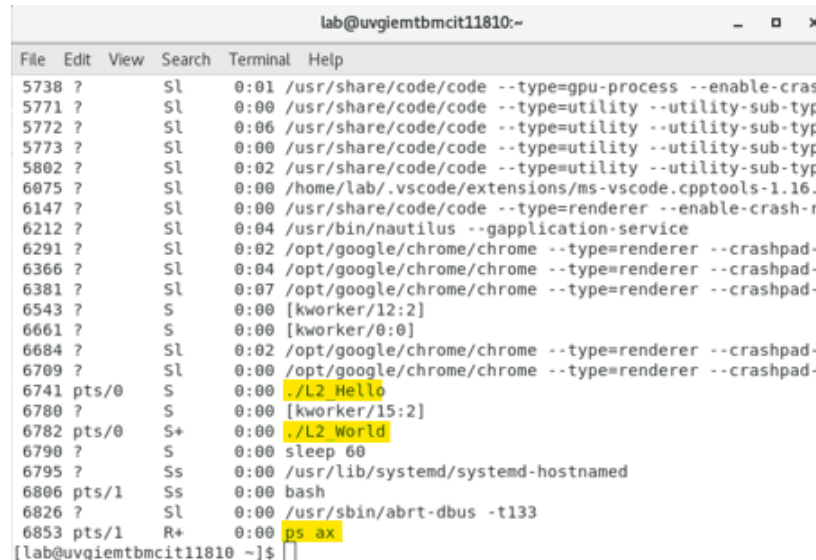


## Laboratorio 3

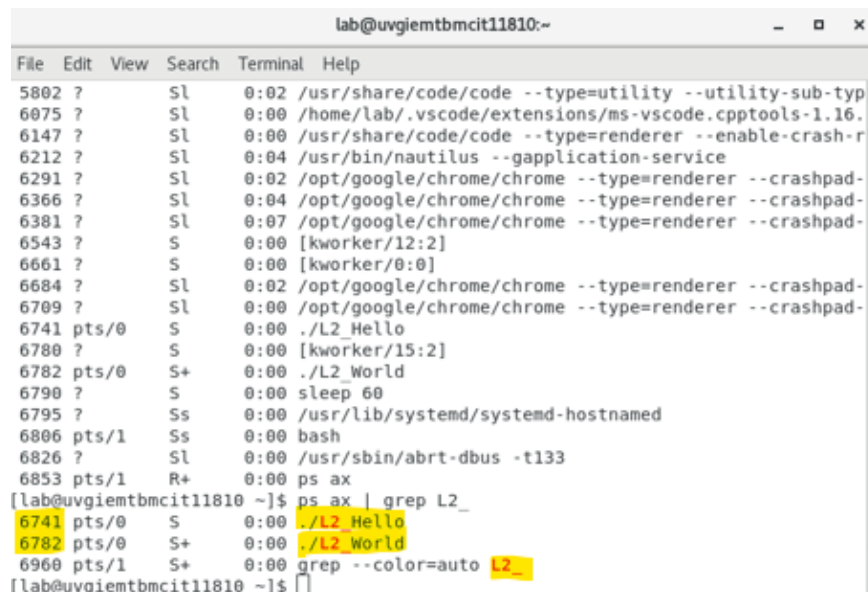
### Primera Parte

#### Inciso 3:



```
lab@uvgiemtbcit11810:~  
File Edit View Search Terminal Help  
5738 ? Sl 0:01 /usr/share/code/code --type=gpu-process --enable-cras  
5771 ? Sl 0:00 /usr/share/code/code --type=utility --utility-sub-typ  
5772 ? Sl 0:06 /usr/share/code/code --type=utility --utility-sub-typ  
5773 ? Sl 0:00 /usr/share/code/code --type=utility --utility-sub-typ  
5802 ? Sl 0:02 /usr/share/code/code --type=utility --utility-sub-typ  
6075 ? Sl 0:00 /home/lab/.vscode/extensions/ms-vscode.cpptools-1.16.  
6147 ? Sl 0:00 /usr/share/code/code --type=renderer --enable-crash-r  
6212 ? Sl 0:04 /usr/bin/nautilus --gapplication-service  
6291 ? Sl 0:02 /opt/google/chrome/chrome --type=renderer --crashpad-  
6366 ? Sl 0:04 /opt/google/chrome/chrome --type=renderer --crashpad-  
6381 ? Sl 0:07 /opt/google/chrome/chrome --type=renderer --crashpad-  
6543 ? S 0:00 [kworker/12:2]  
6661 ? S 0:00 [kworker/0:0]  
6684 ? Sl 0:02 /opt/google/chrome/chrome --type=renderer --crashpad-  
6709 ? Sl 0:00 /opt/google/chrome/chrome --type=renderer --crashpad-  
6741 pts/0 S 0:00 ./L2_Hello  
6780 ? S 0:00 [kworker/15:2]  
6782 pts/0 S+ 0:00 ./L2_World  
6790 ? S 0:00 sleep 60  
6795 ? Ss 0:00 /usr/lib/systemd/systemd-hostnamed  
6806 pts/1 Ss 0:00 bash  
6826 ? Sl 0:00 /usr/sbin/abrt-dbus -t133  
6853 pts/1 R+ 0:00 ps ax  
[lab@uvgiemtbcit11810 ~]$
```

Figura 1. ps ax mientras programas Hello y World están corriendo.



```
lab@uvgiemtbcit11810:~  
File Edit View Search Terminal Help  
5802 ? Sl 0:02 /usr/share/code/code --type=utility --utility-sub-typ  
6075 ? Sl 0:00 /home/lab/.vscode/extensions/ms-vscode.cpptools-1.16.  
6147 ? Sl 0:00 /usr/share/code/code --type=renderer --enable-crash-r  
6212 ? Sl 0:04 /usr/bin/nautilus --gapplication-service  
6291 ? Sl 0:02 /opt/google/chrome/chrome --type=renderer --crashpad-  
6366 ? Sl 0:04 /opt/google/chrome/chrome --type=renderer --crashpad-  
6381 ? Sl 0:07 /opt/google/chrome/chrome --type=renderer --crashpad-  
6543 ? S 0:00 [kworker/12:2]  
6661 ? S 0:00 [kworker/0:0]  
6684 ? Sl 0:02 /opt/google/chrome/chrome --type=renderer --crashpad-  
6709 ? Sl 0:00 /opt/google/chrome/chrome --type=renderer --crashpad-  
6741 pts/0 S 0:00 ./L2_Hello  
6780 ? S 0:00 [kworker/15:2]  
6782 pts/0 S+ 0:00 ./L2_World  
6790 ? S 0:00 sleep 60  
6795 ? Ss 0:00 /usr/lib/systemd/systemd-hostnamed  
6806 pts/1 Ss 0:00 bash  
6826 ? Sl 0:00 /usr/sbin/abrt-dbus -t133  
6853 pts/1 R+ 0:00 ps ax  
[lab@uvgiemtbcit11810 ~]$ ps ax | grep L2_  
6741 pts/0 S 0:00 ./L2_Hello  
6782 pts/0 S+ 0:00 ./L2_World  
6960 pts/1 S+ 0:00 grep --color=auto L2_  
[lab@uvgiemtbcit11810 ~]$
```

Figura 2. ps ax | grep L2\_ mientras programas Hello y World están corriendo.

En este primer inciso se pedía correr dos programas diferentes; en uno de ellos se mostraba en la terminal la palabra world y en otro la palabra hello. Este inciso era solo para observar el funcionamiento del operador pipe | y la opción de grep. En la figura 1 vemos que se listan todos los procesos que se están ejecutando, mientras que en la

figura 2 únicamente los procesos que contienen “L2\_” aparecen listados. El operador pipe | se usa para poder “conectar” la salida de un comando a la entrada de otro comando. Esto le permite encadenar varios comandos para realizar operaciones más específicas. El comando grep es un comando utilizado para buscar patrones de texto dentro de archivos.

**Inciso 4:** Los PIDs de los procesos anteriores se puede ver en la Figura 2. Al ingresar el comando kill seguido del PID de uno de los dos procesos lo que sucede es que el proceso termina. Se pueden terminar varios procesos ingresando los PIDs de los procesos que se quieren terminan únicamente con un kill.

**Inciso 6:** En este inciso se pide compilar un programa que utiliza la librería de pthread. En Linux, al intentar compilar el programa con el comando de gcc salta un error que dice que en el main la función pthread\_create y la función pthread\_join no se encuentran definidas, por lo que no compila el programa. El problema es que la librería no está incluida, por lo que se tiene que incluir cuando se compila el programa escribiendo -lpthread.

**Inciso 7:** En este inciso se pedía ejecutar el programa L2\_Hilos\_Ej2, que lo que hacía era crear un hilo aparte del principal donde se imprimía el mensaje de Hello con un retardo de 1.1s. En el principal se mostraba el mensaje World con un retardo de 1s. Este programa hacía exactamente lo mismo que los programas vistos en los incisos 1 y 2 a primera vista.

**Inciso 8:** Como se menciona la respuesta del inciso anterior, a primera vista parece que es lo mismo, pero ahora se verá la diferencia.

```
lab@uvgiemtbcit11810:~$ ps ax | grep L2_
6366 ?        Sl      0:04 /opt/google/chrome/chrome --type=renderer --crashpad-
6381 ?        Sl      0:07 /opt/google/chrome/chrome --type=renderer --crashpad-
6543 ?        S        0:00 [kworker/12:2]
6661 ?        S        0:00 [kworker/0:0]
6684 ?        Sl      0:02 /opt/google/chrome/chrome --type=renderer --crashpad-
6709 ?        Sl      0:00 /opt/google/chrome/chrome --type=renderer --crashpad-
6741 pts/0      S        0:00 ./L2_Hello
6780 ?        S        0:00 [kworker/15:2]
6782 pts/0      S+       0:00 ./L2_World
6790 ?        S        0:00 sleep 60
6795 ?        Ss       0:00 /usr/lib/systemd/systemd-hostnamed
6806 pts/1      Ss       0:00 bash
6826 ?        Sl      0:00 /usr/sbin/abrt-dbus -t133
6853 pts/1      R+       0:00 ps ax
[lab@uvgiemtbcit11810 ~]$ ps ax | grep L2_
6741 pts/0      S        0:00 ./L2_Hello
6782 pts/0      S+       0:00 ./L2_World
6960 pts/1      S+       0:00 grep --color=auto L2_
[lab@uvgiemtbcit11810 ~]$ kill 6741
[lab@uvgiemtbcit11810 ~]$ kill 6782
[lab@uvgiemtbcit11810 ~]$ ps ax | grep L2_
7173 pts/0      Sl+      0:00 ./L2_Hilos_Ej2
7187 pts/1      S+       0:00 grep --color=auto L2_
[lab@uvgiemtbcit11810 ~]$
```

Figura 3. Comparación de programas del inciso 1 y 2, y el del inciso 6.

Al analizar los procesos listados se puede ver claramente que la diferencia radica en que el ejemplo L2\_Hilos\_Ej2 al trabajar con hilos, solo es un único proceso. Los

programas del inciso 1 y 2 eran dos procesos distintos que se estaban ejecutando al mismo tiempo.

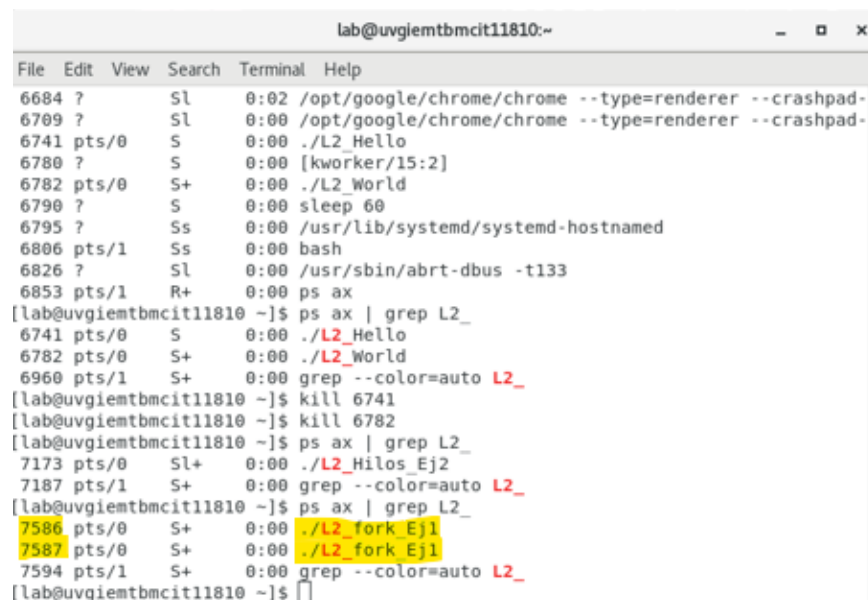
**Inciso 9:** En este inciso se pedía compilar el mismo programa solo que desde Visual Studio Code. Al intentar compilarlo había un problema, de la misma forma que anteriormente la función `pthread_create` no estaba definida. A pesar de incluir la librería `threads` surgía el mismo problema. Esto se soluciona abriendo los archivos json del workspace actual y añadiendo una línea extra añadiendo la librería `-lpthread` al intentar compilarlo.

## Segunda Parte

**Inciso 1:** En este inciso únicamente se pedía estudiar el ejemplo de `L2_fork_Ej1`. Al ver el código se puede notar que al igual que los programas de la primera parte tiene varios prints que mostrarán el mensaje de hello world, y también con sus respectivos delays de 1.1s y 1s.

**Inciso 2:** Al ejecutar el programa muestra de la misma forma que los programas anteriores el mensaje de Hello World, mostrándose el World de con un retardo menor al Hello.

**Inciso 3:**



```
lab@uvgiemtbcit11810:~  
File Edit View Search Terminal Help  
6684 ? Sl 0:02 /opt/google/chrome/chrome --type=renderer --crashpad-  
6709 ? Sl 0:00 /opt/google/chrome/chrome --type=renderer --crashpad-  
6741 pts/0 S 0:00 ./L2_Hello  
6780 ? S 0:00 [kworker/15:2]  
6782 pts/0 S+ 0:00 ./L2_World  
6790 ? S 0:00 sleep 60  
6795 ? Ss 0:00 /usr/lib/systemd/systemd-hostnamed  
6806 pts/1 Ss 0:00 bash  
6826 ? Sl 0:00 /usr/sbin/abrt-dbus -t133  
6853 pts/1 R+ 0:00 ps ax  
[lab@uvgiemtbcit11810 ~]$ ps ax | grep L2_  
6741 pts/0 S 0:00 ./L2_Hello  
6782 pts/0 S+ 0:00 ./L2_World  
6960 pts/1 S+ 0:00 grep --color=auto L2_  
[lab@uvgiemtbcit11810 ~]$ kill 6741  
[lab@uvgiemtbcit11810 ~]$ kill 6782  
[lab@uvgiemtbcit11810 ~]$ ps ax | grep L2_  
7173 pts/0 Sl+ 0:00 ./L2_Hilos_Ej2  
7187 pts/1 S+ 0:00 grep --color=auto L2_  
[lab@uvgiemtbcit11810 ~]$ ps ax | grep L2_  
7586 pts/0 S+ 0:00 ./L2_fork_Ej1  
7587 pts/0 S+ 0:00 ./L2_fork_Ej1  
7594 pts/1 S+ 0:00 grep --color=auto L2_  
[lab@uvgiemtbcit11810 ~]$
```

Figura 4. Procesos ejecutándose mientras se corre `L2_fork_Ej1`

Al observar en la Figura 4 se puede ver que hay dos procesos. Esto se debe a la utilización de la función `fork`, que crea un proceso hijo que tiene su propio PID. A pesar de que solo es un programa vemos que hay dos procesos distintos ejecutándose.

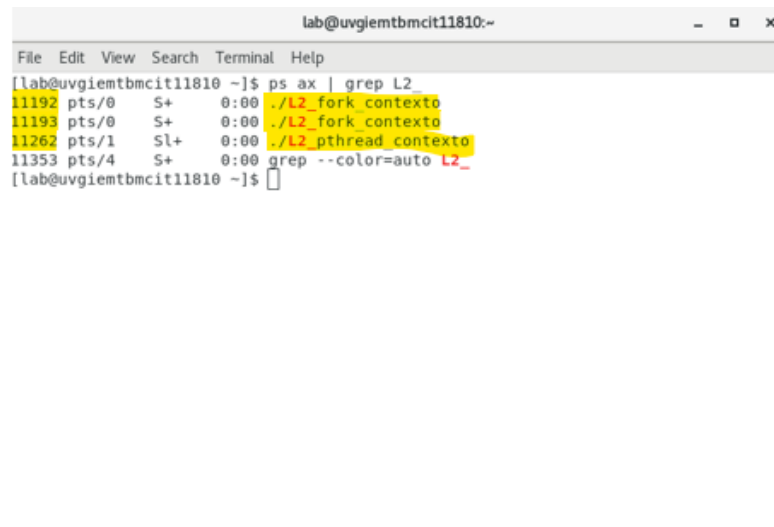
#### Inciso 4:

Tabla 1. Diferencias y similitudes entre L2\_fork\_contexto y L2\_thread\_contexto

Diferencias	Similitudes
Un programa utiliza la librería de pthreads, es decir que trabaja con hilos; mientras que el otro maneja la función fork, es decir que trabaja con procesos.	Ambos tienen la variable contador que en un momento se encuentra en 100 y otro en 200.
En el ejemplo de fork hay condicionales mientras que en el de threads no.	Muestran la variable contador que se imprime de diferentes maneras, una cada 1s y otra cada 1.1s.

**Inciso 6:** Al ejecutar ambos programas se ve una diferencia clara, y es que el programa de pthreads en algún momento la cuenta que empieza de 100 a 101 se pasa a los 200s y se queda ahí, a pesar de que se inició en 100. En el programa que utiliza fork podemos ver los dos prints que se ejecutan, solo que a diferencia del otro programa se puede ver que empieza igual, ambos prints en 100 pero en un momento uno se queda en los 100s, pero el otro se va a los 200s, a diferencia del otro programa que los dos se iban a 200s. Esta diferencia se debe que al trabajar con hilos la variable que se define es la misma ya sea para el hilo principal o para el segundo hilo, y cuando llega el momento que la variable se le asigna el valor de 200, se queda en ese valor y empieza a sumar desde ahí. En el caso del fork, este crea un proceso idéntico al del principal, pero con su propia memoria, y en este caso la variable counter es distinta para ambos procesos y por eso en el proceso padre se queda en los 100s pero en el hijo se queda en los 200s.

#### Inciso 7:



```
lab@uvgiemtbcit11810:~  
File Edit View Search Terminal Help  
[lab@uvgiemtbcit11810 ~]$ ps ax | grep L2  
11192 pts/0 S+ 0:00 ./L2_fork_contexto  
11193 pts/0 S+ 0:00 ./L2_fork_contexto  
11262 pts/1 Sl+ 0:00 ./L2_thread_contexto  
11353 pts/4 S+ 0:00 grep --color=auto L2_  
[lab@uvgiemtbcit11810 ~]$
```

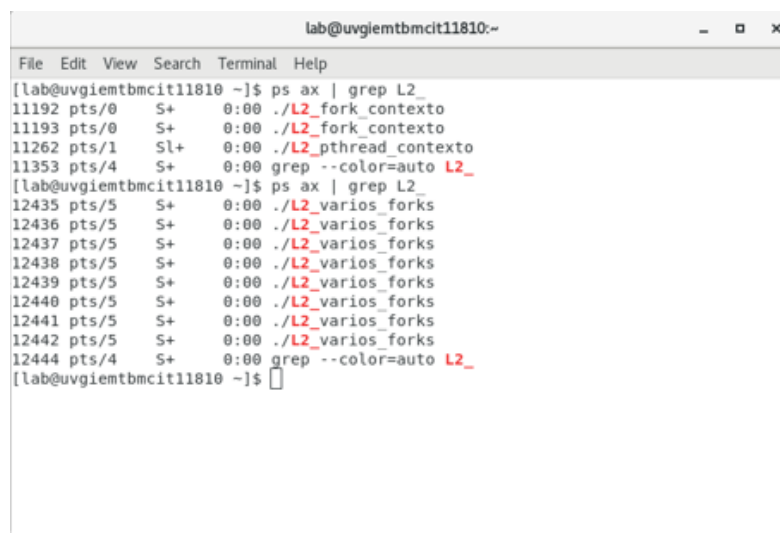
Figura 5. Procesos ejecutándose mientras corren los ejemplos de L2\_thread\_contexto y L2\_fork\_contexto.

En la Figura 5 se observan los procesos y se ve que el programa que utiliza los fork, hay dos procesos, esto es porque la función de fork crea un proceso, mientras que el programa que usa hilos crea un segundo hilo, pero no un proceso.

**Inciso 8:** En cuanto a condición de carrera se da cuando dos o más hilos que se están ejecutando al mismo tiempo intentan acceder a algún espacio que comparten, como memoria, variables, archivo y/o bases de datos. En este caso el ejemplo tenía dos hilos corriendo al mismo tiempo, y había una variable definida globalmente para el programa que los dos hilos tenían que utilizar. Entonces al momento de empezar el programa e hilo principal lleva la cuenta en 100s, pero el segundo hilo luego le quita la cuenta y la empieza en los 200s. Esto puede llevar a resultados que no sean correctos cuando alguno de los hilos realice algún tipo de cálculo, ya que el hilo espera que el contador se incrementará en un 1 pero se incrementa en dos ya que la variable es la misma.

### Tercera Parte

#### Inciso 1:



```
lab@uvgiemtmbmcit11810:~  
File Edit View Search Terminal Help  
[lab@uvgiemtmbmcit11810 ~]$ ps ax | grep L2_  
11192 pts/0 S+ 0:00 ./L2_fork_contexto  
11193 pts/0 S+ 0:00 ./L2_fork_contexto  
11262 pts/1 Sl+ 0:00 ./L2_thread_contexto  
11353 pts/4 S+ 0:00 grep --color=auto L2_  
[lab@uvgiemtmbmcit11810 ~]$ ps ax | grep L2_  
12435 pts/5 S+ 0:00 ./L2_varios_forks  
12436 pts/5 S+ 0:00 ./L2_varios_forks  
12437 pts/5 S+ 0:00 ./L2_varios_forks  
12438 pts/5 S+ 0:00 ./L2_varios_forks  
12439 pts/5 S+ 0:00 ./L2_varios_forks  
12440 pts/5 S+ 0:00 ./L2_varios_forks  
12441 pts/5 S+ 0:00 ./L2_varios_forks  
12442 pts/5 S+ 0:00 ./L2_varios_forks  
12444 pts/4 S+ 0:00 grep --color=auto L2_  
[lab@uvgiemtmbmcit11810 ~]$
```

Figura 6. Procesos mientras se ejecuta el programa de L2\_varios\_forks.

En este inciso se pedía ejecutar el programa de varios forks y ver que aparecía en los procesos mientras se ejecutaba. Al ver la Figura 6 notamos que hay un total de 8 procesos que provienen del mismo programa, esto hace sentido con lo visto en clase ya que cada vez que se ejecuta un fork, todos los procesos que ejecutan esa línea de código crean un proceso hijo. En el código vemos que se ejecutan tres forks seguidos; al ejecutarse el primero se crea un solo hijo, ya que solo estaba el proceso principal, en la siguiente línea al haber dos procesos, cada de estos procesos crea un proceso hijo y por lo tanto hay cuatro; al ejecutarse el último fork hay cuatro procesos en total, y cada uno de ellos crea un proceso hijo por lo que en total debería haber 8 procesos.