A close up of a sign

Description generated with high confidenceInstituto Politécnico Nacional  
Escuela Superior de

Cómputo

Reporte de la práctica 1

Materia:

“Aplicación a Redes”

Profesor:

Rangel Gonzáles Josué

Grupo:

3CV7

Alumnos:

-Sánchez Escamilla Alejandro

-Onofre Resendiz Uriel

Introducción

**Hilo o hebra**

Un proceso típico de Unix puede ser visto como un único **hilo de control**: cada proceso hace sólo una cosa a la vez. Con múltiples **hilos de control** podemos hace más de una cosa a la vez cuando cada hilo de hace cargo de una tarea.  
Beneficios de hilos:

* Se puede manejar eventos asíncronos asignando un hilo a cada tipo de evento. Luego cada hilo maneja sus eventos en forma sincrónica.
* Los hilos de un proceso comparten el mismo espacio de direcciones y descriptores de archivos.
* Procesos con múltiples tareas independientes pueden terminar antes si estas tareas se desarrollan de forma traslapada en hilos separados. De este modo tiempos de espera de la primera tarea no retrasan la segunda.
* Programas interactivos pueden lograr mejor tiempo de respuesta usando hilos para manejar la entrada y salida. Este es un ejemplo del punto previo.
* La creación de un hilo es mucho más rápida y toma menos recursos que la creación de un proceso.

Multi hilos se aplica a máquinas con uno o múltiples procesadores.  
Un hilo contiene la información necesaria para representar un contexto de aplicación dentro de un proceso. Ésta es:

* ID del hilo. No son únicos dentro del sistema, sólo tienen sentido en el contexto de cada proceso.
* Stack pointer
* Un conjunto de registros
* Propiedades de iteración (como política y prioridad)
* Conjunto de señales pendientes y bloqueadas.
* Datos específicos del hilo.

En ingles se escribe Thread y la traducción más clara es hilo. Creo que el ejemplo actual para explicar lo que es y lo que hace un hilo es utilizar las aplicaciones Android.

Si tenemos un dispositivo móvil al que le damos click en el icono de una aplicación para que se abra, entonces el dispositivo seguramente ya tiene varias aplicaciones abiertas, así que podríamos decir que cada aplicación es un hilo. Queda claro que los hilos son independientes, es decir las aplicaciones, lo que haga una no debería de afectar a la otra, ahora entonces cuando estamos en una aplicación vemos que tenemos cosas como descarga de imágenes, accesos a las bases de datos, etc. Estas tareas normalmente tienen un problema: no sabemos cuánto tiempo van a tardar, de hecho, corremos siempre el riesgo de que la tarea no se complete.

Imagina entonces que en una aplicación queremos descargar una imagen y contamos con una conexión a internet que no es la mejor, además es una imagen muy pesada, el usuario al no ver que la imagen se muestra en su teléfono puede pensar que la aplicación no funciona o que simplemente se quedó atorado, pero en realidad solo tomará más tiempo de lo debido. Entonces tenemos un problema ya que cada aplicación es un solo hilo y necesitamos el hilo para trabajar con la imagen y ya que esta tarea está pendiente no podremos avanzar. Los hilos nos ayudan a resolver estos problemas.

Una aplicación móvil bien hecha hoy en día crea un hilo para tareas como descargas, conexiones, etc. de esa manera la aplicación no se queda atascada en esas tareas esperando a que terminen. Los hilos nos permiten separar tareas complejas en tareas más simples y concretas, además nos permite fallar y que esto no sea tan grave, ¿te imaginas que cada vez que algo pase la aplicación o el sitio web se quede colgado? Ahora lo que hacemos es dividir las tareas en hilos y si tenemos algún problema el impacto no es tan negativo, obviamente al dividir en más partes el código también se vuelve más complejo, pero en algunos casos vale la pena asumir ese coste.

PIL / Pillow y aplicar efectos visuales

Python Imaging Library (PIL) es una librería gratuita que permite la edición de imágenes directamente desde Python. Soporta una variedad de formatos, incluidos los más utilizados como GIF, JPEG y PNG. Una gran parte del código está escrito en C, por cuestiones de rendimiento.

Debido a que la librería soporta únicamente hasta la versión 2.7 de Python y, al parecer, no pretende avanzar con el desarrollo, Alex Clark y en colaboración con otros programadores ha desarrollado Pillow, una bifurcación más «amigable», según el autor, de PIL que pretende mantener una librería estable y que se adapte a las nuevas tecnologías (Python 3.x). Por esta razón, recomiendo siempre preferir Pillow en lugar de PIL.

De todas maneras, independientemente de la librería que se desee utilizar, su implementación para los usuarios se mantiene casi idéntica.

Glob

Aunque la interfaz de glob es pequeña, el módulo incluye una gran cantidad de poder. Es útil en cualquier situación donde un programa necesita buscar una lista de archivos en el sistema de archivos con nombres que coinciden con un patrón. Para crear una lista de nombres de archivos que tengan una cierta extensión, prefijo, o cualquier cadena común en el medio, usa glob en lugar de escribir código personalizado para escanear los contenidos del directorio.

Las reglas de patrón para glob no son las mismas que las expresiones regulares usadas por el módulo [**re**](https://rico-schmidt.name/pymotw-3/re/index.html#module-re). En cambio, siguen reglas de expansión de ruta estándar de Unix. Solo hay unos pocos caracteres especiales utilizados para implementar dos caracteres comodines y rangos diferentes. Las reglas de patrón se aplican a segmentos del nombre de archivo (deteniéndose en el separador de ruta, /). Las rutas en el patrón pueden ser relativas o absolutas. Los nombres de variable de shell y tilde (~) no son expandidos.

Primero se abre la imagen a base de su archivo.

Se guarda el tamaño y el modo de la imagen.

Se realiza un mapeo de los pixeles de la imagen original.

Se realiza una nueva imagen con el tamaño y modo de la original y se realiza su mapeo.

Se realiza una nueva imagen con el tamaño y modo de la original y se realiza su mapeo.

Se realiza una nueva imagen con el tamaño y modo de la original y se realiza su mapeo.

Para i en un rango (ancho)

Para i en un rango (ancho)

Para i en un rango (ancho)

Para i en un rango (altura)

Para i en un rango (altura)

Para i en un rango (altura)

Se realiza una matriz con los pixeles de la imagen y se modifica el color azul

Se realiza una matriz con los pixeles de la imagen y se modifica el color verde.

Se realiza una matriz con los pixeles de la imagen y se modifica el color rojo.

Se asigna el valor de “255” al color modificado (azul).

Se asigna el valor de “255” al color modificado (rojo).

Se asigna el valor de “255” al color modificado (verde).

Se guarda una nueva imagen con los pixeles modificados.

Se guarda una nueva imagen con los pixeles modificados.

Se guarda una nueva imagen con los pixeles modificados.

Primero se le pide al usuario los argumentos para la ejecución del programa:  
Matriz A, matriz B y hilos.

Matiz A = 2000 \*1500.

Matriz B= 1500 \* 2000.

Hilos= 2, 4, 8, 16, 32.

Se verifica que los argumentos sean correctos.

Se asigna memoria para las matrices y se crean.

Para i  
Hasta el numero total de hilos

Incremento 1

La operación es exacta

No se utiliza el sobrante

Se utiliza el sobrante

Se crean los hilos solicitados y se llena la matriz

Se crean los hilos solicitados y se llena la matriz

Se mide el tiempo de ejecución con cada cantidad de hilos

Se muestra el tiempo de ejecución.

Explicación

Inciso A

Explicando el paso a paso del diagrama de flujo se inicia con un proceso mediante el cual se añade un tipo de archivo ya sea “PNG”, “JPG” o “JPEG” el cual se guardará como orig\_imagen. Esta misma contendrá un ancho, un largo y un modo, los cuales se guardaran por medio de las funciones width, height = orig\_image.size y mode = orig\_image.mode.

Luego se realizan mapeos de los pixeles de la imagen original por medio de las funciones:

* + orig\_pixel\_map = orig\_image.load() : Esta realiza el primer mapeo de cada uno de los pixeles de la imagen original.
  + new\_image\_r = Image.new(mode, (width,height)): Esta Genera una nueva imagen con el mismo tamaño y modo de la imagen original.
  + new\_pixel\_map\_r=new\_image\_r.load(): Y por último esta función lo que hace es mapear la nueva imagen.

En el siguiente paso se realiza una matriz por medio del mapeo ya realizado anterior mente, en este paso se tomarán los pixeles de ancho y largo para generar una matriz y poder cambiar el color de cada uno de los pixeles. En este caso se cambiarán a color rojo.

Después se le asignara el numero 255 al color que ha sido cambiado por medio de la función:

new\_r = 255

new\_g = orig\_pixel[1]

new\_b = orig\_pixel[2]

En esta misma se observa que los 2 siguientes colores, los cuales son verde y azul, se mantendrán con su valor original.

Y para finalizar esta primera matriz lo que se hace es guardar un nuevo archivo ya modificado con el color anterior ya mencionado.

Luego se vuelve a generar una nueva imagen con el mismo tamaño y modo de la imagen original, con la función anterior ya mencionada, y se realiza el mapeo de esta misma imagen.

En el siguiente paso se realiza nuevamente una segunda matriz, con el mapeo anteriormente hecho, tomando los pixeles de ancho y largo y nuevamente cambiar el color de cada uno de los pixeles. En este caso los pixeles se cambiarán a color verde.

Después se le asignara el numero 255 al color que ha sido cambiado por medio de la función:

new\_r = orig\_pixel[0]

new\_g = 255

new\_b = orig\_pixel[2]

Podemos volver a observar que 2 colores, los cuales son rojo y azul, se mantendrán con su valor original.

Finalmente, para esta matriz, lo que se hace es guardar un nuevo archivo ya modificado con el color anterior ya mencionado.

Luego se vuelve a generar por última vez una nueva imagen con el mismo tamaño y modo de la imagen original, con la función anterior ya mencionada, y se realiza el mapeo de esta misma imagen.

En el siguiente paso se realiza nuevamente una segunda matriz, con el mapeo anteriormente hecho, tomando los pixeles de ancho y largo y nuevamente cambiar el color de cada uno de los pixeles. En este caso los pixeles se cambiarán a color azul.

Después se le asignara el numero 255 al color que ha sido cambiado por medio de la función:

new\_r = orig\_pixel[0]

new\_g = orig\_pixel[1]

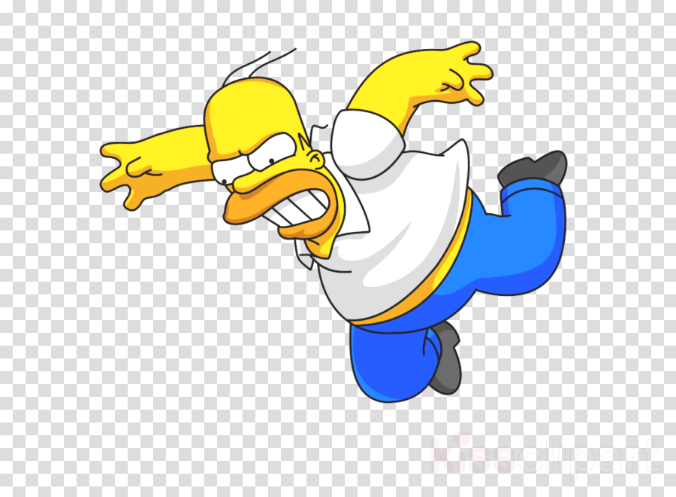
new\_b = 255

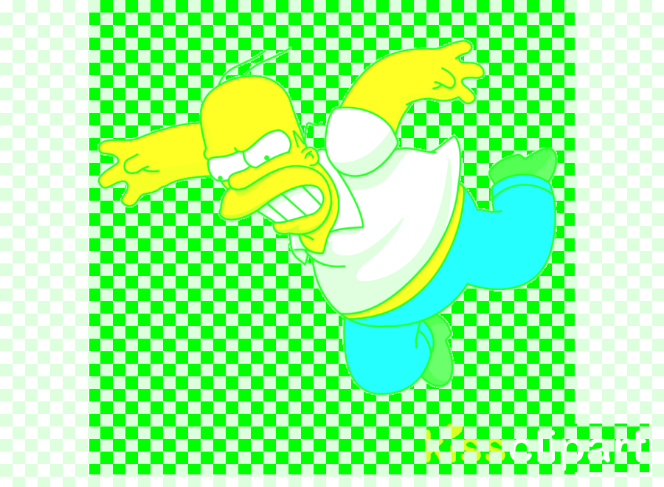
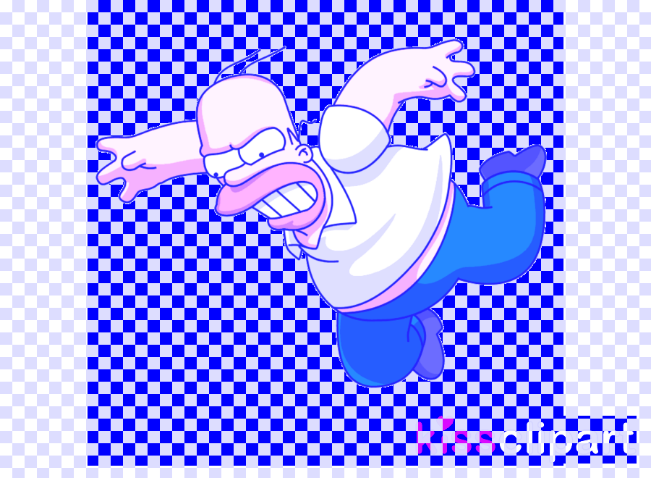
Podemos volver a observar que 2 colores, los cuales son rojo y verde, se mantendrán con su valor original.

Y por último paso, se guardará la última imagen ya modificada.

Cabe mencionar que en el caso de que no exista ninguna imagen archivada, las primeras 2 matices pasaran a la siguiente matriz y la última se moverá directo a finalizar el programa.

Capturas de la ejecución Inciso A





Inciso B

Explicando el paso a paso del diagrama este inicia a partir de la solicitud de los argumentos que se necesitan los cuales son:

* Matiz A = 2000 \*1500.
* Matriz B= 1500 \* 2000.
* Hilos= 2, 4, 8, 16, 32.

Posteriormente se verifica que cada uno de los datos solicitados sean correctos, en caso contrario se enviará un mensaje el cual dirá “ejecución mal lograda”.

Una vez estos sean verificados se comenzará a solicitar espacio en memoria y posteriormente a la creación de las matrices.

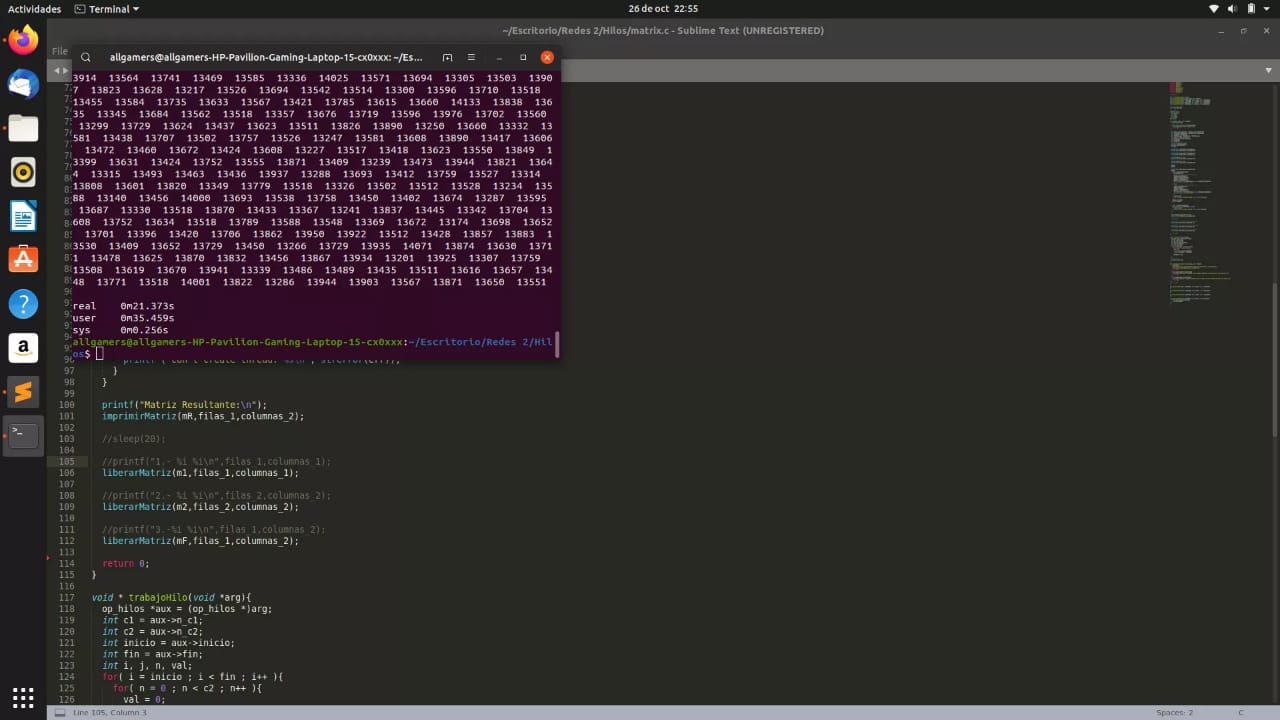
Como siguiente paso se llevará a cabo un ciclo para llenar las matrices con los hilos solicitados, en este caso el ciclo será para i, hasta el numero total de hilos y con un incremento de 1 en 1.

Posteriormente se realizará una función condicional para la posibilidad de que al realizar la operación de división, “rango = filas\_1/n\_hilos “, resulte inexacta, por lo que en el caso de que sea así se utilizara una función para el sobrante. Y posteriormente se comenzaran a crear el número de hilos solicitados mediante el ciclo y la función de creación de hilos: “pthread\_create(&hilos[i], NULL, trabajoHilo,&job[i])”.

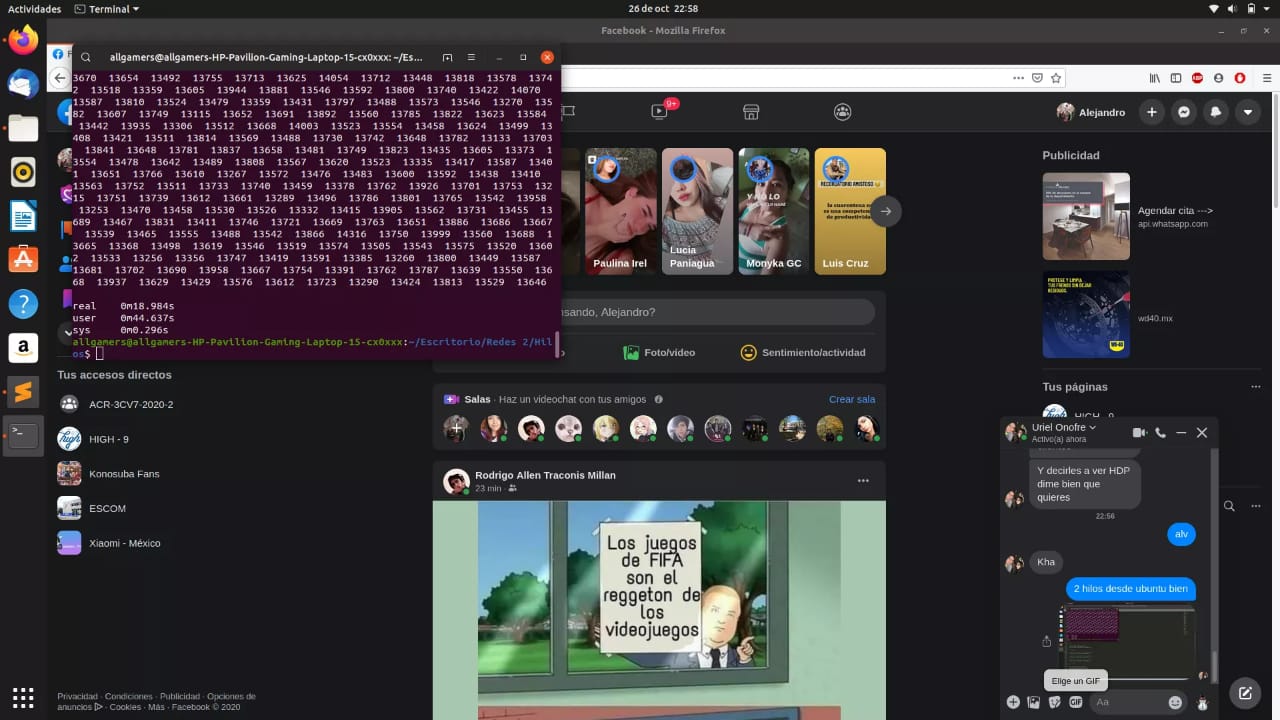
Finalmente, al terminar el ciclo se medirá el tiempo de ejecución en base a los hilos creados y se mostrará en pantalla el tiempo, así como la matriz resultante.

Capturas de la ejecución Inciso B

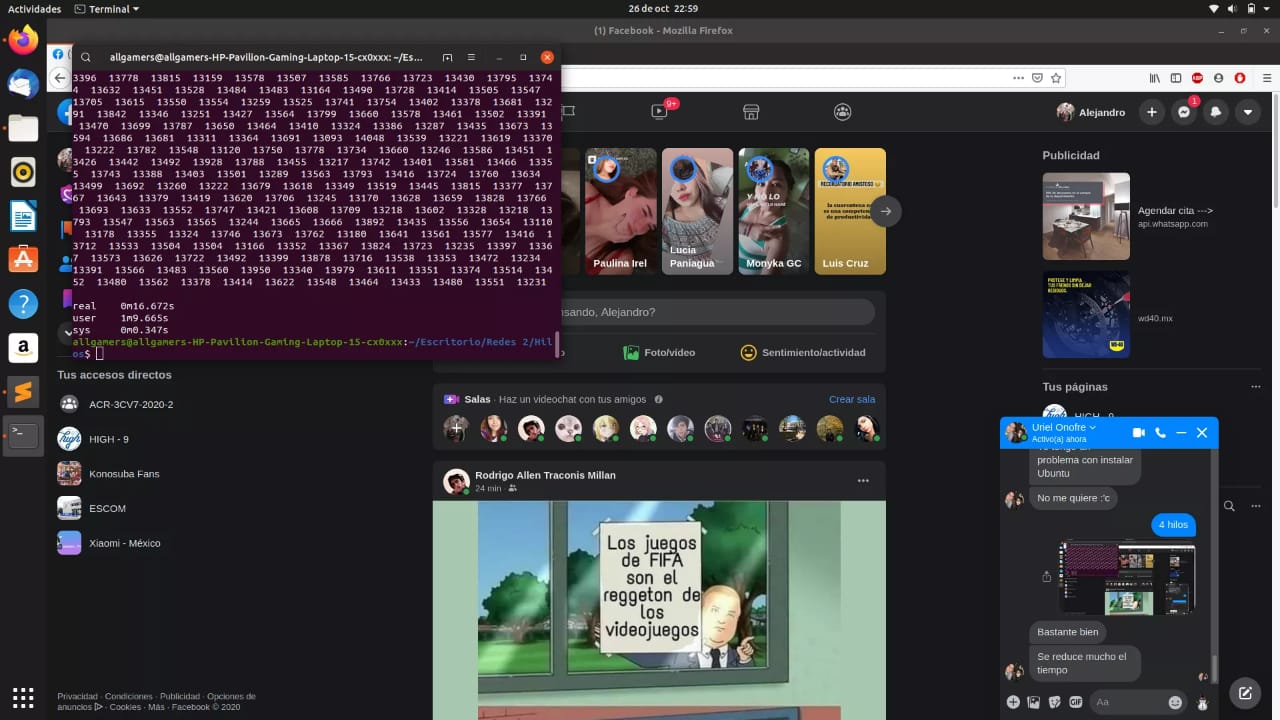
2 hilos.



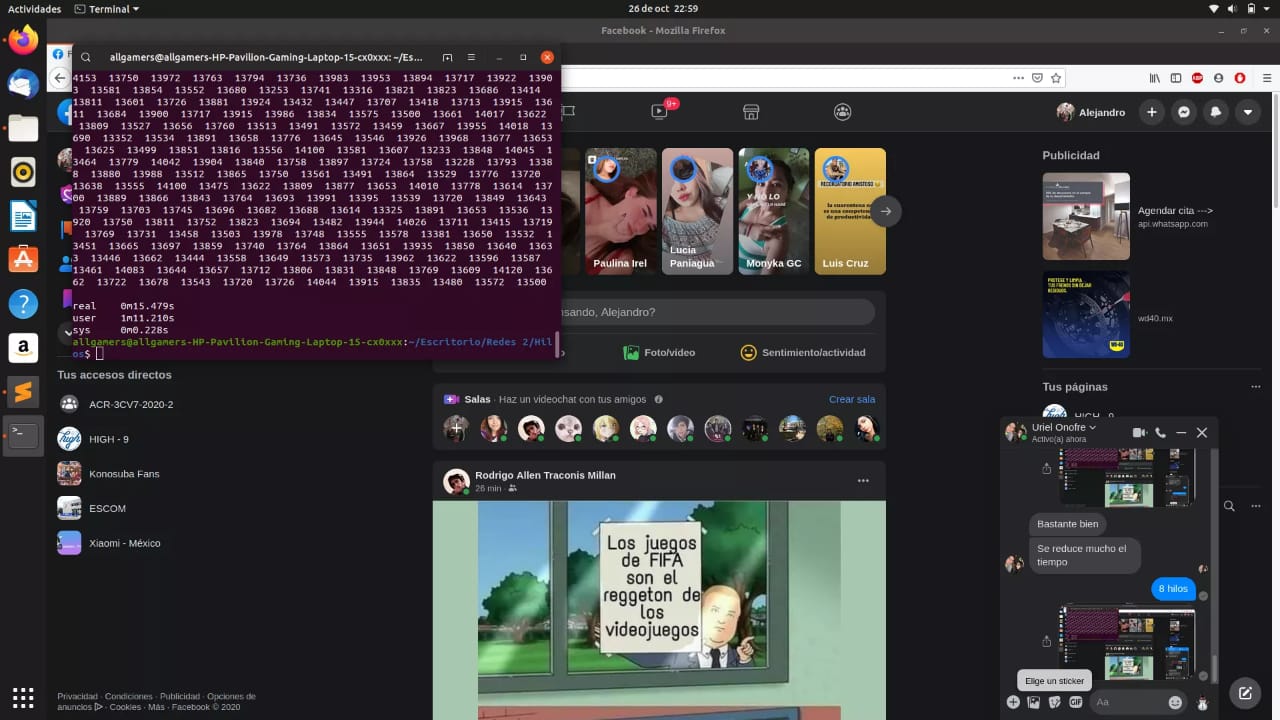
4 hilos.

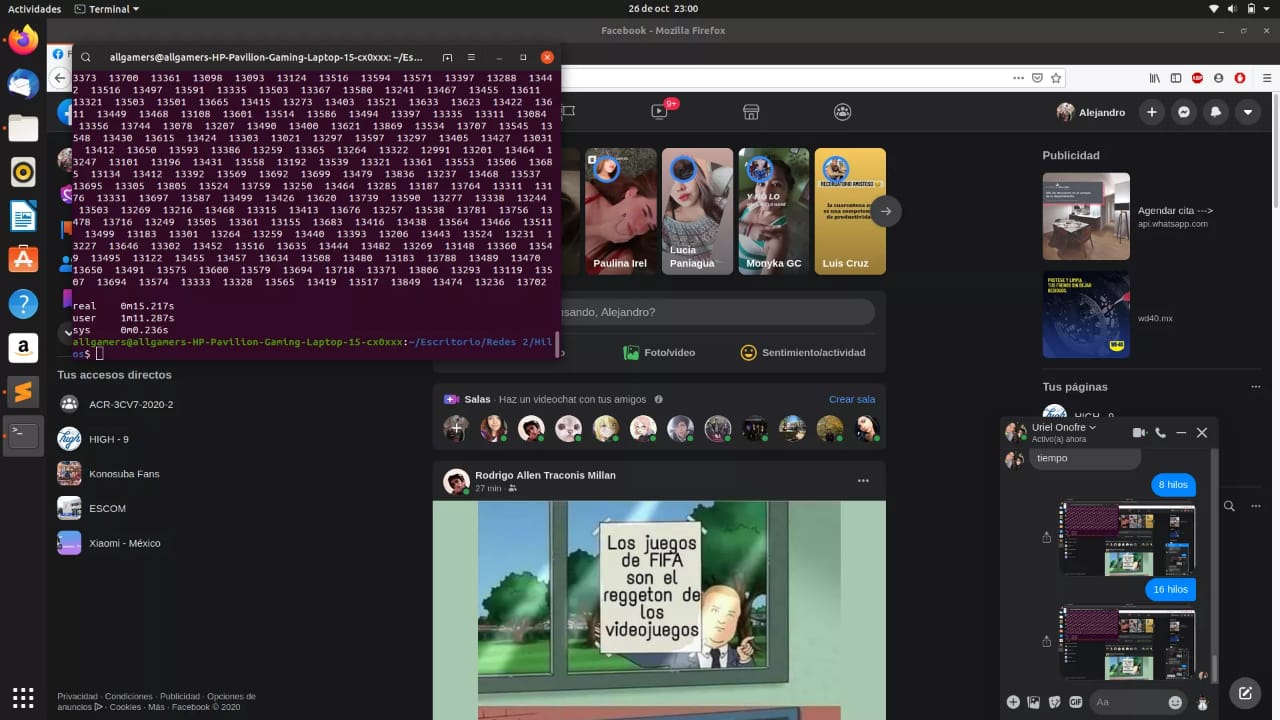


8 hilos.

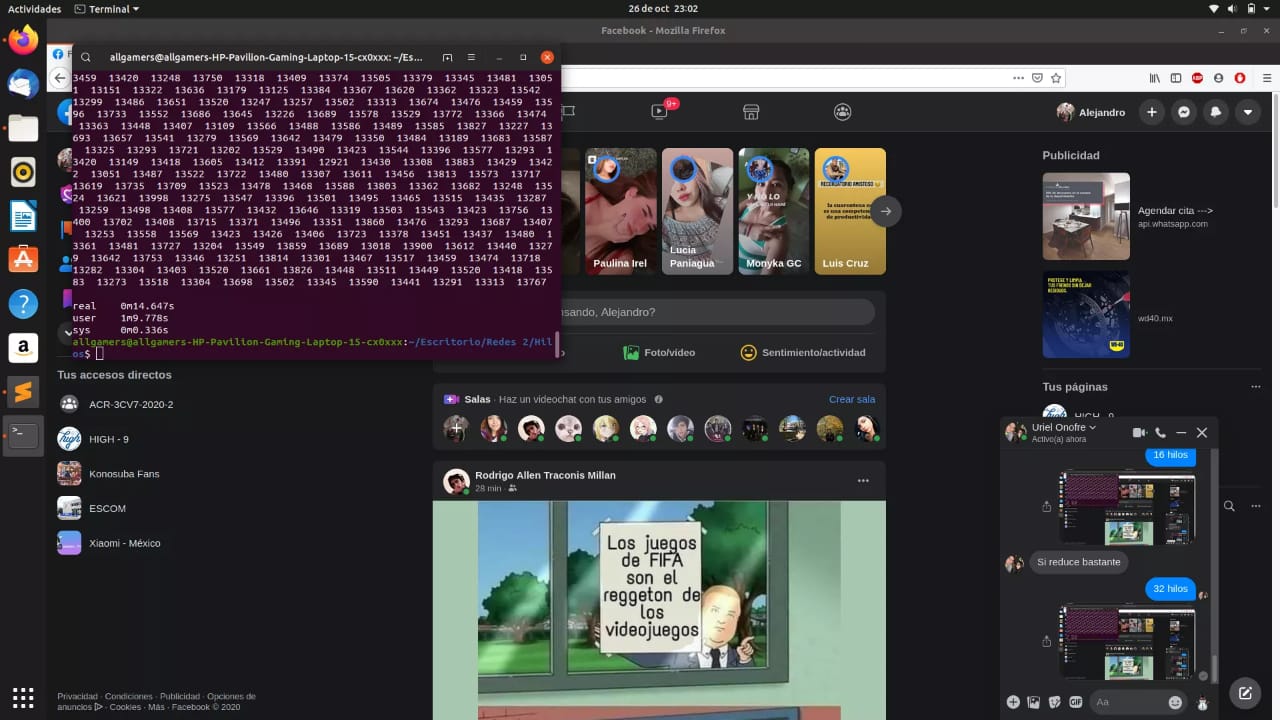


16 hilos.



32 hilos.

2000 hilos



Conclusiones

Onofre Resendiz Uriel:

De la practica realizada puedo decir que fue bastante entretenida dado que en mi caso nunca había realizado algún programa que utilizara imágenes y mucho menos que cambiara colores de estas mismas. Considero que dados los conocimientos ya adquiridos durante la clase fue una practica lograda. En cuanto al inciso A, puedo decir que fue un tanto compleja dado que se tuvo que investigar un poco en cuanto a las funciones necesarias para su realización, aunque en su mayoría fueron dadas por el profesor, esta práctica sí tomo su tiempo dada su complejidad. En cuanto al inciso B ya se tenia una gran parte del programa dado que fue hecho en clase, pero la parte del tiempo de ejecución fue agregado, de esto se puede decir que mientras mas hilos ejecuten la matriz, esta disminuye el tiempo de ejecución, por lo que es mejor el ejecutar este tipo de programas con varios hilos pero al ejecutarlo con una gran cantidad de hilos (5000) llega un punto en donde al tener demasiados hilos todos luchan por la ejecución por lo que se vuelve lento el proceso.

Escamilla Sánchez Alejandro:

Considero que esta práctica fue un tanto laboriosa ya que se para el inciso A se tuvo que recurrir a la investigación de algunas funciones, en el caso del inciso B ya se tenia una gran parte del Código y solo se tuvieron que hacer algunos cambios.

En cuanto al tiempo de ejecución del inciso A puedo decir al ejecutar el programa con los hilos solicitados, este si disminuye en cuanto a tiempo, pero en algunas veces el procesador si podría llegar a tardar mas con respecto a ejecutarlo con muchos mas hilos y esto se debe a que estos se atascan, dado que los hilos tardan en pasar al procesador por ser una gran cantidad de ellos.

Por lo que no vale tanto la pena ejecutar la matriz con una gran cantidad de hilos.