Threads:

Un thread es un set de instrucciones el cual se ejecuta cuando es llamado, el calendarizador se encarga de posicionarlo en la cola de procesos donde el decide que trabajo va a ser el necesario para asignarle los recursos que se necesiten, esto sucede en instantes de tiempo tan rápidos en el orden de 1 milisegundo = 10 giga procesos.

Un thread es un subset de procesos o bien un componente del proceso , por ejemplo, un proceso puede tener dos threads de ejecución corriendo en un mismo proceso, aun así este puede tener una cantidad finita de threads. También se puede decir que un thread is una unidad de calendarización y ejecución.

Un proceso de "peso pesado" tiene un "single thread" de control, también el proceso puede tener muchos threads de control (multithreading) el cual puede hacer varias tareas en un mismo tiempo.
Por ejemplo, una página web del explorador puede tener un thread cargando imágenes, otro thread cargando el texto, un thread haciendo consultas al servidor, etc… todo corre en un mismo instante de tiempo para el ojo humano.

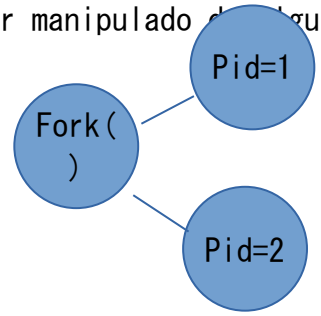
El software moderno normalmente trabaja optimizado y veloz, esto quiere decir que usa multithreading para poder mejorar la velocidad del cliente en su uso, podemos definir que multithreading tiene 4 subcategorías que la definen.

1. Velocidad de respuesta: El usuario no puede notar que un thread está siendo ejecutado aun así teniendo la parte del software en completo bloqueo el thread puede estar corriendo o multiples threads, esto lo puede hacer completamente eficiente.
2. Uso de recursos compartidos: Los threads pueden acceder a los espacios de memoria del escope en el que se encuentran más facilmente en un nivel de programación alto ya que comparten código y data, esto permite a la aplicación tener diferentes threads de actividad en el mismo spacio de memoria.
3. Economía: Alocar memoria y recursos es muy costoso para el sistema operativo, ya que los threads comparten recursos del proceso al que pertenecen es menos costoso crear un proceso y multiples threads.
4. Escalabilidad: Puede agrandarse según sea el hardware, puede darse una n cantidad de threads según sea la cantidad de procesadores.

Fork()

La llamada a sistema fork() es usado para crear un proceso, no requiere argumentos y retorna un ID de proceso el cual es un entero. Después de la llamada un proceso hijo es creado, los **DOS PROCESOS** ejecutarán la siguiente instrucción "al mismo tiempo".

Si la llamada a fork() se cumple con éxito retorna un valor mayor o igual a cero, crea dos copias idénticas de espacio de memoria una para el padre y otra para el hijo, podemos decir que son procesos independientes del otro, ya que el espacio de memoria y las variables definidas antes de la llamada estarían en el espacio de memoria del proceso. En otras palabras se genera un clon del proceso el cual puede ser manipulado de alguna forma de buen uso.

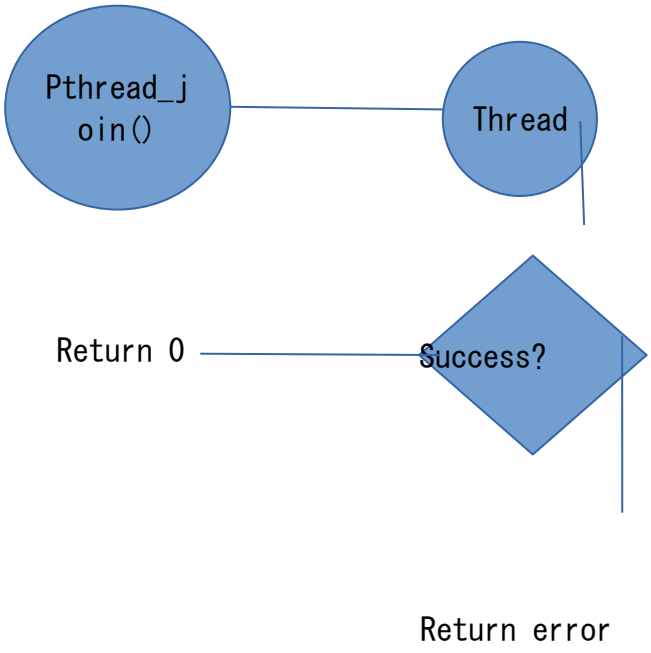


Esto no es un thread, es un proceso totalmente diferente pero con el mismo bloque de datos, es mucho más costoso que un thread pero según sea el uso puede hacer mucha significancia en la aplicación.

Pthread_join(pthread_t thread, void **retval)

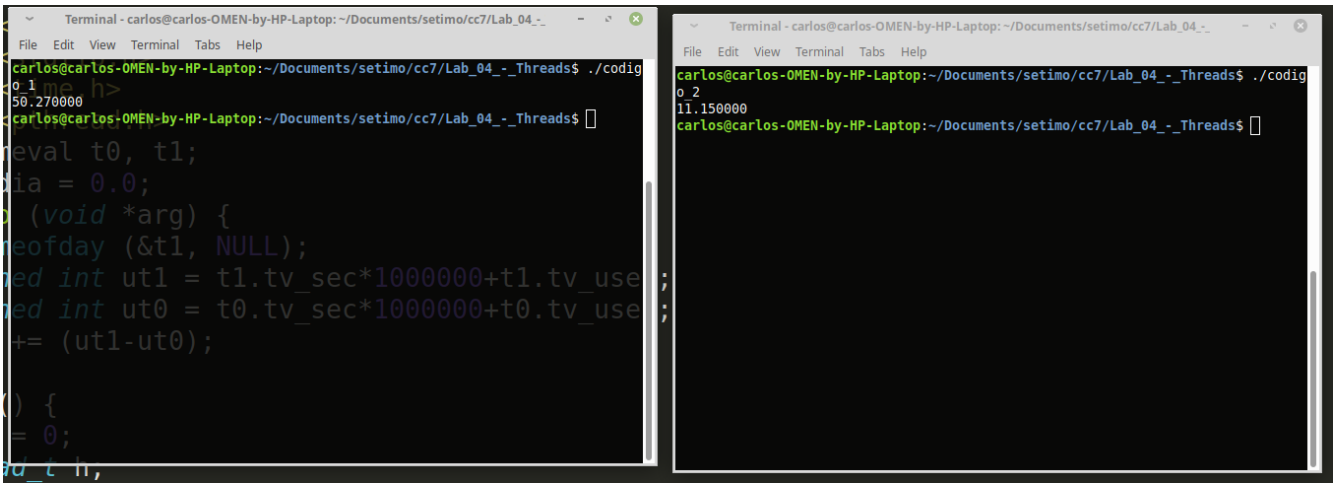
Pthread_join espera a que un thread que recibe como argumento sea finalizado con éxito, el thread tiene que ser joineable. La función después de la llamada retorna un cero en caso de éxito, en caso de error retorna un número de error al puntero especificado por retval.
Tipos de error:

- EDEADLK
- EINVAL
- EINVAL
- ESRCH



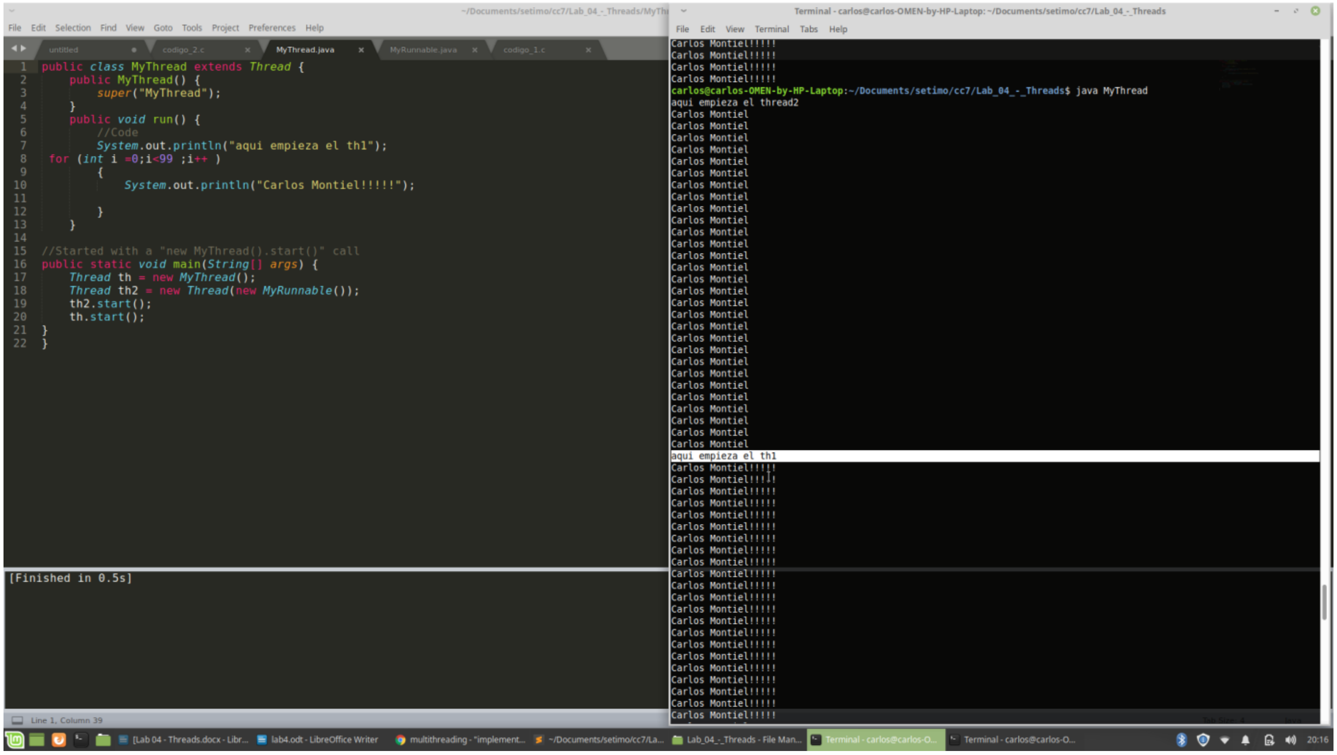
Codigo_1.c vs Codigo_2.c

En esta fase hay un ejemplo muy grafico acerca de la diferencia de las funciones fork() y pthread_create(), donde una genera un proceso completo mientras la otra solamente genera un thread en el mismo proceso el cual es mucho mas eficiente que fork() debido a que fork() copia todo el bloque de memoria en un nuevo proceso mientras el thread usa los recursos del mismo proceso, se crearon 100 threads en el codigo 2 y en el codigo 1 se crearon 100 procesos, es una gran diferencia aproximadamente de 5 veces de eficiencia.



Java, threads vs runnable

Los dos son casi lo mismo, la diferencia mas obvia es que cuando extiende de Threads significa que el proceso va a esperar a que el thread haya finalizado con exito y seguira, este interrumpe el proceso que se este manejando en el momento, si llamamos al thread que implementa a runnable antes que el que extiende a threads este empieza con su ejecucion y en un instante de tiempo cuando toma el calendarizador le da prioridad al thread extendido por threads este espera a que termine para seguir con el proceso, vea la imagen abajo.
Es explicito donde dice “Aqui comienza el th1” que se refiere al que extiende de threads, ese espera a que termine y continua imprimiendo los faltantes de th2.


~/Documents/setimo/cc7/Lab_04_-_Threads/MyThr
File Edit Selection Find View Goto Tools Project Preferences Help
untitled
codigo_2.c
MyThread.java
MyRunnable.java
codigo_1.c
public class MyThread extends Thread {
public MyThread() {
super("MyThread");
}
public void run() {
//Code
System.out.println("aqui empieza el th1");
for (int i =0;i<99 ;i++)
{
System.out.println("Carlos Montiel!!!!!");
}
}
//Started with a "new MyThread().start()" call
public static void main(String[] args) {
Thread th = new MyThread();
Thread th2 = new Thread(new MyRunnable());
th2.start();
th.start();
}
}
[Finished in 0.5s]
Line 1, Column 39
Terminal - carlos@carlos-OMEN-by-HP-Laptop: ~/Documents/setimo/cc7/Lab_04_-_Threads
File Edit View Terminal Tabs Help
Carlos Montiel!!!!!
Carlos Montiel!!!!!
Carlos Montiel!!!!!
Carlos Montiel!!!!!
carlos@carlos-OMEN-by-HP-Laptop:~/Documents/setimo/cc7/Lab_04_-_Threads$ java MyThread
aqui empieza el thread2
Carlos Montiel
aqui empieza el th1
Carlos Montiel!!!!!
Lab 04 - Threads.docx - LibreOffice Writer
lab4.odt - LibreOffice Writer
multithreading - "implement...
~/Documents/setimo/cc7/La...
Lab_04_-_Threads - File Man...
Terminal - carlos@carlos-O...
Terminal - carlos@carlos-O...
20:16