

Tarea 1.

Cabello Figueroa Israel

Octubre 2020

1 2. Manejo de procesos e hilos en c

1.1 Procesos

Para crear un proceso en C dentro de las distribuciones Linux usamos el comando `fork`, visto en clase. Para conocer el PID y el PPID de un proceso usamos las funciones `getpid` y `getppid`.

1.2 Hilos

Para usar hilos en c, con la librería POSIX de Linux, `pthread.lib` debemos incluir algunas cabeceras a diferencia del manejo de procesos. Para almacenar variables en un hilo usamos `pthread`

Para la creación o lanzamiento de los hilos usamos `pthread create` que está en la librería. Para la terminación de un hilo usamos `pthread join` que también está en la librería. Para pasar parametros a un hilo usamos `pthread_t` como en una variable. En la *Figura1* hay una muestra del uso de estos comandos.

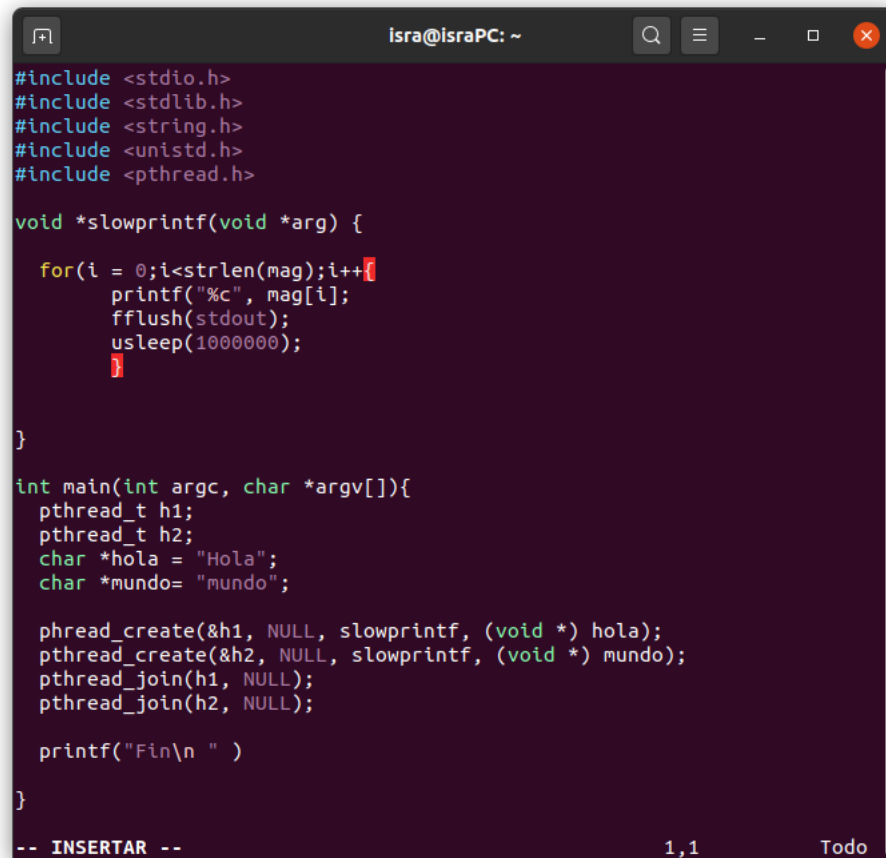
2 3. Conceptos Python

2.1 Global Interpreter Lock

En español la traducción sería Bloqueo de Interpretador Global, es una característica de Python muy desafortunada, básicamente y en palabras breves, es una propiedad que tiene Python que su interpretador solo puede ejecutar un hilo a la vez, es decir solo puede ejecutar una tarea, esta característica no es visible para desarrollo de monohilo pero para hacer multihilo en un CPU con más de un core, es un cuello de botella.

2.2 Ley de Amdahal

La Ley de Amdahal nos describe como cambia el rendimiento de un programa o un sistema al cambiar una parte/pieza del mismo, de forma breve establece que la mejora será proporcional solo al tiempo que se emplee la parte o pieza cambiada y el rendimiento que está tenga, lo cual nos permite definir si vale la

A screenshot of a terminal window titled 'Isra@IsraPC: ~'. The window has a dark purple background with light-colored text. The code is written in C and uses syntax highlighting. It includes headers for stdio, stdlib, string, unistd, and pthread. A function 'slowprintf' is defined, which prints a string character by character with a 1-second delay between each character. The 'main' function creates two threads, 'hola' and 'mundo', using 'pthread_create', and joins them with 'pthread_join'. It ends with a 'printf' statement and a return statement. The terminal status bar at the bottom shows '-- INSERTAR --', '1,1', and 'Todo'.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

void *slowprintf(void *arg) {
    for(i = 0; i<strlen(mag); i++){
        printf("%c", mag[i]);
        fflush(stdout);
        usleep(1000000);
    }
}

int main(int argc, char *argv[]){
    pthread_t h1;
    pthread_t h2;
    char *hola = "Hola";
    char *mundo = "mundo";

    pthread_create(&h1, NULL, slowprintf, (void *) hola);
    pthread_create(&h2, NULL, slowprintf, (void *) mundo);
    pthread_join(h1, NULL);
    pthread_join(h2, NULL);

    printf("Fin\n ")
}

-- INSERTAR -- 1,1 Todo
```

Figure 1: Ejemplo de manipulación de procesos e hilos en C

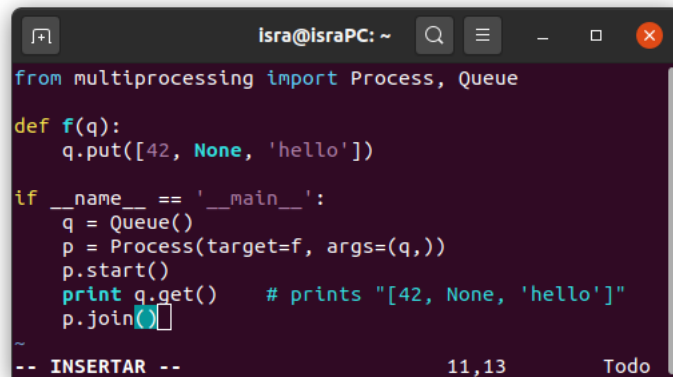
pena hacer un cambio. En nuestro caso aplicado a multiprocesos, tendríamos que evaluar si el proceso que queremos cambiar para realizar multithread vale la pena, ya que podría suceder que suponga mucho esfuerzo y en caso de que sea una parte de nuestro programa que se use muy poco tiempo podría no merecer el esfuerzo.

2.3 multiprocessing

La librería multiprocessing es la que nos permite hacer tareas multiples en python, está librería tiene varios comandos. El primero y más básico es Process, este comando tiene como argumentos una función y sus respectivos argumentos, así definimos nuestros procesos. Una vez definidos podemos usar el comando start, que no recibe argumentos como atributo de nuestro proceso para iniciarlo,

y al igual que en python con `join()` podemos esperar a que el proceso termine.

Para comunicar nuestros procesos usamos la clase `Queue` como en el ejemplo de la figura 2, donde también usamos `Process start` y `join`

A screenshot of a terminal window titled 'Isra@IsraPC: ~'. The code inside is as follows:

```
from multiprocessing import Process, Queue


def f(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=f, args=(q,))
    p.start()
    print q.get()    # prints "[42, None, 'hello']"
    p.join()
```

The terminal shows the code being executed, with a status bar at the bottom indicating '-- INSERTAR --', '11,13', and 'Todo'.

Figure 2: Ejemplo de Queue en Python

Otra función de la librería `multiprocessing` es `Lock`, que nos permite asegurar que solo un proceso usará la salida del sistema como lo muestra el ejemplo de la figura 3

A screenshot of a terminal window titled 'Isra@IsraPC: ~'. The code inside is as follows:

```
from multiprocessing import Process, Lock

def f(l, i):
    l.acquire()
    print 'hello world', i
    l.release()

if __name__ == '__main__':
    lock = Lock()

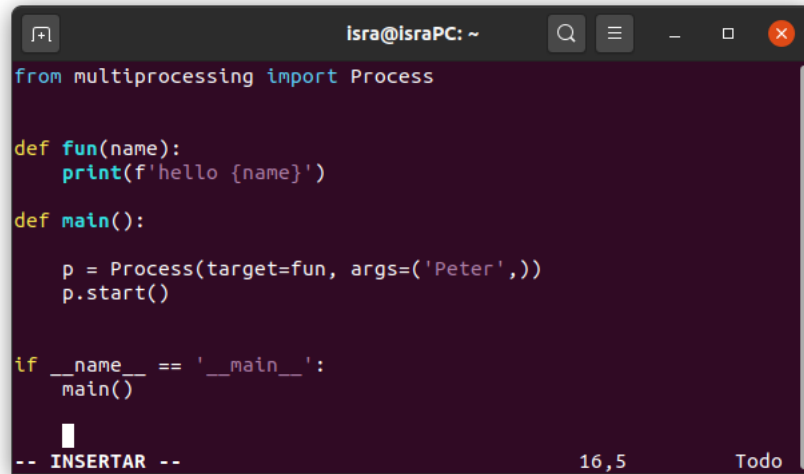
    for num in range(10):
        Process(target=f, args=(lock, num)).start()
```

The terminal shows the code being executed, with a status bar at the bottom indicating '-- INSERTAR --', '13,9', and 'Todo'.

Figure 3: Ejemplo de Lock() en Python

3 4. Crear un proceso.

Para crear un proceso con multiprocessing, podemos hacerlo con el siguiente ejemplo.

A screenshot of a terminal window titled 'isra@israPC: ~'. The window contains Python code that uses the 'multiprocessing' library to create a process. The code defines a function 'fun' that prints 'hello {name}', a 'main' function that creates and starts a process 'p' with 'fun' as the target and 'Peter' as the argument, and a standard if-main guard. The terminal shows the code being typed, with a cursor at the end of the last line. The status bar at the bottom indicates '-- INSERTAR --', '16,5', and 'Todo'.

```
from multiprocessing import Process

def fun(name):
    print(f'hello {name}')

def main():
    p = Process(target=fun, args=('Peter',))
    p.start()

if __name__ == '__main__':
    main()
```

Figure 4: Ejemplo de creación de proceso.

4 Referencias.

Use la documentacion de python en: [\[\[https://docs.python.org/2/library/multiprocessing.html\]\]](https://docs.python.org/2/library/multiprocessing.html)
Y la documentacion de la librería de c en : [\[\[https://computing.llnl.gov/tutorials/pthreads/\]\]](https://computing.llnl.gov/tutorials/pthreads/)