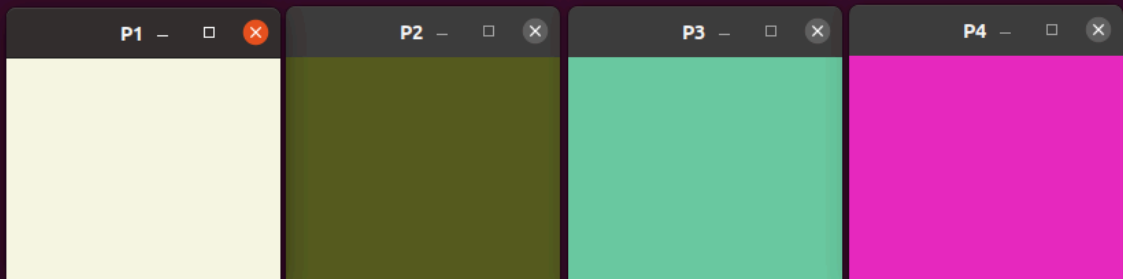
```
8> toty:start(basic, 1000, 500).
```

## -Compilem total.erl:

```
6> c(total).
total.erl:8: Warning: erlang:now/0: Deprecated BIF. See the "Time and Time Correction in Erlang" chapter of the ERTS User's Guide for more information.
total.erl:9: Warning: random:seed/3: the 'random' module is deprecated; use the 'rand' module instead
total.erl:55: Warning: random:uniform/1: the 'random' module is deprecated; use the 'rand' module instead
{ok,total}
```

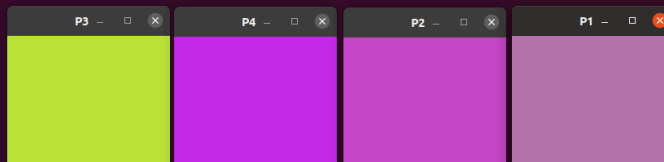
## -Provem el sistema basic de multicast amb els següents paràmetres:

```
7> toty:start(basic, 50, 10).
true
8> 
```



## -Provem amb paràmetres de sleep i de jitter més elevats:

```
8> toty:start(basic, 200, 100).
* 2: syntax error before: toty
8> toty:start(basic, 200, 100).
** exception error: bad argument
    in function  register/2
       called as register(toty,<0.143.0>)
    in call from toty:start/3 (toty.erl, line 5)
9> 
```



**-Encara més elevats:**

```
f3324858@aul-1942:~/Escriptori$ erl
Erlang/OTP 22 [erts-10.6.4] [source] [64-bit] [smp:6:6] [ds:6:6:10] [async-threads:1]

Eshell V10.6.4 (abort with ^G)
1> toty:start(basic, 500, 250).
true
2> 
```



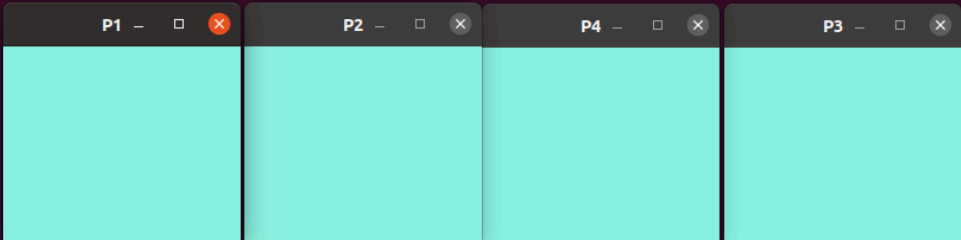
**-Parem l'experiment d'aquesta forma:**

```
2> toty:stop().
stop
3> 
```

**-Ara iniciarem el sistema total de multicast, per veure les diferències:**

```
f3324858@aul-1942:~/Escriptori$ erl
Erlang/OTP 22 [erts-10.6.4] [source] [64-bit] [smp:6:6] [ds:6:6:10] [async-threads:1]

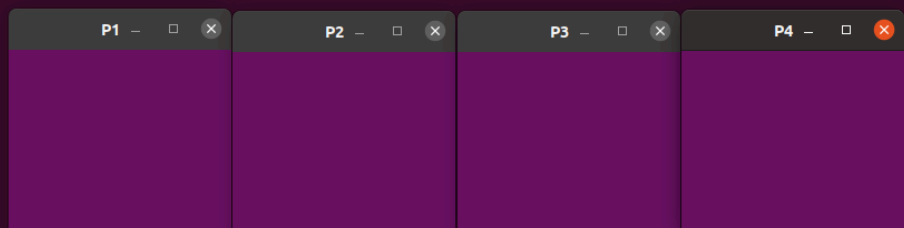
Eshell V10.6.4 (abort with ^G)
1> toty:start(basic, 500, 250).
true
2> toty:stop().
stop
3> toty:start(total, 50, 10).
true
4> 
```



-I també provarem amb diferents paràmetres de sleep i de jitter:

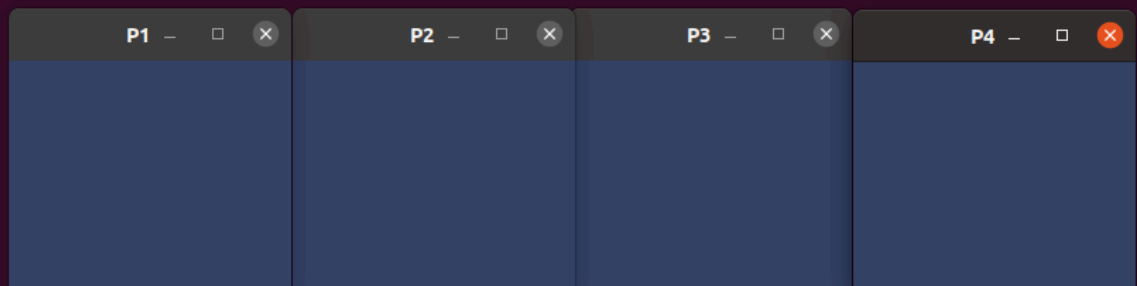
```
f3324858@aui-1942:~/Escriptori$ erl
Erlang/OTP 22 [erts-10.6.4] [source] [64-bit] [smp:6:6] [ds:6:6:10] [async-threads:1]

Eshell V10.6.4 (abort with ^G)
1> toty:start(basic, 500, 250).
true
2> toty:stop().
stop
3> toty:start(total, 50, 10).
true
4> toty:start(total, 100, 50).
** exception error: bad argument
    in function register/2
       called as register(toty,<0.131.0>)
    in call from toty:start/3 (toty.erl, line 5)
5> □
```



```
f3324858@aui-1942:~/Escriptori$ erl
Erlang/OTP 22 [erts-10.6.4] [source] [64-bit] [smp:6:6] [ds:6:6:10] [async-threads:1]

Eshell V10.6.4 (abort with ^G)
1> toty:start(total, 500, 250).
true
2> □
```



## EXPERIMENTS:

Quan es compara el sistema de multicast bàsic amb el sistema de multicast amb ordre total, les diferències en la capacitat de mantenir la sincronització dels treballadors són clares i justificades per la naturalesa dels dos mecanismes:

### 1. Sincronització amb el multicast amb ordre total:

- El sistema de multicast amb ordre total garanteix que tots els treballadors reben i processen els missatges en el mateix ordre. Això s'aconsegueix implementant un mecanisme de consens entre els nodes, com es descriu a l'algoritme de l'ISIS, on cada missatge ha de ser proposat i acordat abans de ser lliurat.
- Aquest mecanisme elimina completament el risc de desincronització independentment dels valors de Sleep i Jitter, sempre que el sistema no falli.

### 2. Impacte de Sleep i Jitter:

- En el cas del multicast amb ordre total, Sleep i Jitter continuen afectant els retards i la velocitat del sistema, però no provoquen desincronització. Això es deu al fet que els missatges no es lliuren fins que tots els nodes han arribat a un consens sobre l'ordre.
- Encara que Jitter introdueix variabilitat en els temps d'arribada dels missatges, l'ordre total s'imposa perquè els treballadors no processen cap missatge fins que s'estableixi un ordre acordat.

### 3. Comparació experimental:

- Multicast bàsic: Com que els missatges s'envien i es processen immediatament després de rebre'ls, els treballadors poden desincronitzar-se fàcilment si l'ordre d'arribada no és consistent entre nodes.
- Multicast amb ordre total: Aquí, els treballadors es mantenen sincronitzats perquè només es lliuren missatges amb un ordre acordat, evitant qualsevol discrepància.

### 4. Estudi teòric sobre el nombre de missatges:

Per a utilitzar total es fa servir un cert nombre de missatges. Per exemple: necessitem enviar 1 missatge a  $n$  workers per a fer la sol·licitud de multicasting, després aquests workers tornen a contestar al que ha sol·licitat amb un missatge, i per últim qui va sol·licitar envia un missatge a tots els workers de la xarxa. En total s'envien  $3n$  missatges, on  $n$  es el nombre de workers de la xarxa

### 5. Conclusió:

- Multicast bàsic no pot mantenir la sincronització de treballadors en situacions on hi ha variabilitat en la latència (alt Jitter) o freqüència elevada de missatges (baix Sleep). Això és degut a la manca d'un protocol per imposar un ordre global.
- Multicast amb ordre total, per contra, garanteix sincronització independent de les condicions de xarxa, gràcies al seu mecanisme d'acord d'ordre global. Això el fa molt més robust i fiable per mantenir consistència entre els treballadors.