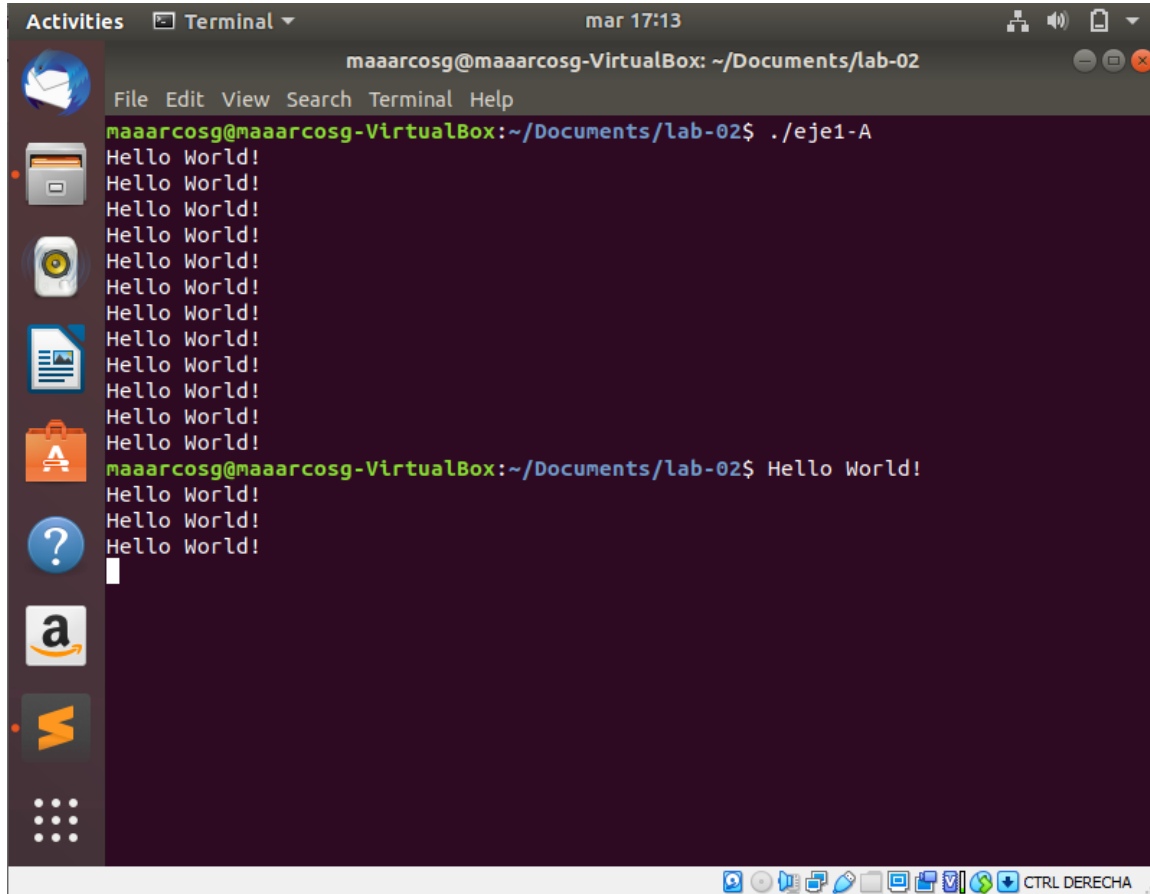


## Laboratorio No.2

### Ejercicio No. 1



```
Activities Terminal mar 17:13
maaaarcosg@maaaarcosg-VirtualBox: ~/Documents/lab-02
File Edit View Search Terminal Help
maaaarcosg@maaaarcosg-VirtualBox:~/Documents/lab-02$ ./eje1-A
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
maaaarcosg@maaaarcosg-VirtualBox:~/Documents/lab-02$ Hello World!
Hello World!
Hello World!
Hello World!
```

- **¿Cuántos procesos se crean en cada uno de los programas?**  
En ambos programas se ejecutan 16 procesos.
- **¿Por qué hay tantos procesos en ambos programas cuando uno tiene cuatro llamadas `Fork()` y el otro sólo tiene una?**  
En ambos programas se ejecutan 4 veces `fork()`, la única diferencia es la sintaxis del código, ya que `for` es una abreviación de código. El código del ejercicio **ejercicio-01A** los `fork` se ejecutan de manera explícita.

### Ejercicio No. 2

Ejercicio A:

```
maaaarcosg@maaaarcosg-VirtualBox:~/Documents/lab-02$ ./eje-02A
69620.000000
```

## Ejercicio B:

```
maaacosg@maaacosg-VirtualBox: ~/Process-lab02
File Edit View Search Terminal Help
maaacosg@maaacosg-VirtualBox:~/Process-lab02$ gcc -o eje-02B.o ejercicio-02B.c
maaacosg@maaacosg-VirtualBox:~/Process-lab02$ ./eje-02B.o
307.000000
-998.000000
-1012.000000
-1095.000000
-1087.000000
maaacosg@maaacosg-VirtualBox:~/Process-lab02$ -1088.000000
4447874.000000
4582723.000000
```

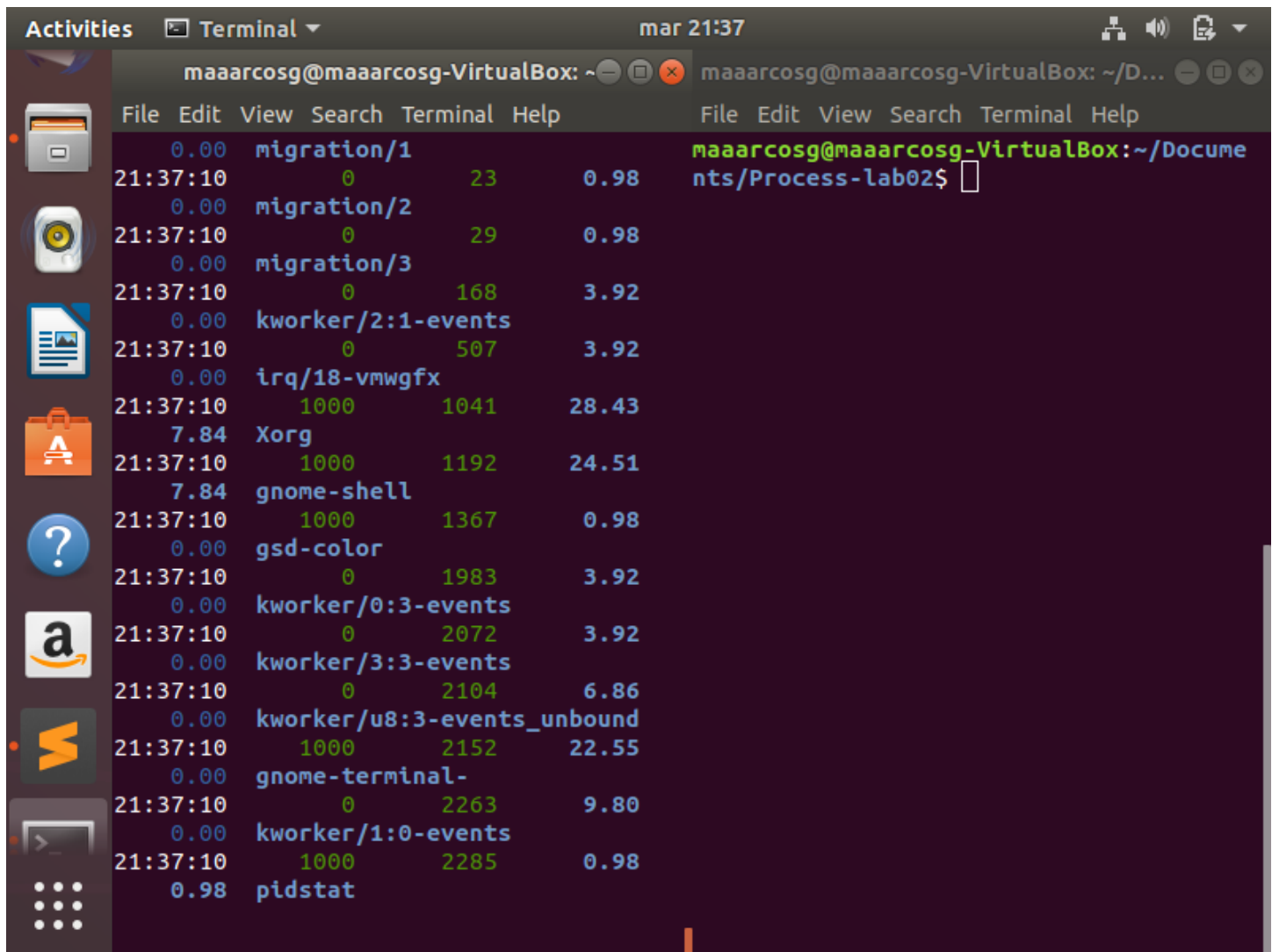
- **¿Cuál, en general, toma tiempos más largos?**

Es el primer programa el cual ejecuta los for de manera secuencial, ya que espera a que un for termine para que el otro for comience su ejecución.

- **¿Qué causa la diferencia de tiempo, o por qué se tarda más el que se tarda más?**

Por ejecutar de forma secuencial los for, en el segundo programa se ejecutan y se realiza la concurrencia, el cual es una técnica para mantener al CPU con la ejecución de cada proceso de manera en la que se simulen los for que se ejecutan al mismo tiempo.

### Ejercicio No. 3



The screenshot shows a terminal window titled 'Terminal' with a timestamp of 'mar 21:37'. The terminal displays the output of the 'top' command, showing system metrics and process information. The output is as follows:

```
0.00 migration/1
21:37:10 0 23 0.98
0.00 migration/2
21:37:10 0 29 0.98
0.00 migration/3
21:37:10 0 168 3.92
0.00 kworker/2:1-events
21:37:10 0 507 3.92
0.00 irq/18-vmwgfx
21:37:10 1000 1041 28.43
7.84 Xorg
21:37:10 1000 1192 24.51
7.84 gnome-shell
21:37:10 1000 1367 0.98
0.00 gsd-color
21:37:10 0 1983 3.92
0.00 kworker/0:3-events
21:37:10 0 2072 3.92
0.00 kworker/3:3-events
21:37:10 0 2104 6.86
0.00 kworker/u8:3-events_unbound
21:37:10 1000 2152 22.55
0.00 gnome-terminal-
21:37:10 0 2263 9.80
0.00 kworker/1:0-events
21:37:10 1000 2285 0.98
0.98 pidstat
```

- **¿Qué tipo de cambios de contexto incrementa notablemente en cada caso, y por qué?**  
Cuando se movía la interfaz gráfica el cambio de contexto incrementaba, debido a que Xorg estaba realizando una llamada al sistema, en este caso la llamada es a la interfaz gráfica.
- **¿Qué diferencia hay en el número y tipo de cambios de contexto de entre programas?**  
El programa que no tiene fork toma más tiempo de ejecución en comparación con el programa que sí contiene el fork.
- **¿A qué puede atribuir los cambios de contexto voluntarios realizados por sus programas?**  
Al primer programa el de sin fork, por las llamadas a sistema se pueden dar que tomaba más tiempo de ejecución. En cambio, el segundo programa con los fork, debido a que adentro de cada uno de los forks había una llamada WAIT se realizaron más cambios de contexto.
- **¿A qué puede atribuir los cambios de contexto involuntarios realizados por sus programas?**  
Al primer programa por las interrupciones de los procesos dentro del CPU al realizar un solo proceso, no había una distribución de procesos adecuada, por lo que el procesamiento con lleva a cambios de contextos involuntarios.
- **¿Por qué el reporte de cambios de contexto para su programa con `fork()` muestra cuatro procesos, uno de los cuales reporta cero cambios de contexto?**  
Por la creación de los procesos, ya que cada fork se creaban 4 procesos a partir del padre, se creó al hijo.

#### Ejercicio No. 4

```
0 R 1000 3463 3112 99 80 0 - 1093 - pts/0 00:00:10 eje-04
1 Z 1000 3464 3463 0 80 0 - 0 - pts/0 00:00:00 ej <defunct>
```

- **¿Qué significa la Z y a qué se debe?**

Esta letra Z el código del estado del proceso, el estado Z significa “zombie” lo cual es que se encuentra difunto y se da más que todo cuando el proceso haya completado su ejecución.

Padre 3510

Hijo 3511

```
Killed
maaarcosg@maaarcosg-VirtualBox:~/Documents/lab-02$
```

- **¿Qué sucede en la ventana donde ejecutó su programa?**

Finaliza, en donde el while infinito del padre se ejecutaba, es decir, matamos el proceso en el que iba. Lo cual no ocurría cuando el proceso padre continuaba su ejecución.

- **¿Quién es el padre del proceso que quedó huérfano?**

El padre del proceso huérfano es el hijo del proceso anterior, es decir, el padre del hijo fue el que matamos y automáticamente se volvió el padre.