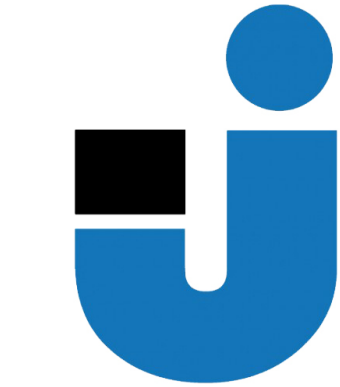


Universidad Nacional
ARTURO JAURETCHE

Sistemas Distribuidos

Laboratorio nº 1

Paralelización MPI

Autor
Emiliano Salvatori

8 de diciembre de 2020

Índice

1. Introducción

En el siguiente informe se detalla lo realizado como parte del laboratorio de la materia **Sistemas Distribuidos** para la **Comisión nº 1**.

La implementación realizada para este trabajo se basa en los siguientes objetivos:

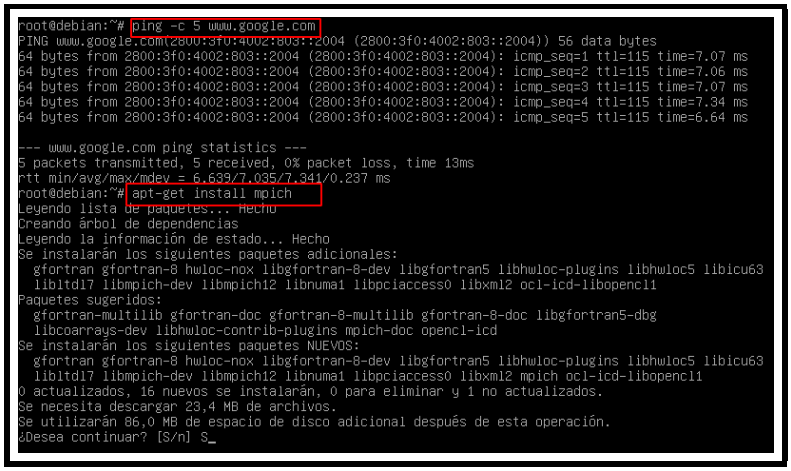
- Configuración de 3 máquinas virtuales bajo entorno GNU/Linux.
- Configuración de un cluster mediante SSH/NFS
- Entendimiento básico del pasajeo de mensajes mediante MPI
- Sincronización bajo Sistemas Distribuidos NTP
- Paralelización de código para calcular el resultado de la multiplicación de matrices.

2. Configuración de las máquinas virtuales

2.1. Instalación las herramientas a utilizar

Para poder realizar la construcción de un cluster, se procede a configurar tres máquinas virtuales bajo VirtualBox, bajo la disposición de un Nodo Maestro y dos Nodos Esclavos.

Para ello comprobamos que tenemos conexión a internet haciendo uso de la herramienta *ping*. Una vez confirmada la conexión a internet se procede a instalar los programas necesarios:



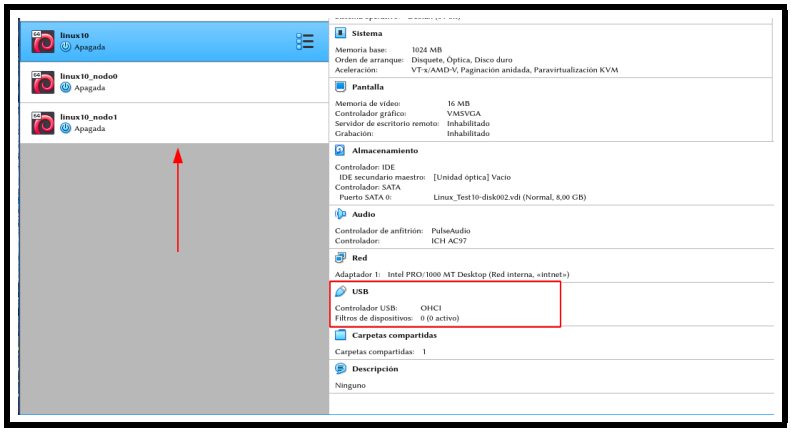
Los programas que se instalan son:

1. mpich
2. uml-utilities

3. bridge-utils
4. openssh-server
5. nfs-kernel-server
6. nfs-common
7. net-tools
8. vim : útil para poder suplantar a los editores *nano* y *vi* que vienen por defecto.

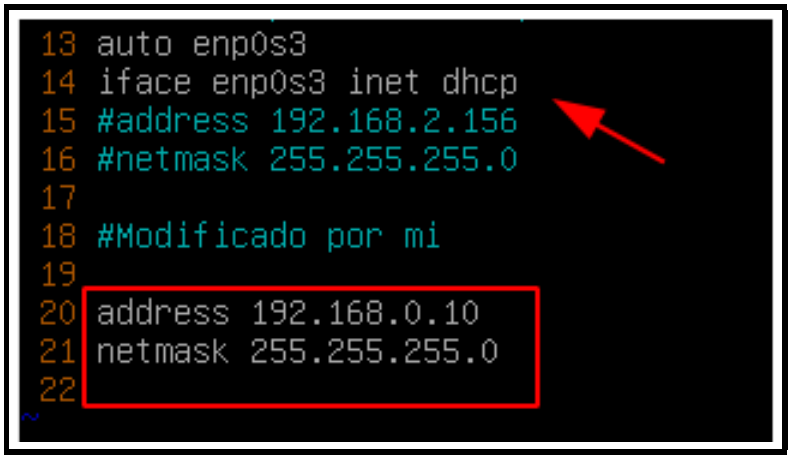
2.2. Clonación de la máquina Maestro

Luego de instalados los programas listados, se procede a clonar la máquina virtual:



Se puede observar que se tiene una carpeta compartida, la cual será necesaria para poder editar el código en la máquina host, y poder de esta manera intercambiar los archivos que querramos con mayor facilidad.

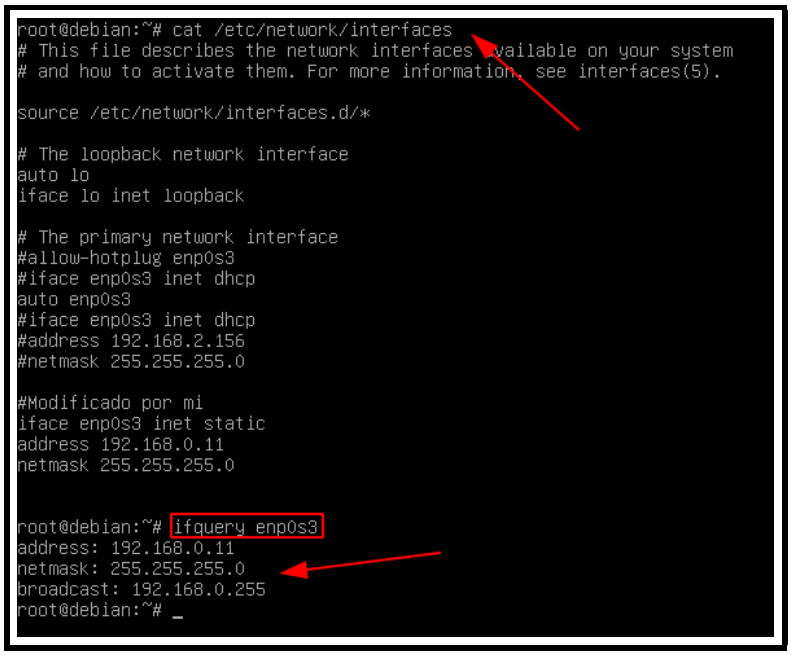
2.3. Configuración de las direcciones IP estáticas



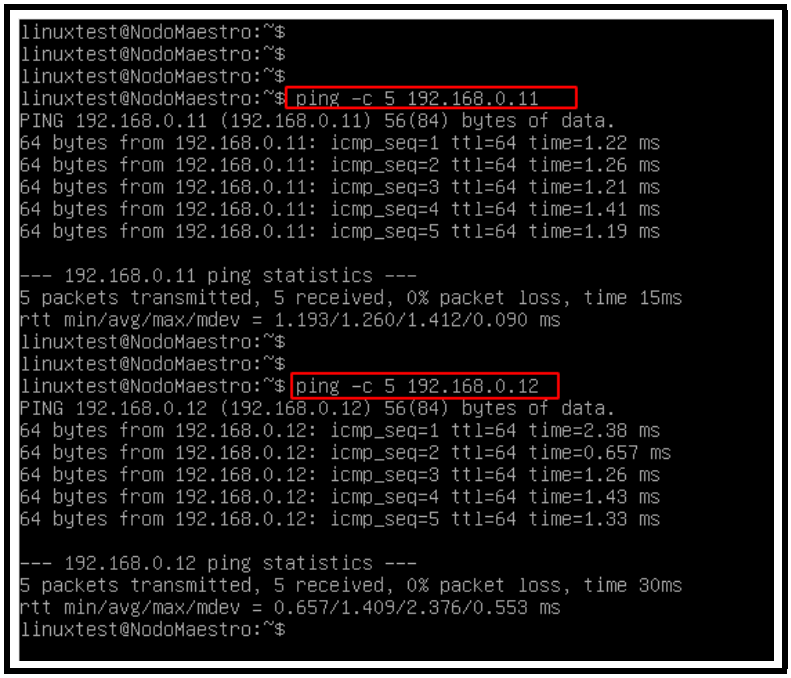
Se procede a configurar para cada máquina las direcciones IP, para ello se utiliza el editor *vim*, asignándosele las siguientes direcciones a cada una de las máquinas:

- **Maestro**: 192.168.0.10
- **Esclavo nº 1**: 192.168.0.11
- **Esclavo nº 2**: 192.168.0.12

Como ejemplo, vemos en la máquina Esclavo nº 1, la configuración correspondiente:



Como se puede visualizar, desde el Nodo Maestro realizamos un ping a las dos direcciones de los nodos Esclavos:



2.4. Configuración mediante la herramienta SSH

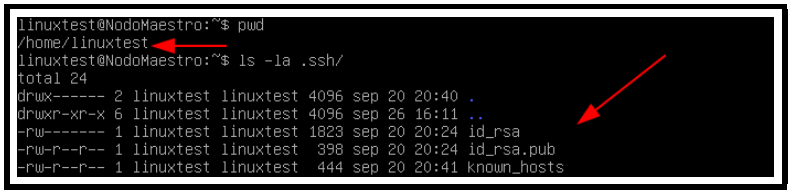
Para que desde el Maestro se pueda acceder a los distintos nodos de forma remota y sin tener que ingresar la contraseña del nodo accedido se hace uso del paquete *ssh*.

Para ello primero se crea una carpeta de nombre *share* en el directorio */home/linuxtest* del Mestro, haciendo esto también en cada nodo.

Luego se procede a ejecutar:

```
$ ssh-keygen -t rsa -b 2048
```

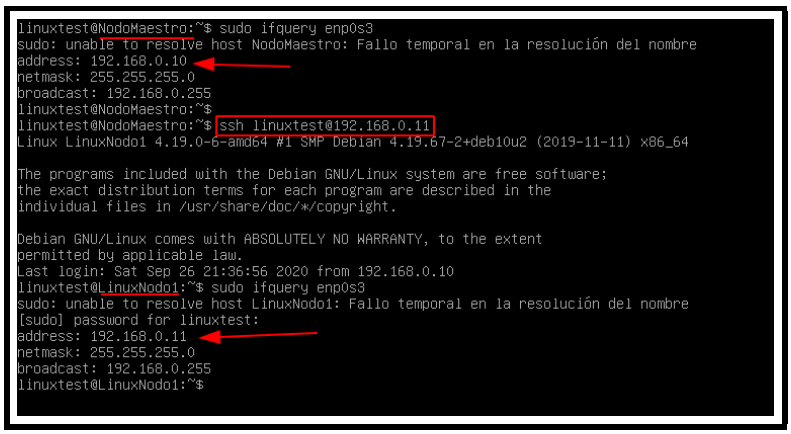
Esto hace posible que en el nodo Maestro se generen las llaves públicas y privadas a utilizar. A continuación podemos observar las llaves ya generadas dentro de la carpeta:



Copiamos en cada directorio *.ssh* de los nodos Esclavos la llave pública del Maestro de la siguiente manera:

```
$ scp id_rsa.pub linuxtest@192.168.0.11:~/.ssh/id_rsa.pub
$ scp id_rsa.pub linuxtest@192.168.0.12:~/.ssh/id_rsa.pub
```

Ahora podremos observar que se puede ingresar desde el Maestro al **Nodo Esclavo nº 1**:



Como también se puede ingresar desde el Maestro al **Nodo Esclavo nº 2**:

2.5. Configuración de los nombres de los host

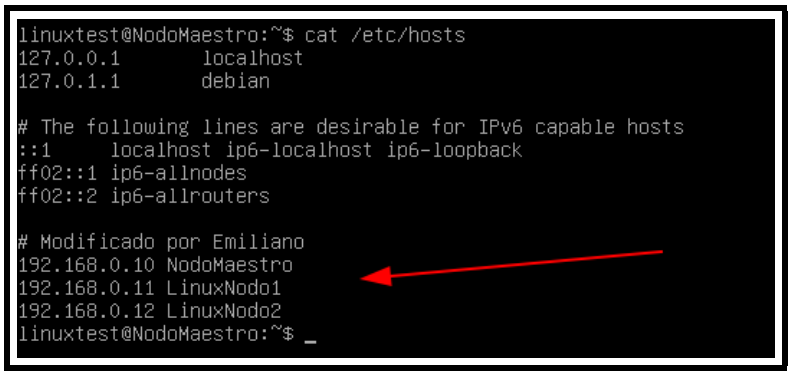
En cada uno de los nodos, será necesario modificar sus direcciones IP estáticas por un pseudónimo que apunte a sus nombres de host. Primero, en cada uno de los nodos, será necesario ejecutar el siguiente comando:

```
$ sudo hostnamectl set-hostname <NOMBRE_NODO>
```

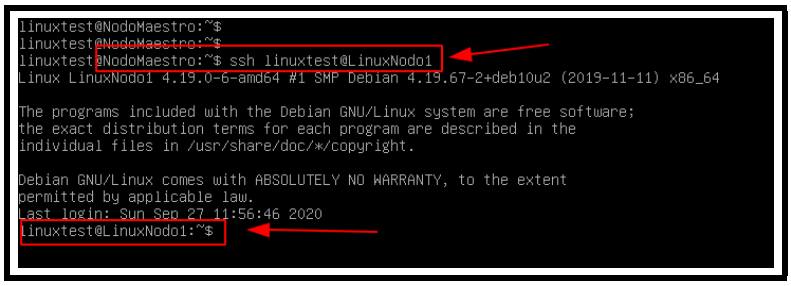
Donde *NOMBRE_NODO* será el nombre que nosotros le querramos dar a cada máquina. Se optó por los siguientes nombres:

- **NodoMaestro**: Será el Nodo Maestro que controlará los otros dos nodos Esclavos
- **LinuxNodo1**: Nodo Esclavo nº 1
- **LinuxNodo2**: Nodo Esclavo nº 1

Una vez modificado cada uno de los nombres de los hosts, lo siguiente será editar en los tres nodos el archivo */etc/hosts* para poder relacionar cada dirección con las máquinas que forman el cluster:



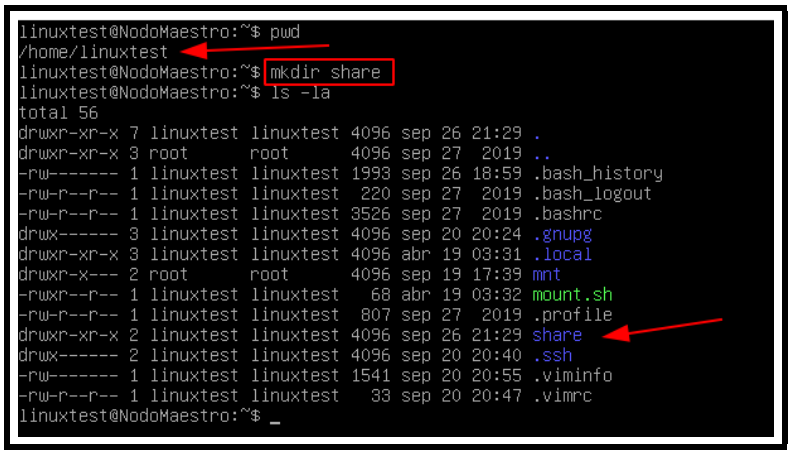
Ahora podemos observar que con la nueva configuración es posible conectarse a los distintos nodos ingresando solamente el nombre del host:



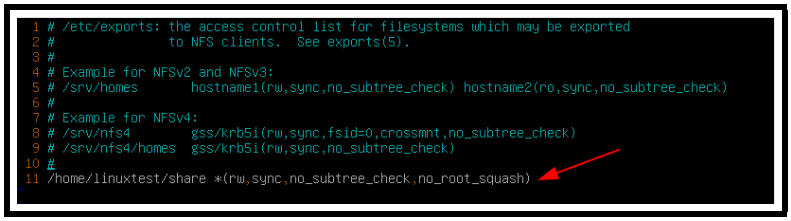
2.6. Configuración NFS

Mediante la herramienta NFS es posible tener disponible una carpeta mediante la cual se compartirá el código entre las máquinas que conforman el Cluster, para que luego sea ejecutado por los distintos nodos.

Para llevar a cabo esto, primero es necesario crear una carpeta con el nombre *share* en la ubicación */home/linuxtest*; y luego editar el archivo denominado *exports*.



Este archivo permite indicarle al SO el recurso que será accedido por los distintos nodos, proporcionando información sobre quienes podrán acceder, el nivel de permisos que tendrán los nodos (como leer y escribir), si el servidor responderá a peticiones cuando se complete la tarea del disco, y además se debe indicar el nivel de acceso del superusuario (root) de los clientes al sistema de archivos.



Y luego utilizamos el siguiente comando:

```
$ sudo exportfs -a
```

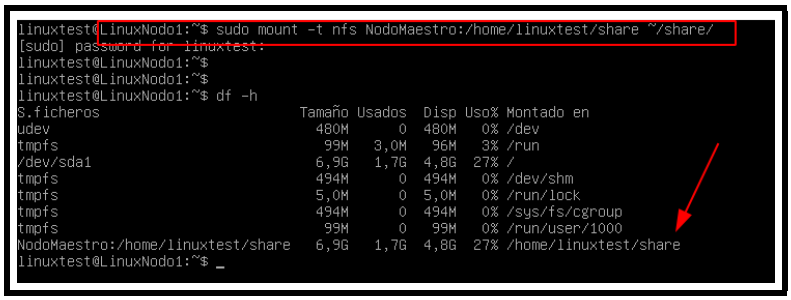
Una vez que tenemos los esto procedemos dentro de cada uno de los nodos Esclavos, a montar la carpeta compartida con el siguiente comando:

```
$ sudo mount -t nfs NodoMaestro:/home/linuxtest/share
/share
```

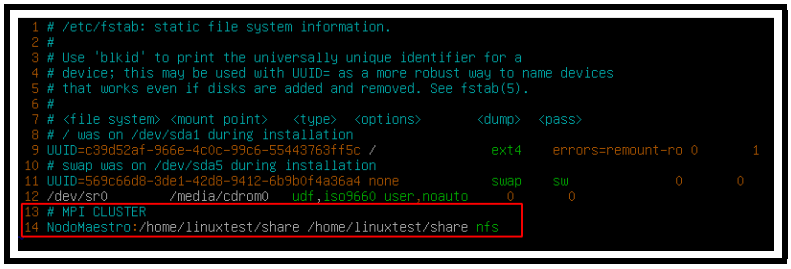
Y Nos fijamos si se montó de forma correcta mediante:

```
$ df -h
```

Efectivamente, observamos que se monta de forma correcta:



Como nos aseguramos que montó correctamente, entonces ahora procedemos a generar una entrada en el archivo */etc/fstab* para que se monte de forma automática cada vez que se inicien los nodos:

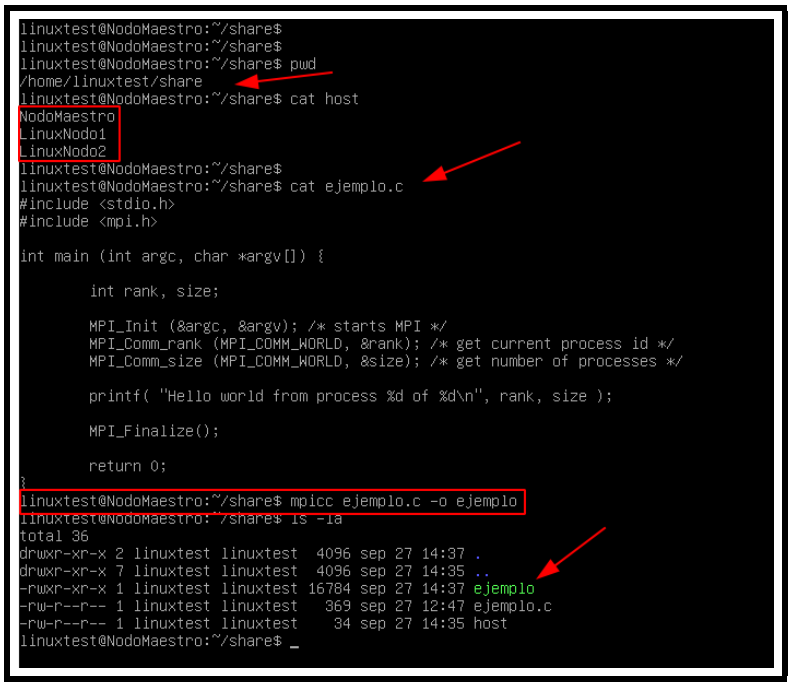


3. Testeo sobre la librería MPI

Ahora que ya se encuentra configurado correctamente el cluster, procedemos a realizar un ejemplo con un código sencillo para saber si es posible paralelizar el código compilado.

Para esto primero editamos un archivo que contenga los nombres de los host que estaremos utilizando para paralelizar el código y luego compilamos el código utilizando el siguiente comando:

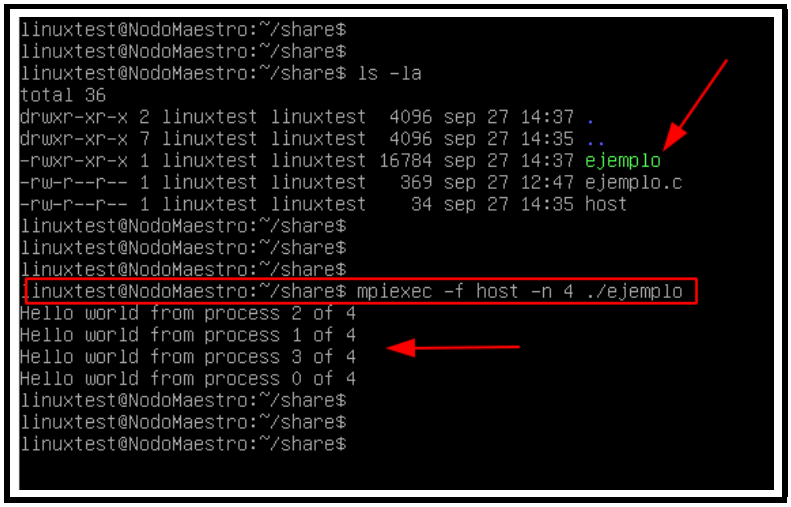
```
$ mpicc ejemplo.c -o ejemplo
```



Como podemos observar en la imagen anterior, la compilación nos genera un ejecutable denominado “ejemplo”, el cual ejecutamos

```
$ mpiexec -f host -n 4 ./ejemplo
```

Luego de ello se obtiene:



4. Paralelización para el producto entre matrices

Para la implementación de las librerías MPI, se utilizará el cálculo de multiplicación entre matrices. Para poder paralelizar la ejecución y poder distribuir la carga de cálculos entre distintos nodos, se utilizá la librería `mpi.h`, junto con la configuración desarrollada en la sección 3.

Para que los nodos tengan acceso al ejecutable, es necesario que sea suministrado en una dirección donde todo el conjunto de máquinas que conforman el cluster, puedan acceder a el. Para ello se utiliza la dirección configurada en el apartado 2.6

4.1. Consideraciones generales

Para la ejecución del producto entre matrices, se deben tener en cuenta algunas consideraciones especiales del programa:

1. Las dimensiones de las matrices son estáticas; es decir, que los elementos de la **matriz A** serán determinados en el código, sin posibilidad de ser modificadas en tiempo de ejecución. Esto es así ya que de lo contrario, sería necesario utilizar las funciones `malloc()` y `calloc()` y leer las entradas desde el teclado, implementaciones que excenden al alcance del trabajo actual.

   Lo mismo sucede con la cantidad de elementos de la **matriz B**.

2. El producto entre matrices se hace entre una **matriz A** cuyas dimensiones pueden ser modificadas en el código (es decir que la cantidad de elementos de sus filas y columnas pueden variar); y una **matriz B** con una cantidad de elementos por columna también alterable por el usuario, pero con sus filas **igual a las Columnas de A**.

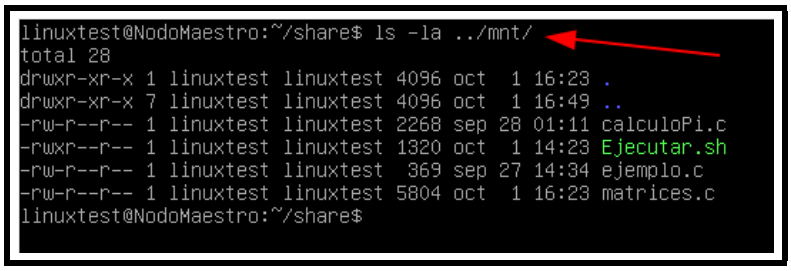
   Esto se implementó de esta manera ya que de lo contrario, habría que manejar el ingreso de datos desde el teclado, labor que excedía al tiempo estimado para la entrega del trabajo. De esta forma, siempre será posible el cálculo del producto entre las matrices que se busquen.

3. La inicialización de los elementos de las matrices se realiza mediante una estructura de control, con un `for`, cuyos contadores comienzan en 0 y se incrementan de uno a 1 por cantidad de columnas en A, y en Filas por B.

4.2. Ejecución del código

Para editar el código se utilizó el editor *vim* en un SO anfitrión de tipo GNU/Linux. El archivo se colocó en la carpeta compartida entre el SO anfitrión y el SO huésped (Debian 10), como se puede ver a continuación en la siguiente imagen:

Se puede observar, que el contenido es el mismo en la carpeta compartida de Debian:

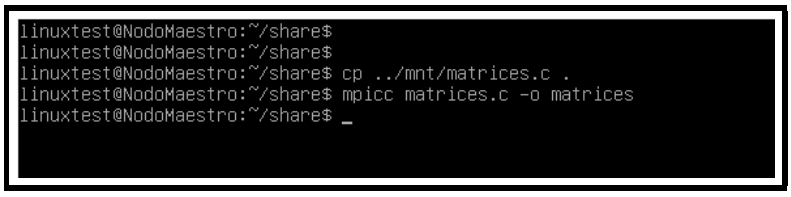


Ahora procedemos a ingresar el siguiente comando para copiar desde la carpeta compartida (`mnt`) a la compartida entre el cluster (`share`):

```
$ cp ../mnt/matrices.c
```

Una vez copiada a la carpeta `share` lo compilamos mediante el siguiente comando:

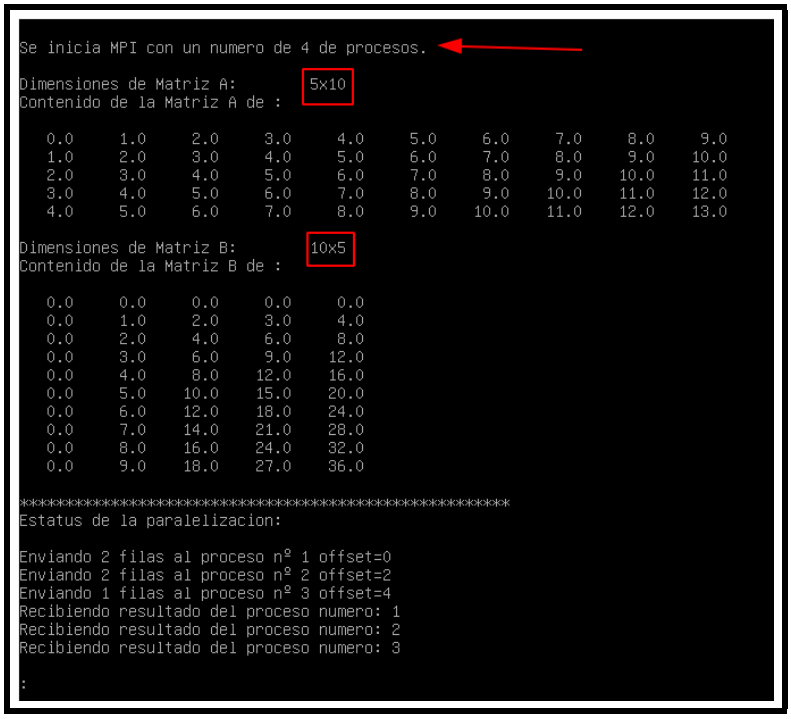
```
$ mpicc matrices.c -o matrices
```

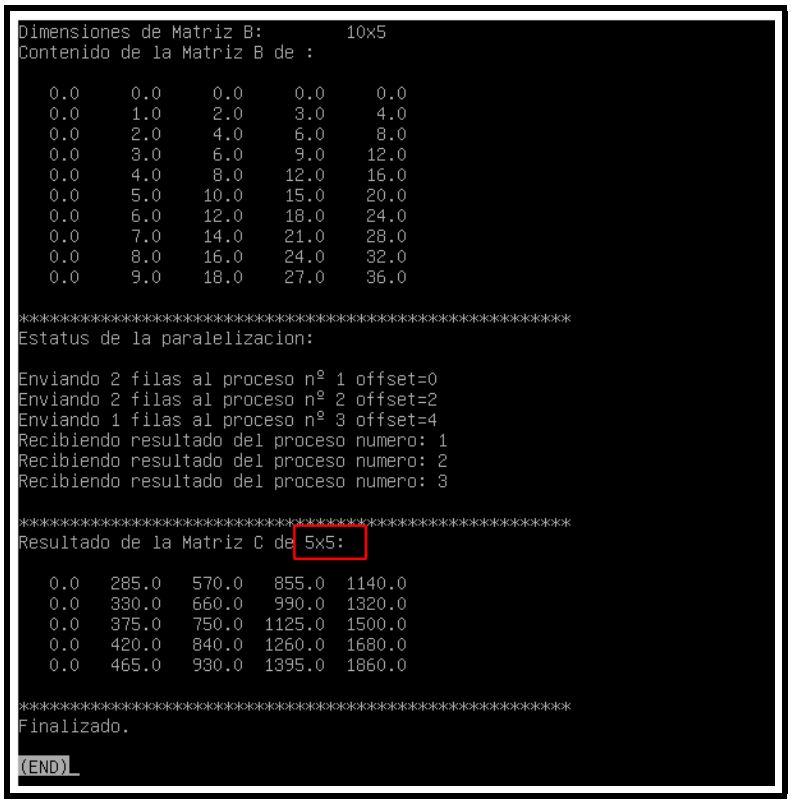


Vemos que compiló sin ningún tipo de problemas. Procedemos luego a ejecutarlo indicando que se repartira la carga de cálculos entre los nodos que se encuentran en el archivo `host` y que se requiere paralelizar entre 4 procesos. Cabe resaltar que se utiliza el programa `less` el cual permita *paginar* la terminal, visualizando todo el contenido volcado:

```
$ mpiexec -f host -n 4 ./matrices | less
```

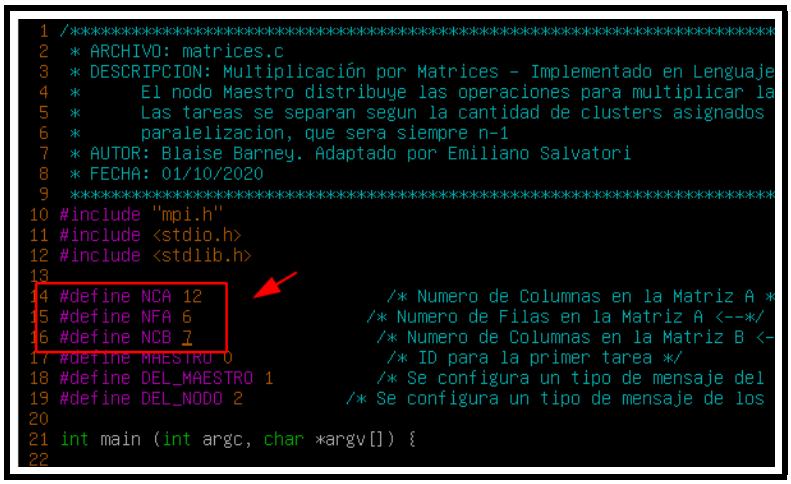
Visualizando la ejecución del proceso de forma correcta:





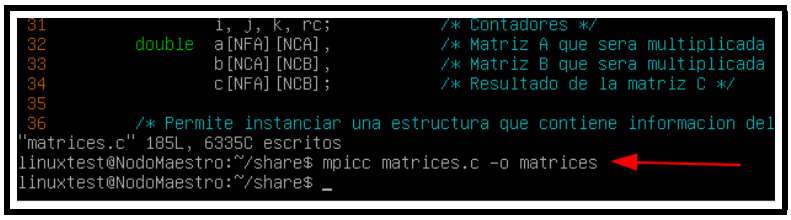
4.3. Modificación de las Dimensiones de las matrices

Ahora procedemos a realizar la modificación de las dimensiones de A y de B. Para ello editamos directamente en el Nodo Maestro, el archivo *matrices.c*, y modificamos el valor de NCA (número de columnas de A) y de NFA (número de filas de A), llevándola a una dimensión de $D_A = 12X6$; modificamos también el valor de NCB (número de columnas de B), llevando a la matriz a una dimensión de $D_B = 7X12$:



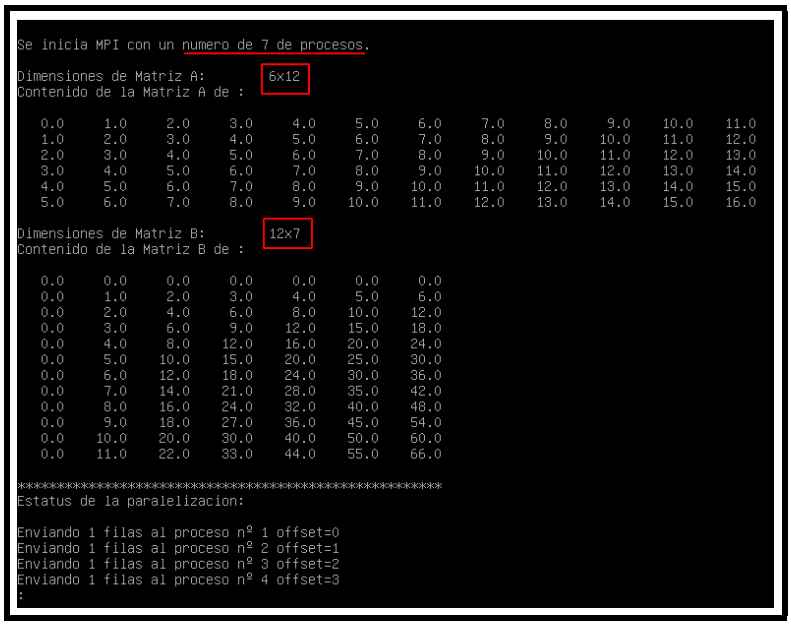
Nuevamente procedemos a compilar el archivo con el siguiente comando:

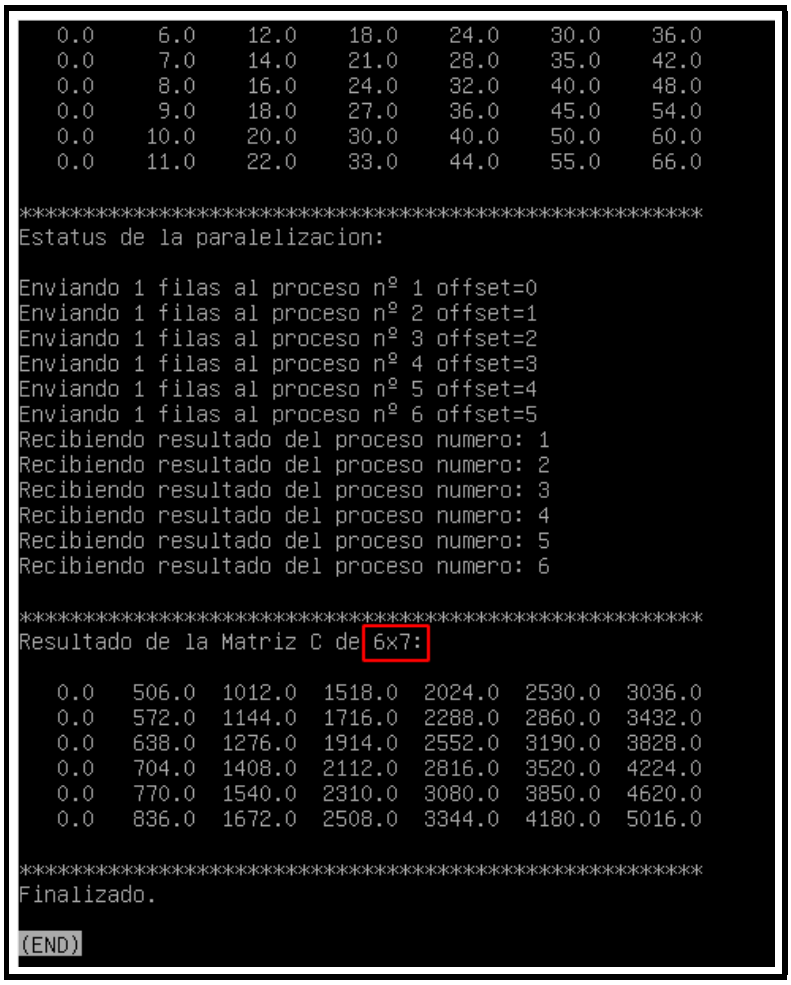
```
$ mpicc matrices.c -o matrices
```



Vemos que compiló sin ningún tipo de problemas. Procedemos luego a ejecutarlo indicando que se repartira la carga de cálculos entre los nodos que se encuentran en el archivo `host` y que se requiere paralelizar entre 7 procesos. Nuevamente, como toda la pantalla de la terminal no alcanza para visualizar la salida de los proceso, entonces se procede a utilizar la herramienta `less`:

```
$ mpiexec -f host -n 7 ./matrices | less
```


Se inicia MPI con un numero de 7 de procesos.
Dimensiones de Matriz A: 6x12
Contenido de la Matriz A de :
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0
3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0
4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0
5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0
Dimensiones de Matriz B: 12x7
Contenido de la Matriz B de :
0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 2.0 3.0 4.0 5.0 6.0
0.0 2.0 4.0 6.0 8.0 10.0 12.0
0.0 3.0 6.0 9.0 12.0 15.0 18.0
0.0 4.0 8.0 12.0 16.0 20.0 24.0
0.0 5.0 10.0 15.0 20.0 25.0 30.0
0.0 6.0 12.0 18.0 24.0 30.0 36.0
0.0 7.0 14.0 21.0 28.0 35.0 42.0
0.0 8.0 16.0 24.0 32.0 40.0 48.0
0.0 9.0 18.0 27.0 36.0 45.0 54.0
0.0 10.0 20.0 30.0 40.0 50.0 60.0
0.0 11.0 22.0 33.0 44.0 55.0 66.0
****
Estatus de la paralelizacion:
Enviando 1 filas al proceso nº 1 offset=0
Enviando 1 filas al proceso nº 2 offset=1
Enviando 1 filas al proceso nº 3 offset=2
Enviando 1 filas al proceso nº 4 offset=3
:



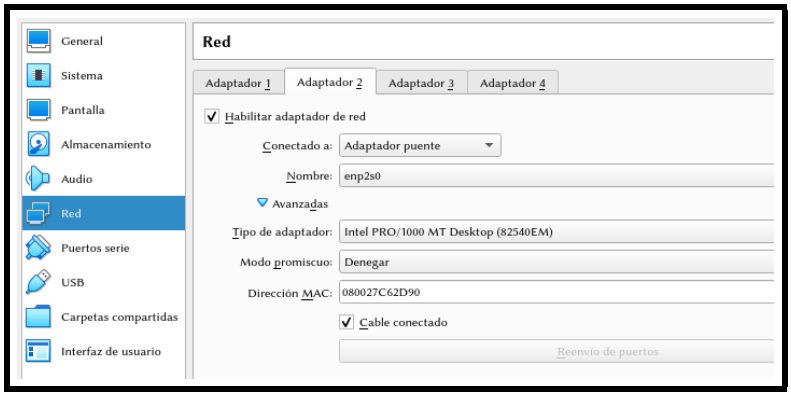
5. Implementación con NTP

5.1. Configuración inicial del sistema para NTP

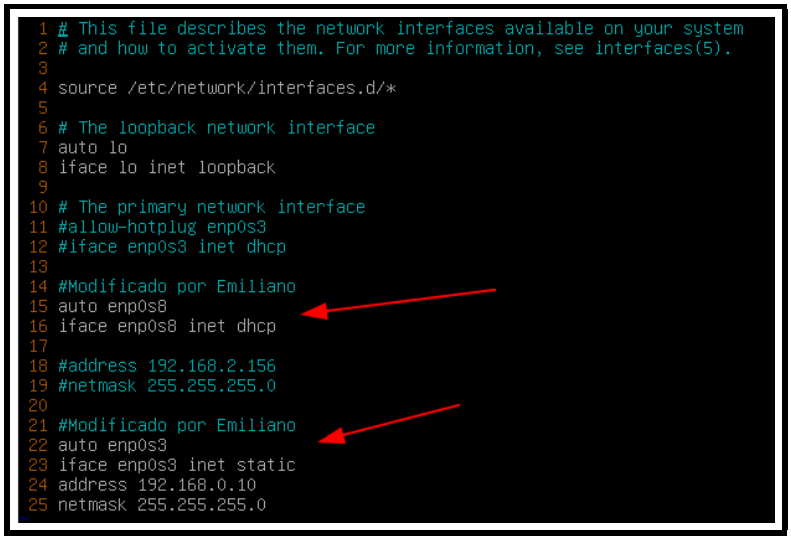
Para el siguiente ejercicio es necesario modificar los nodos del sistema para que tengan salida a internet y con ello poder instalar todos los paquetes necesarios para el correcto funcionamiento de NTP. Cabe resaltar que estas modificaciones en los nodos esclavos serán *momentáneas*. La configuración en cada uno de los nodos que conforman el sistema, consistirá en:

- Habilitar un nuevo adaptador en la configuración de Red, como *Adaptador Puente*.
- Configurar el archivo *interfaces* para que tome y configure el nuevo adaptador.

Para ello vamos a realizar el primer punto en los tres nodos, como se puede ver a continuación:



Luego de ello se configura en las tres máquinas el archivo, de la siguiente forma:



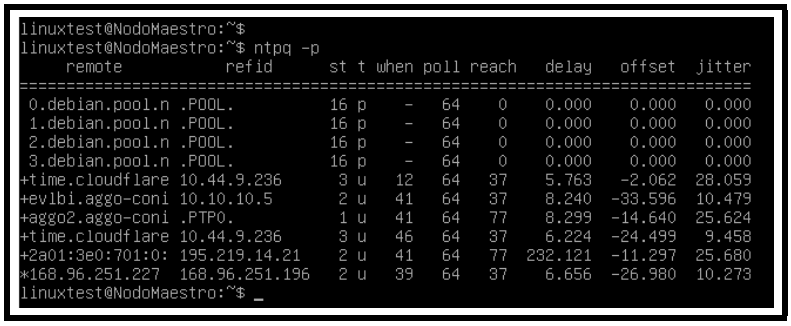
Y se procede a instalar en cada una de las máquinas, lo requerido, mediante el siguiente comando:

```
$ sudo apt-get install ntp ntp-doc
```

Se puede ver que todo funciona correctamente cuando se lanza el siguiente comando:

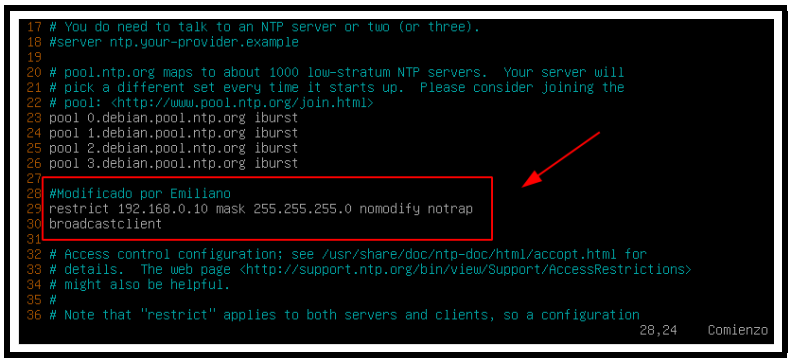
```
$ ntpq -p
```

El programa *ntpq* se utiliza para supervisar las operaciones ntpd del demonio NTP y determinar su rendimiento. Cuando se ingresa el parámetro *-q* se imprime una lista de los pares conocidos por el servidor, así como un resumen de su estado.



5.2. Modificación del archivo de configuración NTP en el Maestro

Para el correcto funcionamiento de NTP, es necesario modificar en el nodo Maestro, el archivo *ntp.conf*, para que este nodo pueda encargarse de la difusión a los demás nodos del sistema con la opción *broadcastclient*. Asimismo, con la configuración ingresada se restringe la modificación del sistema sólo al maestro. La modificación queda de la siguiente manera:



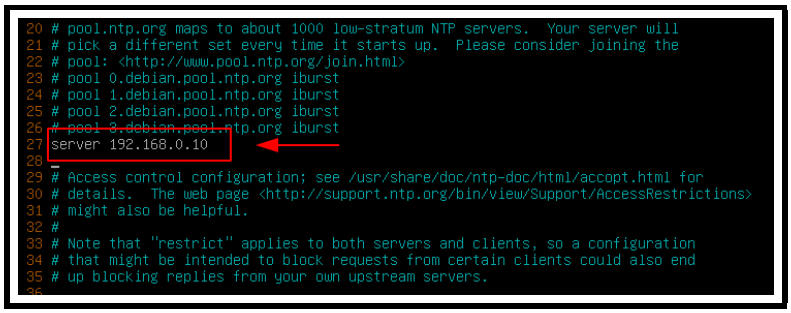
Luego de ello, se ingresa un comando para que se permita la comunicación mediante UPD por el puerto 123:

```
$ sudo iptables -A INPUT -m state --state NEW -m udp -p
udp --dport 123 -j ACCEPT
```

Si todo sale de forma correcta, se vuelve al prompt:

5.3. Modificación del archivo de configuración NTP en los Esclavos

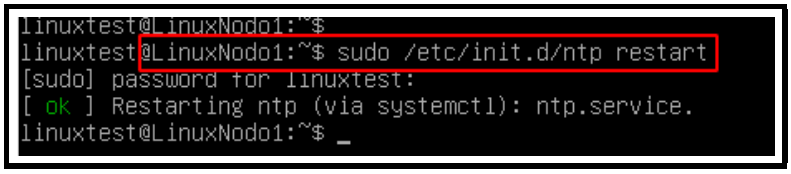
Para la configuración de los nodos Esclavos, se debe modificar también el archivo *ntp.conf* pero comentando todas las lineas de los pools, y especificando que el servidor será el nodo Maestro, quedando de la siguiente forma:



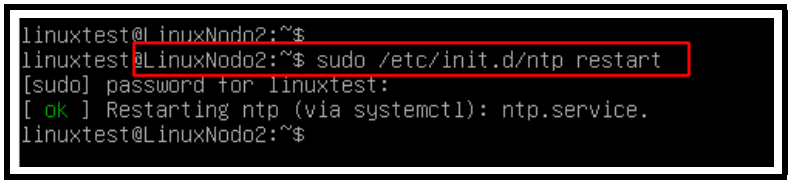
Se procede a reiniciar el servicio de NTP en cada uno de los nodos esclavos con el siguiente comando:

```
$ sudo /etc/init.d/ntp restart
```

Y se visualiza el correcto reinicio en el nodo Esclavo nº 1:



Y en el Nodo Esclavo nº 2:



A continuación visualizamos que todo esté funcionando correctamente ingresando tanto en el Nodo Esclavo nº 1 como en el nº 2 el siguiente comando:

```
$ ntpq -p
```

En el nodo Esclavo nº 1:

```
linuxtest@LinuxNodo1:~$ ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 NodoMaestro     168.96.251.197   2 u    6   64    1    1.374   -0.062   0.000
linuxtest@LinuxNodo1:~$
```

Y en el nodo Esclavo nº 2:

```
linuxtest@LinuxNodo2:~$ ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 NodoMaestro     168.96.251.197   2 u   16   64    3    1.311  318879.  35.970
linuxtest@LinuxNodo2:~$ _
```

En este punto se debe volver atrás las modificaciones realizadas en la sección 5.1, es decir que se quitan de los nodos Esclavos el segundo Adaptador de Red, quedando sólo el primer Adaptador como Red Interna y se vuelven para atrás las modificaciones establecidas en el archivo *interfaces*.

5.4. Prueba del funcionamiento distribuido de NTP

A continuación se pueden apreciar el tiempo transcurrido operando con NTP. Para ello volvemos a realizar lo comentado en la sección 4.2 y realizamos un cambio agregando más columnas y filas al código fuente:

```
 1 /*****************************************************************************
 2  * ARCHIVO: matrices.c
 3  * DESCRIPCION: Multiplicación por Matrices - Implementado en Lenguaje C
 4  *      El nodo Maestro distribuye las operaciones para multiplicar las matrices.
 5  *      Las tareas se separan segun la cantidad de clusters asignados para la
 6  *      paralelizacion, que sera siempre n-1
 7  * AUTOR: Blaise Barney. Adaptado por Emiliano Salvatori
 8  * FECHA: 01/10/2020
 9  *****************************************************************************/
10 #include "mpi.h"
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 #define NCA 12                  /* Numero de Columnas en la Matriz A */
15 #define NFA 6                  /* Numero de Filas en la Matriz A <--*/
16 #define NCB 7                   /* Numero de Columnas en la Matriz B <-- */
17 #define MAESTRO 0               /* ID para la primer tarea */
18 #define DEL_MAESTRO 1          /* Se configura un tipo de mensaje del nodo maestro*
19 #define DEL_NODO 2            /* Se configura un tipo de mensaje de los nodos*/
20
```

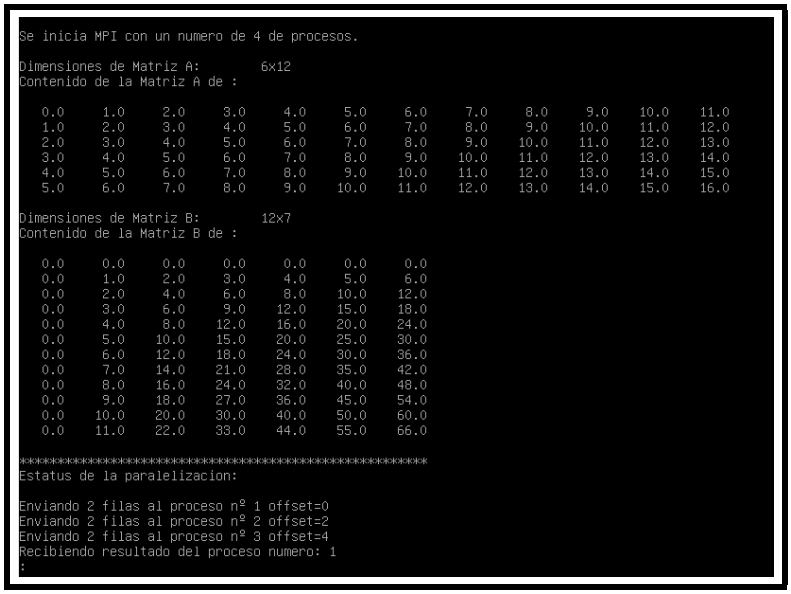
A continuación volvemos a compilar utilizando los comandos:

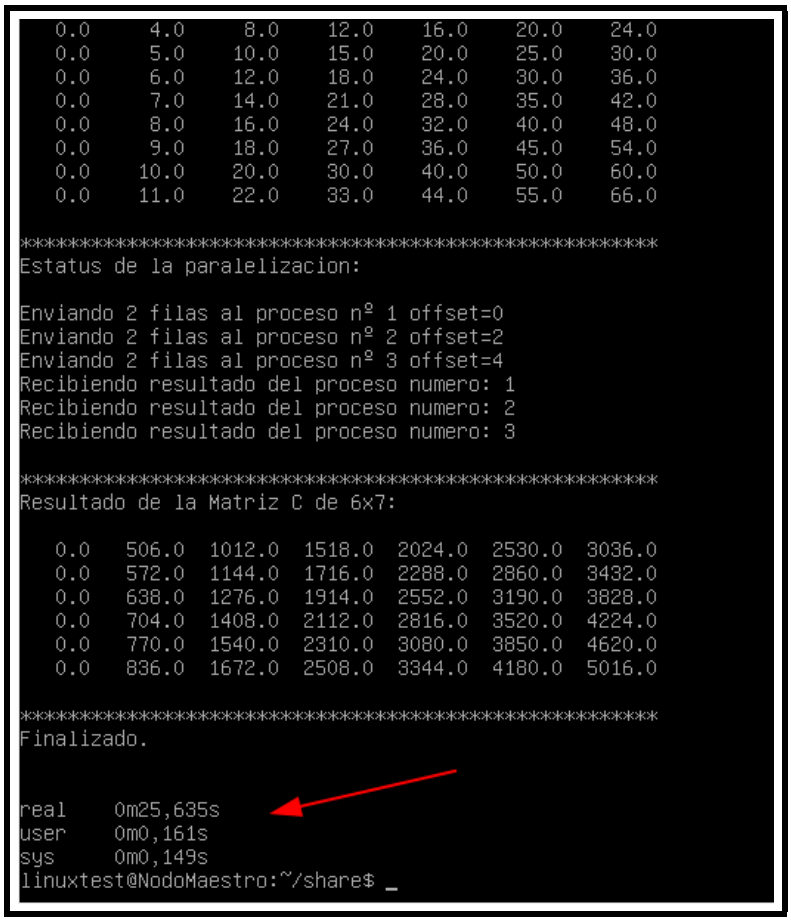
```
$ mpicc matrices.c -o matrices
```

```
linuxtest@NodoMaestro:~/share$
linuxtest@NodoMaestro:~/share$
linuxtest@NodoMaestro:~/share$ mpicc matrices.c -o matrices
linuxtest@NodoMaestro:~/share$
```

Vemos que compiló sin ningún tipo de problemas. Procedemos luego a ejecutarlo indicando que se repartira la carga de cálculos entre los nodos que se encuentran en el archivo `host` y que se requiere paralelizar entre 4, sin antes testear el tiempo utilizando para ello la herramienta *time* procesos mediante:

```
$ time mpiexec -f host -n 4 ./matrices | less
```





6. Código del programa

A continuación se visualiza el código en lenguaje C que permite obtener el producto entre dos matrices:

```c
/******************************************************************************
 * ARCHIVO: matrices.c
 * DESCRIPCION: Multiplicacion por Matrices - Implementado en Lenguaje C
 *       El nodo Maestro distribuye las operaciones para multiplicar las matrices.
 *       Las tareas se separan segun la cantidad de clusters asignados para la
 *       paralelizacion, que sera siempre n-1
 * AUTOR: Blaise Barney. Adaptado por Emiliano Salvatori
 * FECHA: 01/10/2020
 ******************************************************************************/
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define NCA 10                  /* Numero de Columnas en la Matriz A */
#define NFA 5                   /* Numero de Filas en la Matriz A <--*/
#define NCB 5                   /* Numero de Columnas en la Matriz B <-- */
#define MAESTRO 0               /* ID para la primer tarea */
#define DEL_MAESTRO 1           /* Se configura un tipo de mensaje del nodo maestro*/
#define DEL_NODO 2              /* Se configura un tipo de mensaje de los nodos*/

int main (int argc, char *argv[]) {

        int     numtareas,              /* Numero de tareas en ejecucion */
```

```
24                  taskid,                 /* Identificador de las tareas */
25                  numprocesos,            /* Numero de procesadores que tomaran la tarea*/
26                  origen,                 /* Id de la tarea segun el mensaje de origen */
27                  dest,                   /* Id de la tarea segun el mensaje destino */
28                  menstipo,               /* Tipo de mensaje */
29                  filas,                  /* Numero de filas de A enviadas a cada nodo */
30                  averow, extra, offset,  /* Determina la cantidad de filas para cada nodo */
31                  i, j, k, rc;            /* Contadores */
32          double  a[NFA][NCA],            /* Matriz A que sera multiplicada */
33                  b[NCA][NCB],            /* Matriz B que sera multiplicada */
34                  c[NFA][NCB];            /* Resultado de la matriz C */
35
36          /* Permite instanciar una estructura que contiene informacion del mensaje enviado */
37          MPI_Status status;
38
39          /* Se inicializa el entorno de ejecucion para MPI. */
40          MPI_Init(&argc,&argv);
41
42          /* Determina el rango del proceso de llamada en el controlador de seniales
43          INPUT: recibe el controlador de seniales
44          OUTPUT: el rango del proceso de llamada el el grupo del controlador de seniales */
45          MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
46
47
48          /* Determina el tamanio del grupo de los nodos asociados con el programa main
49          INPUT: Recibe el parametro de entrada que vendria a ser el controlador de seniales
50          OUTPUT: numero de los proceso en el grupo del controlador de seniales */
51          MPI_Comm_size(MPI_COMM_WORLD,&numtareas);
52
53          /* En caso que el numero de procesadores a ejecutar sea menor a 2, entonces abortar */
54          if (numtareas < 2 ) {
55                  printf("Se requieren al menos 2 procesos para ejecutar el programa. Abortando...\n");
56                  MPI_Abort(MPI_COMM_WORLD, rc);
57                  exit(1);
58          }
59
60          numprocesos = numtareas-1;
61
62          /************************** Proceso Maestro *********************************/
63
64          if (taskid == MAESTRO) {
65
66                  printf("\n");
67                  printf("Se inicia MPI con un numero de %d de procesos.\n",numtareas);
68
69                  for (i=0; i<NFA; i++)
70                          for (j=0; j<NCA; j++)
71                                  a[i][j]= i+j;
72                  for (i=0; i<NCA; i++)
73                          for (j=0; j<NCB; j++)
74                                  b[i][j]= i*j;
75
76
77                  /* Se tratan de imprimir los datos por pantalla de las matrices */
78
79                  printf("\n");
80                  printf("Dimensiones de Matriz A: \t %dx %d\n",NFA, NCA);
81                  printf("Contenido de la Matriz A de :\n");
82
83                  for (i=0; i<NFA; i++){
84                          printf("\n");
85                          for (j=0; j<NCA; j++){
86                                  a[i][j]= i+j;
87                                  printf("%6.1f\t", a[i][j]);
88                          }
89                  }
90
91                  printf("\n");
92                  printf("\n");
93                  printf("Dimensiones de Matriz B: \t %dx %d\n",NCA, NCB);
94                  printf("Contenido de la Matriz B de :\n");
95
96                  for (i=0; i<NCA; i++){
97                          printf("\n");
98                          for (j=0; j<NCB; j++){
99                                  b[i][j]= i*j;
100                                 printf("%6.1f\t", b[i][j]);
101                         }
102                 }
103
104                 printf("\n");
105                 printf("\n");
106                 printf("*****************************************************\n");
107                 printf("Estatus de la paralelizacion: \n");
108
109                 /* Se envia los datos de la matriz a los nodos */
110
111                 averow = NFA/numprocesos;
112                 extra = NFA%numprocesos;
113                 offset = 0;
114                 menstipo = DEL_MAESTRO;
```

```
115
116                 printf("\n");
117                 for (dest=1; dest<=numprocesos; dest++) {
118
119                         filas = (dest <= extra) ? averow+1 : averow;
120                         printf("Enviando %d filas al proceso nÂº %d offset=%d\n",filas,dest,offset);
121                         MPI_Send(&offset, 1, MPI_INT, dest, menstipo, MPI_COMM_WORLD);
122                         MPI_Send(&filas, 1, MPI_INT, dest, menstipo, MPI_COMM_WORLD);
123                         MPI_Send(&a[offset][0], filas*NCA, MPI_DOUBLE, dest, menstipo,
124                                         MPI_COMM_WORLD);
125                         MPI_Send(&b, NCA*NCB, MPI_DOUBLE, dest, menstipo, MPI_COMM_WORLD);
126                         offset = offset + filas;
127                 }
128
129                 /* Se recibe los resultados de los nodos */
130
131                 menstipo = DEL_NODO;
132                 for (i=1; i<=numprocesos; i++) {
133
134                         origen = i;
135                         MPI_Recv(&offset, 1, MPI_INT, origen, menstipo, MPI_COMM_WORLD, &status);
136                         MPI_Recv(&filas, 1, MPI_INT, origen, menstipo, MPI_COMM_WORLD, &status);
137                         MPI_Recv(&c[offset][0], filas*NCB, MPI_DOUBLE, origen, menstipo,
138                                         MPI_COMM_WORLD, &status);
139                         printf("Recibiendo resultado del proceso numero: %d\n",origen);
140                 }
141
142                 /* Se imprimen resultados */
143                 printf("\n");
144                 printf("*******************************************************\n");
145                 printf("Resultado de la Matriz C de %dx%d:\n",NFA, NCB);
146
147                 for (i=0; i<NFA; i++) {
148                         printf("\n");
149                         for (j=0; j<NCB; j++)
150                                 /* printf("%6.2f\t", c[i][j]); */
151                                 printf("%6.1f\t", c[i][j]);
152                 }
153
154                 printf("\n");
155                 printf("\n*******************************************************\n");
156                 printf ("Finalizado.\n");
157                 printf("\n");
158         }
159
160         /************************** Nodos Esclavos ********************************/
161
162         if (taskid > MAESTRO) {
163
164                 menstipo = DEL_MAESTRO;
165                 MPI_Recv(&offset, 1, MPI_INT, MAESTRO, menstipo, MPI_COMM_WORLD, &status);
166                 MPI_Recv(&filas, 1, MPI_INT, MAESTRO, menstipo, MPI_COMM_WORLD, &status);
167                 MPI_Recv(&a, filas*NCA, MPI_DOUBLE, MAESTRO, menstipo, MPI_COMM_WORLD, &status);
168                 MPI_Recv(&b, NCA*NCB, MPI_DOUBLE, MAESTRO, menstipo, MPI_COMM_WORLD, &status);
169
170                 for (k=0; k<NCB; k++)
171                         for (i=0; i<filas; i++) {
172                                 c[i][k] = 0.0;
173                                 for (j=0; j<NCA; j++)
174                                         c[i][k] = c[i][k] + a[i][j] * b[j][k];
175                         }
176
177                 menstipo = DEL_NODO;
178
179                 MPI_Send(&offset, 1, MPI_INT, MAESTRO, menstipo, MPI_COMM_WORLD);
180                 MPI_Send(&filas, 1, MPI_INT, MAESTRO, menstipo, MPI_COMM_WORLD);
181                 MPI_Send(&c, filas*NCB, MPI_DOUBLE, MAESTRO, menstipo, MPI_COMM_WORLD);
182         }
183
184         /* Finaliza la entorno de ejecucion MPI */
185         MPI_Finalize();
186 }
```

7. Conclusiones

Con este Trabajo, se pudo adentrar en las prácticas de los sistemas operativos de tipo GNU/Linux. Asimismo se pudo comprender de una forma más cercana el funcionamiento de programas cuya ejecución se realiza de forma distribuida empleando las librerías MPI.

Luego de esto, se pudo comprobar la mejora de tiempos con el uso del protocolo

NTP, el cual es un protocolo basado en un sistema cliente-servidor; el mismo provee a los clientes con tres productos fundamentales: clock offset, round-trip delay y referencia de dispersión: el Offset especifica la diferencia entre la hora del sistema local y la referencia externa de reloj; el Round-trip delay especifica las latencias de tiempo medidas durante la transferencias de paquetes dentro de la red; y la Referencia de dispersión de tiempo especifica el máximo número de errores asociados con la información de tiempo recibido de un reloj externo.

Cabe destacar que las configuraciones realizadas para establecer una correcta comunicación entre todo el sistema (tanto para la sección 2.4 como para la 5) hizo posible que se comprendiera de mejor forma el funcionamiento del File System de Linux.