# **PORTAFOLIO PRÁCTICA 3 - MPI**

**EJERCICIO MULTIPLES PRODUCTORES-CONSUMIDORES**

En el nuevo programa, en vez de identificar cada proceso con un identificador único 0,1,2 para el productor, bufer y consumidor respectivamente, se identifica al productor con np procesos (4 en este caso), nc=5 procesos para consumidores y el buffer con el identificador 4.

Como la solución es basada en el uso de etiquetas he creado dos variables constantes enteras para la etiqueda de productor y consumidor:

Cons int etiq\_productor = 0,

Etiq\_consumidor = 1;

Como indica el enunciado del problema, la función de los productores y la de los consumidores reciben como parámetro el número de orden del productor o del consumidor, respectivamente (esos números son los números de orden en cada rol, comenzando en 0, no son los identificadores de proceso). Estos números de orden se calculan en el main

El productor ahora produce num\_items/np y el consumidor consume num\_items/nc. En el caso de la función del productor en el envio de mensaje Ssend se le pasa los siguientes valores:

MPI\_Ssend( &valor\_prod, 1, MPI\_INT, id\_buffer, etiq\_productor, MPI\_COMM\_WORLD );

Y para los consumidores, en la function consumidor tanto para el envio Ssend para la peticion como Recv para recibir el mensaje se le pasan los siguientes valores:

MPI\_Ssend( &peticion, 1, MPI\_INT, id\_buffer, 0, MPI\_COMM\_WORLD);

MPI\_Recv ( &valor\_rec, 1, MPI\_INT, id\_buffer, 0, MPI\_COMM\_WORLD,&estado );

En la function buffer, se calcula en vez de la id\_emisor\_aceptable, como es una solucion basada en etiquetas, la etiqueta acceptable segun si solo se puede producir o consumir:

if ( num\_celdas\_ocupadas == 0 ) // si buffer vacío

etiqueta\_aceptable = etiq\_productor; // $~~~$ solo prod.

else if ( num\_celdas\_ocupadas == tam\_vector ) // si buffer lleno

etiqueta\_aceptable = etiq\_consumidor; // $~~~$ solo cons.

else // si no vacío ni lleno

etiqueta\_aceptable = MPI\_ANY\_TAG; // $~~~$ cualquiera

Y se recibe el mensaje con la etiqueta aceptable:

MPI\_Recv( &valor, 1, MPI\_INT, MPI\_ANY\_SOURCE, etiqueta\_aceptable, MPI\_COMM\_WORLD, &estado );

En la function Main el unico cambio realizado ha sido controlar que el numero de procesos que se pasa está correcto y pasar id\_propio como parámetro a la función\_productor y función consumidor:

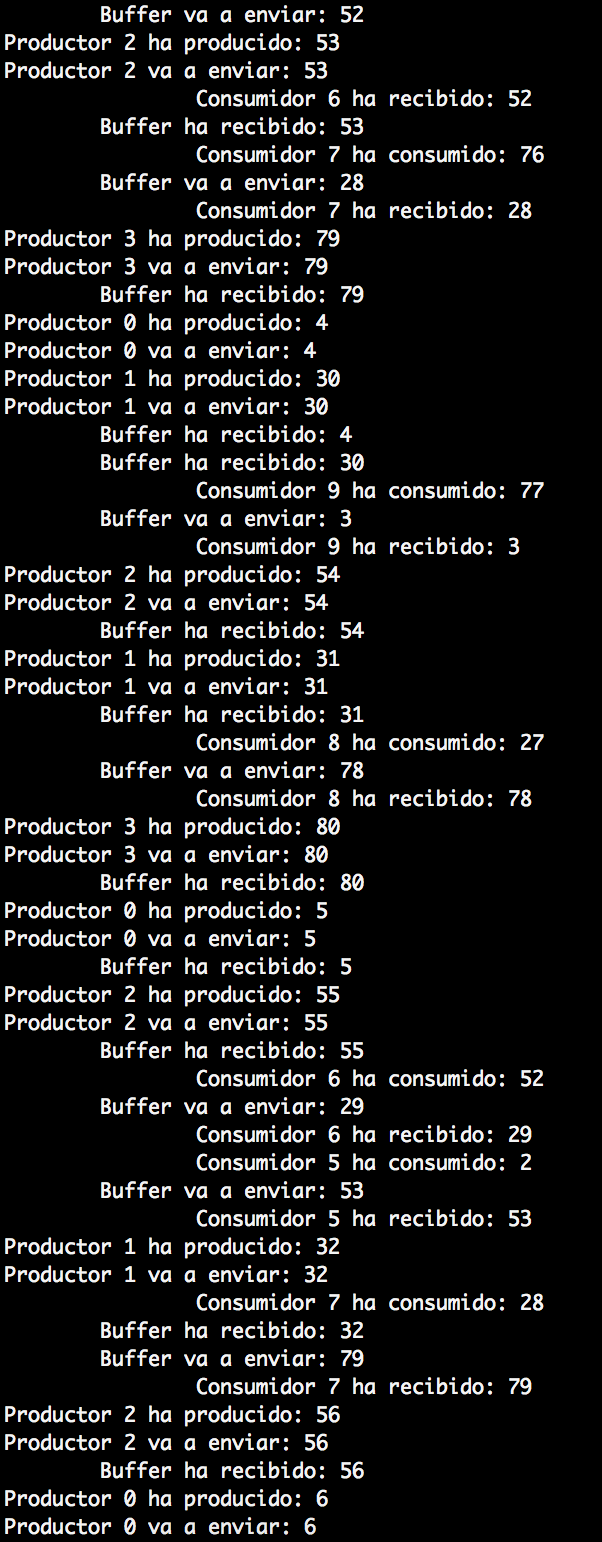
if (num\_items % nc != 0 || num\_items % np != 0){

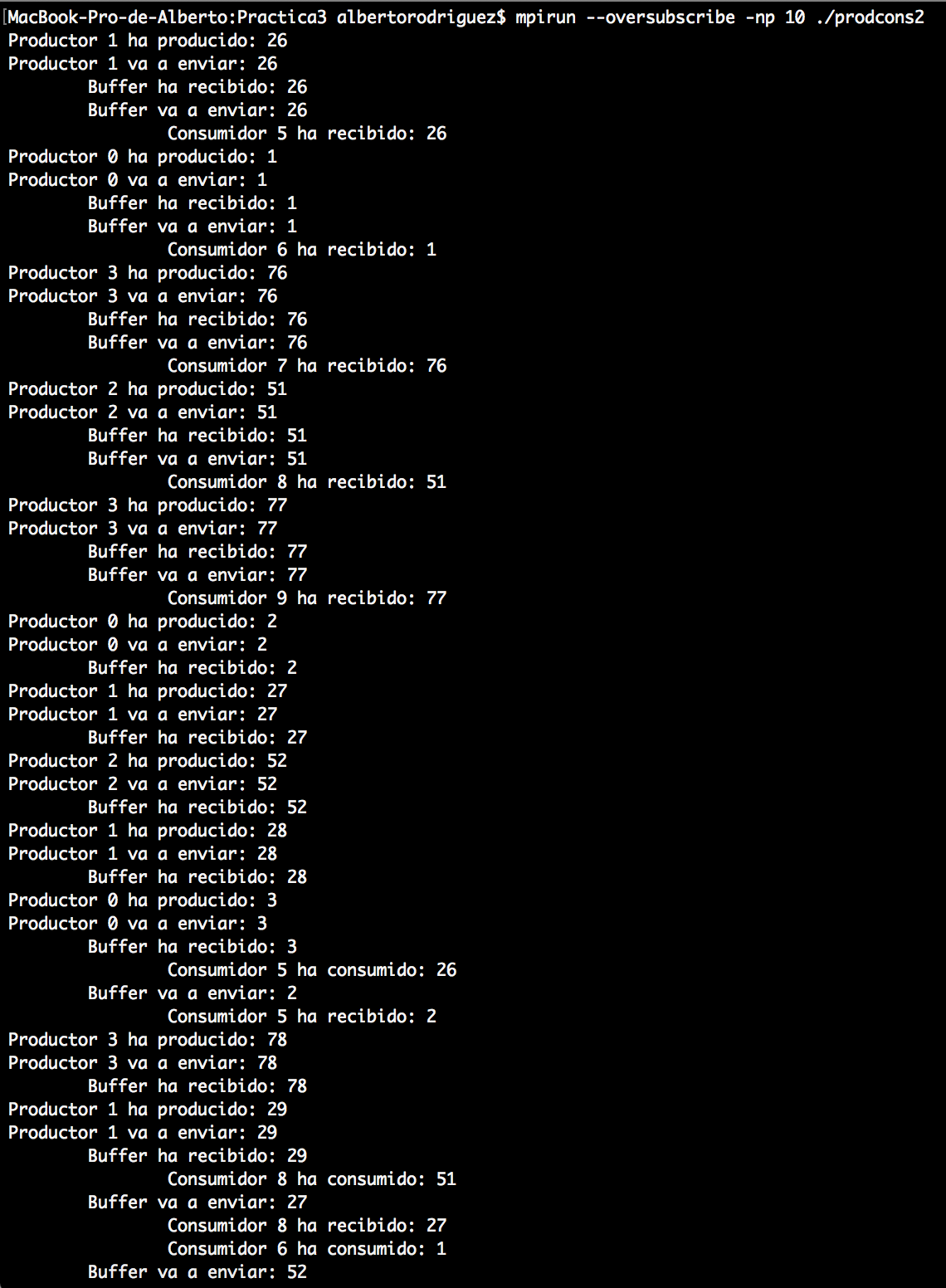
cout << "error: num\_items debe ser múltiplo de nc y de np " << endl;

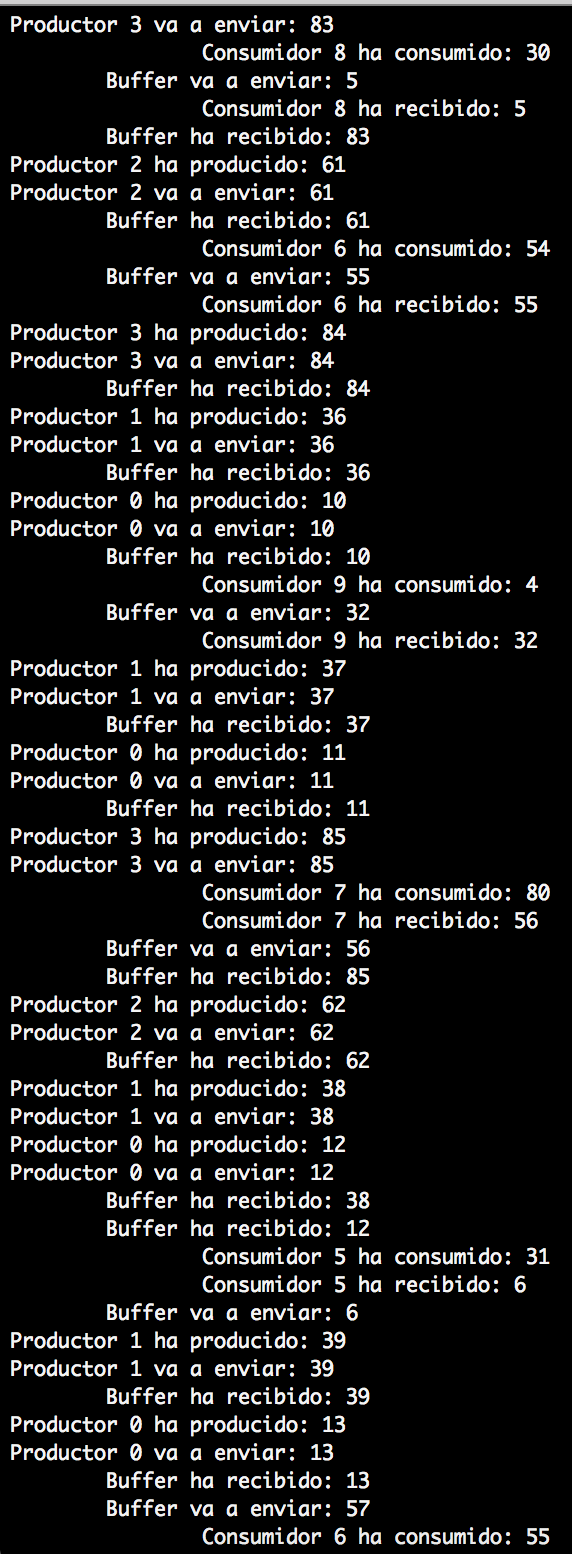
return 1;

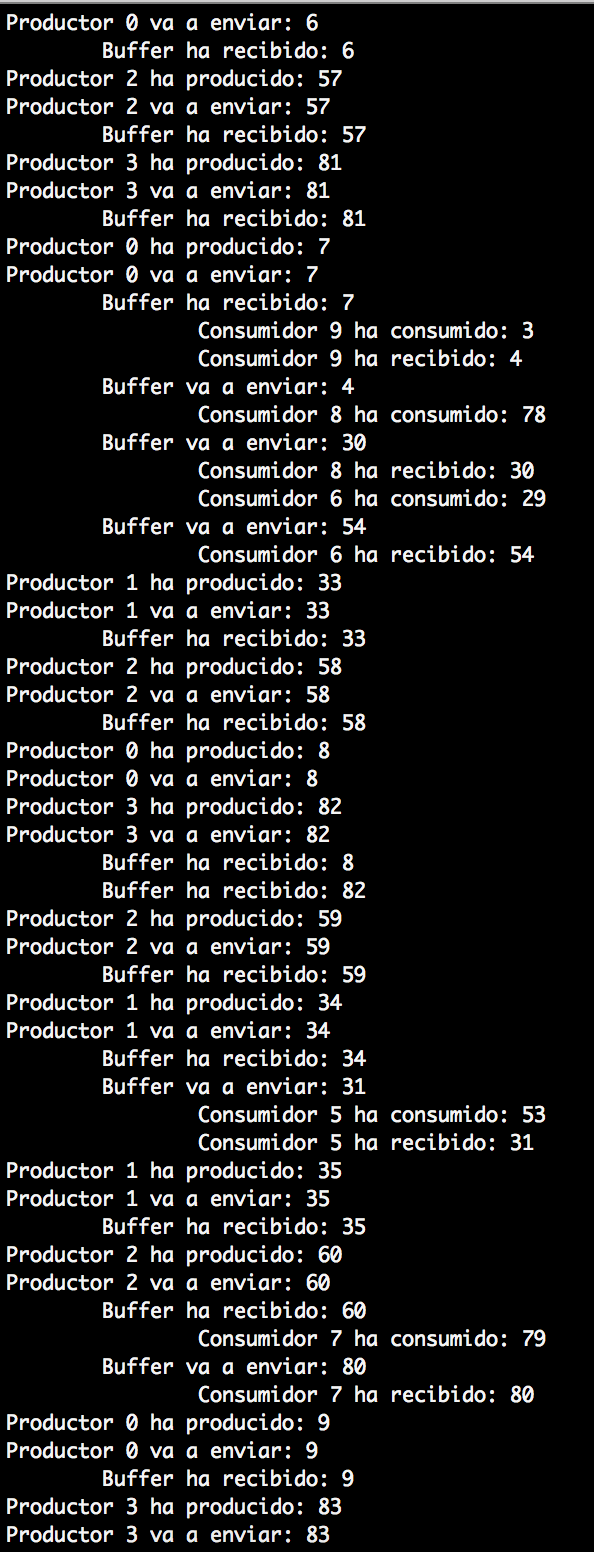
}

Ejemplo Salida del programa:







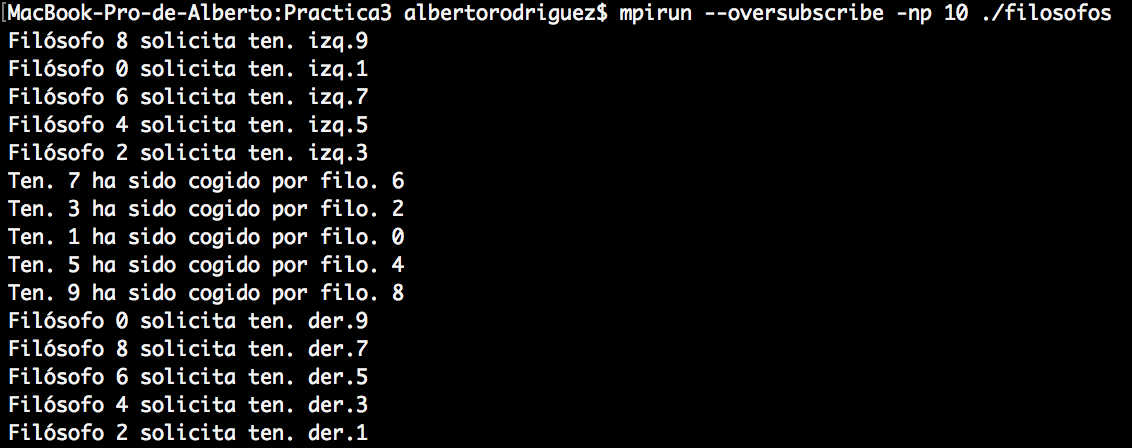


ADJUNTO CÓDIGO ARCHIVO -> prodcons2-mu.cpp

**EJERCICIO FILOSOFOS**

Archivo filoso-plantilla.cpp copiado en filósofos-interb.cpp y completado según las directrices del guión de la práctica.

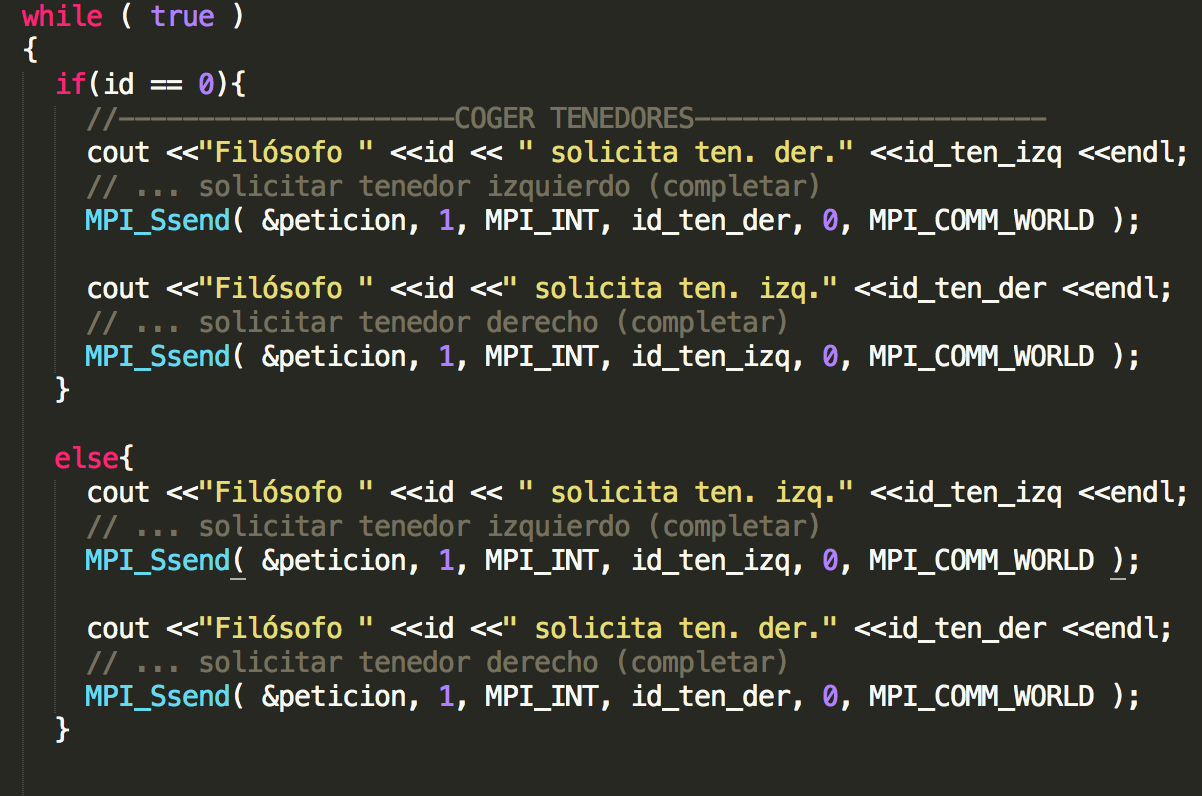
Una vez hecho esto, al ejecutarlo compruebo que se da interbloqueo cuando un filósofo solicita un tenedor y este ya lo tiene otro filosofo:



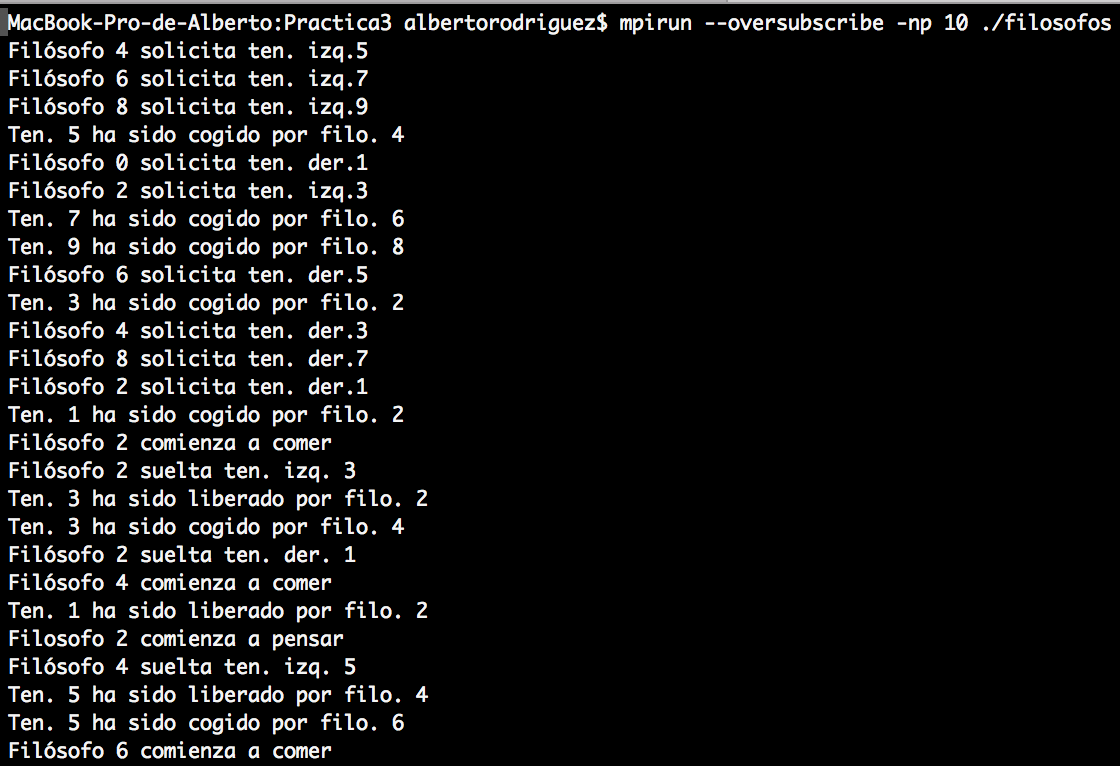
Se puede comprobar en la secuencia, que el filosofo 0 coge el tenedor izquierdo (1). Este tenedor es el derecho del filosofo 2 quien solicita coger el tenedor derecho y no puede, por lo que se produce interbloqueo.

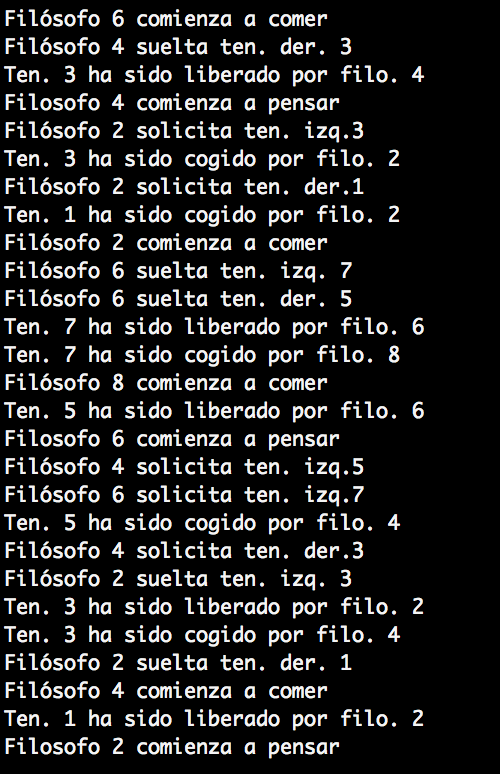
Archivo -> filosofos-interb.cpp

La solución a este interbloqueo la he implementado en el archivo filósofos.cpp. La solución ha sido que uno de los filósofos debe comenzar a coger por la derecha, en vez de por la izquierda.



Salida del programa sin interbloquedo:





**Filósofos – Camarero**

Para este problema, he copiado el código donde se producía interbloqueo (filósofos-interb.cpp) en filósofos-cam.cpp

La solución para el problema de interbloqueo ha sido introducir dos nuevos pasos a los filósofos, que es sentarse a la mesa y levantarse. El nuevo proceso camarero controlará los filósofos que se sientan a la mesa con un contador (filósofos\_sentados) que se incrementara cuando haya menos de 5 filósofos sentadas y decrementará cuando se levante un filósofo.

Para saber si se puede que levantar o sentar un filosofo he creado dos nuevas etiquetas (etiq\_sentarse = 0 y etiq\_levantarse = 1) que utilizo para pedir permiso con el envio síncrono:

MPI\_Ssend( &peticion, 1, MPI\_INT, id\_camarero, etiq\_sentarse, MPI\_COMM\_WORLD );

MPI\_Ssend( &peticion, 1, MPI\_INT, id\_camarero, etiq\_levantarse, MPI\_COMM\_WORLD );

El camarero siempre aceptará las peticiones de un filosofo para levantarse.

Otra modificación ha sido el numero de procesos, he creado dos variables, una para los procesos efectivos y otra para los procesos esperados (11). Tambien he añadido el identificador del proceso camarero (id\_camarero=num\_procesos\_efectivos).

A continuación muestro un listado parcial de la salida del programa ejecutado con 11 procesos

mpirun –oversubscribe –np 11 ./filósofos-cam

