SEGURIDAD, PROCESOS Y ZÓCALOS

AA3. PROGRAMACIÓN DE COMUNICACIONES EN RED

Boj Cobos, Daniel
DAM
05/2023

TEORÍA PREVIA 4

INTRODUCCIÓN A LA CRIPTOGRAFÍA EN LAS COMUNICACIONES 4

DESCRIPCIÓN 4

SEGURIDAD 4

Confidencialidad 4

Integridad 4

Disponibilidad 4

ATAQUES A UN SISTEMA 5

Interrupción 5

Intercepción 5

Modificación 5

Fabricación 5

TIPOS DE ATAQUE 5

Malware 5
Métodos habituales de infección 5

Virus 6
Ejemplos 6

Gusanos 6
Ejemplos 6

Troyanos 6
Ejemplos: 7

Ransonware 7
Ejemplos 7

Spyware 7
Ejemplos 7

Adware 7
Ejemplos 8

Malware sin archivo **8**
Ejemplos 8

Phishing **8**

Denegación de servicio distribuido - DDoS **8**
Ejemplos 9

Diferencia entre malware y ransomware **9**
Ejemplos de ataques de ransomware 10

MÉTODOS CRIPTOGRÁFICOS **10**

Criptografía simétrica **10**
DES (Data Encryption Standard) 11
AES (Advanced Encryption Standard) o Rijndael 11

Criptografía asimétrica **11**
RSA (Rivest, Shamir y Adleman) 12
Funciones Hash criptográficas 12

PROGRAMACIÓN MULTIPROCESO **12**

LIBRERÍA MULTIPROCESSING **12**

MULTIPROCESO **12**

Procesos **12**

Diferencia ente hilos y procesos **13**
Diferencias entre procesos e hilos 13
Diagrama multihilo 14
Diagrama multiproceso 15
Diagrama de ejecución en Sistema 16

Ciclo de vida **17**
Creación 17
Estados 17
Paso de funciones 17
Ejemplo 17
Ejecución secuencial 18

BLOQUEO DE VARIABLES EN PROCESOS E HILOS **18**

HILOS DAEMON Y NON-DAEMON **19**

Non-Daemon **19**

Daemon **19**
Diferencias 20
Ciclo vida hilo non-daemon y Daemon 20
Ciclo de vida de un hilo 21
Identificación de un proceso *daemon* en el terminal de Debian 21

PRÁCTICA: ANÁLISIS Y REIMPLEMENTACIÓN DEL CÓDIGO DE EJEMPLO **21**

Estudio del programa original **22**

Análisis paso a paso del código de ejemplo **22**

Reimplementación del código **23**

Análisis paso a paso del código reimplementado **24**

RECURSOS **26**

AA3.Programación de comunicaciones en red

Teoría previa

Introducción a la criptografía en las comunicaciones

Descripción

La criptografía estudia las técnicas de ocultación de la información para evitar que pueda ser usada por observadores no autorizados, es decir, para protegerla frente al acceso o interactuación por parte de entidades no autorizadas. La información puede protegerse contra ciertos tipos de acciones:

- Protección contra acceso no autorizado
- Protección contra la interceptación de la información
- Protección contra la inserción no autorizada de información
- Protección contra la modificación no autorizado de información

La criptografía se aplica transformando el mensaje original en un mensaje donde la información se ha codificado para que resulte imposible de comprender a no ser que se tengan las claves para su decodificación. La decodificación es el proceso contrario en el que reconvertimos la información del mensaje para que resulte inteligible.

Seguridad

Podemos considerar que un sistema es seguro cuando cumple unos requisitos estipulados:

Confidencialidad

Solo los usuarios autorizados pueden acceder a la información.

Integridad

Solo los usuarios autorizados pueden crear o modificar la información.

Disponibilidad

Los recursos siempre tienen que mantenerse accesibles para los usuarios autorizados.

Ataques a un sistema

Los ataques a nuestro sistema pueden ser al **hardware**, al **software** o a nivel de la capa de datos.

Todos estos ataques pueden clasificarse según su tipología:

Interrupción

Son ataques que impiden el acceso a un recurso dejándolo bloqueado.

Intercepción

Son ataques en los que el atacante obtiene información de los recursos de carácter confidencial.

Modificación

Son ataques en los que el atacante no solo accede a la información, sino que también la modifica o elimina → Afectan a la integridad del sistema.

Fabricación

Ataques que generan información nueva falsa → Afectan a la integridad del sistema.

Tipos de ataque

Los *hackers* y *crackers* buscan explotar las vulnerabilidades de los sistemas para poderse hacer con su control. Para ello, usan diversos tipos de ataque que debemos conocer para poder gestionar nuestras políticas de seguridad.

Malware

Es la denominación general para todo tipo de software ideado para infectar y dañar un sistema y responde al término inglés *malicious software*. Según sus características, podemos distinguir varios tipos de *malware*.

Métodos habituales de infección

Hay programas maliciosos diseñados para aprovechar vulnerabilidades no parcheadas de sistemas desactualizados, *exploits* conocidos de programas o que incluso imitan aplicaciones legítimas para infectar los equipos. A pesar de ello, **el método más común es la interacción de usuarios**, normalmente, con desconocimientos de las medidas de seguridad básicas.

Como veremos a continuación, cada tipo de malware puede presentar unas características propias en cuanto a los vectores de infección y propagación y a su comportamiento.

Virus

Un virus es un código que infecta los archivos de un sistema, pero necesita que el usuario final de este lo ejecute para actuar. Está diseñado para dañar de alguna forma el equipo infectado y se distingue por dos características: Actúa de modo trasparente al usuario y tienen la capacidad de autorreplicarse. La forma más común de infección es ocultándolos en otros archivos para que el usuario los ejecute sin ser consciente.

Ejemplos

- **Stuxnet:** Creado, seguramente, por los gobiernos de EEUU e Israel para atacar a Irán, se extendió durante el 2010. Se propagaba a través de la infección de una memoria USB e infectó a más de 20000 equipos. Estaba diseñado para atacar software de control industrial de Siemens.

Gusanos

Son programas que infectan un sistema y son capaces de autorreplicarse para difundirse por la red a los equipos que tengan acceso al infectado. Su objetivo principal es, precisamente, este aumento de población y, a diferencia de los virus, su propagación no necesita de la interacción del usuario. Su propósito no suele ser el de dañar el equipo, sino que sirven como puertas de entrada para malware de otro tipo, como, por ejemplo, para crear *bots*. Además, a diferencia de los virus, no necesita infectar archivos, sino que se replican directamente entrando en la memoria. Aun así, pueden suponer una ralentización de los sistemas debido al consumo de memoria.

Ejemplos

- **SQL Slammer:** Se usó en 2003 para realizar un ataque DDoS tras infectar a más de 75000 equipos, tenía la característica de poder detectar equipos que no estaban protegidos por antivirus.

Troyanos

Son programas de apoyo que abren puertas traseras en los sistemas para que puedan penetrar e infectarlos otro *malware*, normalmente, usados como entrada para el robo de información. Los *troyanos* no tienen la capacidad de propagarse por sí mismos y dependen de la interacción del usuario. Actualmente, los motivos más usuales de infección incluyen la descarga de archivos P2P, la ejecución de scripts JS en webs poco seguras, *exploits* de aplicaciones desactualizadas y ejecutables escondidos en archivos adjuntos. Hay que comentar también que es un *software* poco sintomático, con lo que no hay que esperar a sospechar de la infección para realizar pruebas regulares a nuestro sistema. Hemos de ser conscientes del peligro real de estos programas que pueden llegar a ser capaces de enviar nuestras pulsaciones de teclado a los ciberdelincuentes.

Ejemplos:

- **Qbot:** Troyano bancario que lleva activo desde el 2007 y se usa para robar credenciales y datos bancarios del usuario.
- **TrickBot:** Originado en 2016 para llevar a cabo el robo de datos bancarios de los usuarios.

Ransonware

Son ataques pensados para encriptar los datos de un sistema con objetivo de que los atacantes soliciten un rescate por ellos ya que pueden decodificarse mediante claves. Es uno de los tipos de ataque que más han aumentado los últimos años, esto se debe a que, para los ciberdelincuentes, resulta bastante sencillo aprovecharse del desconocimiento de muchos usuarios en materia de seguridad o de las grietas que se producen en muchas compañías precisamente por descuidos de estos. El *ransonware* trabaja bloqueando el equipo, encriptando su disco duro, para que, a posteriori, los ciberdelincuentes exijan el pago de un rescate al usuario, de ahí su nombre, a cambio de las claves de desencriptado, que muchas veces no llegan a proporcionar nunca.

Ejemplos

Se analizan de forma pormenorizada en la siguiente sección.

Spyware

Son programas ideados para recoger datos sobre el usuario sin su conocimiento ni consentimiento. Este software puede instalarse y ejecutarse de forma autónoma a través de otro programa que sí está ejecutando el usuario, pero no tienen capacidad de autorreplicado. Aunque puedan parecerse al *adware*, son programas que trabajan a escondidas y ocultando su rastro, recopilando información de carácter privado del usuario y, muchas veces, susceptible de ser usada para infectar o invadir el equipo de este. Para evitar la actividad de estos programas, es importante saber cómo usar un cortafuegos y seguir unas normas estrictas de navegación segura.

Ejemplos

- **CoolWebSearch:** Se aprovecha de los usuarios que aún usan Internet Explorer, un navegador repleto de vulnerabilidades y ya deprecado, para enviar datos privados de estos.
- **Gator:** Se incluye dentro de programas de intercambio P2P y transmite hábitos de navegación del usuario.

Adware

Es software que muchas veces tiene un componente *spyware;* son programas maliciosos que muestran publicidad y pueden recoger información del usuario para realizar perfiles de este,

normalmente para adaptar la publicidad mostrada a sus gustos. El *adware* no tiene una intención perniciosa por sí mismo, por lo que se genera algo de debate entre los especialistas sobre clasificarlo o no como *malware*, aun así, el hecho de que muchos de estos programas intenten aprovecharse del desconocimiento del usuario medio para obligarlo a usar un tipo de software lo acerca al concepto de programa malintencionado. El escenario más común en el que encontramos estos programas es durante la instalación de software gratuito.

Ejemplos

- **Fireball**: En 2017 se descubrió que infectaba a más de 250 millones de equipos. Al infectar un equipo cambia la página de inicio del navegador por la de un buscador falso, Trotus, que incluye anuncios en todas las páginas que visite el usuario.
- **Appearch**: Es otra falsa página de inicio del navegador que se viene incluida en los instaladores de software de descarga gratuita.

Malware sin archivo

Es un tipo de programa que hace uso de programas legítimos para infectar los sistemas. Este comportamiento hace que sea muy difícil de identificar y rastrear ya que no deja archivos ni procesos que puedan ser analizados, es decir, no deja ninguna huella tras de sí. Es uno de los tipos de ataque más recientes ya que empezó a usarse en 2017. Estas infecciones, al no usar archivos intermedios, se instalan directamente en la memoria del equipo y no pasan por el disco duro.

Ejemplos

- **Frodo:** Es un ***malware*** de tipo ***stealth***, se oculta y no deja ningún rastro detectable. Se detectó su primera activación en septiembre de 2022 y se usa para implantar troyanos en los sectores de arranque, por lo que pudo detectarse a través del intento de instalación del troyano.

Phishing

Más que ataques directos, son técnicas que utilizan los atacantes para ocultar su identidad mediante la suplantación y así poder obtener datos privados que más tarde usarán para sus ataques. Suelen obtenerse a través de mails falsos que conducen a webs que suplantan la identidad de webs corporativas, llamadas de teléfono y también SMS y, más recientemente, servicios de mensajería.

Denegación de servicio distribuido - DDoS

Son ataques que bloquean un equipo o sistema a base de realizar un enorme número de peticiones a este. Es un ataque más complejo y suelen ser muy complicados de rastrear. La técnica más usual es la del uso de *botnets**,*** equipos que han sido previamente infectados por *troyanos* o *gusanos*.

Ejemplos

- **Code Red:** Fue un ataque realizado en 2001 que fue capaz de afectar a la Casa Blanca y se aprovechó, una vez más, de equipos sin parches de actualización de seguridad y se propagó a través del servidor web Microsoft IIS.
- **DynDNS**: En 2016, este ataque afectó a directamente a los servidores DNS y dejó sin servicio a más de la mitad de dominios de Internet. Se propagó gracias a las vulnerabilidades de los dispositivos IoT.

Diferencia entre malware **y** ransomware

Solemos generalizar todos estos tipos de programas dentro del término *virus*, pero en verdad, esto es incorrecto y demasiado disperso.

Como se ha visto en las definiciones, ***malware* es el concepto correcto para englobar todo tipo de software ideado para infiltrarse e infectar equipos y sistemas, evidentemente, sin conocimiento ni consentimiento del usuario.** Así, dependiendo de las características del *malware*, podremos clasificarlo en los distintos tipos que hemos descrito.

En general, los programas *malware**,*** como mínimo, consumirán los recursos del sistema produciendo una disminución de su rendimiento. Además, según su propósito, se comportarán con un grado de virulencia distinto; contemplando desde software pensado para enviar información de los hábitos del usuario, hasta *malware* capaz de bloquear un equipo, borrar sus datos o bloquear su acceso.

Dentro de los distintos tipos de *malware*, encontramos el ransomware, **que, como se ha visto, son programas ideados para el secuestro de los datos de un usuario (entiendo como usuario también a las compañías) mediante la encriptación de su disco duro**. Así, este software requiere que el usuario descargue y ejecute algún archivo sin conocimiento de que oculta el software malicioso. Este tipo de ataque es uno de los que más ha aumentado en la actualidad, debido a que son una forma bastante sencilla para los ciberdelincuentes de conseguir dinero fácil y una evidencia de lo necesario que es educar a la población sobre las bases de la seguridad informática.

Por esto motivo, es importante seguir algunas recomendaciones básicas para evitar la infección:

- Mantener el sistema actualizado y al día de todos los parches de seguridad.
- Usar navegadores seguros y actualizados, personalmente, recomiendo usar navegadores anónimos y seguros como Brave o LibreWolf.
- Borrar Adobe Flash si aún está presente en nuestro sistema, es un software muy susceptible que hace tiempo fue deprecado.
- Conocer el uso del cortafuegos y disponer de un antivirus adaptado a nuestras necesidades. Aunque para el usuario común será suficiente con usar el nuevo antivirus nativo que implementa Windows desde Windows 10, es muy importante aprender a usarlo y mantenerlo activo en segundo plano mientras navegamos.
- Revisar el equipo de forma regular con software de detección avanzado como ESET Online Scanner.
- Seguir buenos hábitos de navegación: No descargar software sospechoso o del que desconozcamos su autor o remitente, no navegar por páginas sospechosas, comprobar

los links que recibimos en mensajes y correos, no descargar archivos adjuntos en correos a no ser que estemos seguros de su origen.
- Comprobar los archivos que descarguemos antes de ejecutarlos.

Ejemplos de ataques de ransomware

Durante los últimos años no ha sido poco común ver información sobre ataques importantes con *ransomware* en los medios de comunicación. Uno de ellos es el conocido como **Virus de la Policia Nacional**, que se basa en un *phising* para atacar con un *ransomware* y que, en verdad, no es un único programa sino una familia que actúan con el mismo patrón de extorsión basadas en el programa original *Koler*. El virus se propagó principalmente a través de *spam* y *banners* falsos en páginas de descarga de material multimedia. Una vez infectado el equipo, se muestra un mensaje al usuario indicándole que la Policía Nacional le ha bloqueado el equipo por haber accedido a sitios web ilegales y que debe pagar una multa para desbloquearlo. Evidentemente, los ciberdelincuentes se aprovechan de la vergüenza del usuario ante el hecho del que se le acusa para que decida pagar rápidamente sin pensar en que le están engañando. El principal problema de este ataque fue su rápido factor de propagación.

Otro ataque muy conocido realizado hace poco tiempo fue el del programa **WannaCry**. Un ataque específico a sistemas son SO Windows que exigía un pago mediante bitcoin para rescatar los datos. El ataque se extendió rápidamente a nivel global durante mayo de 2017 debido, sobre todo, al uso de sistemas desactualizados con agujeros de seguridad no parcheados. Se supone que el programa se basa en un software desarrollado por la NSA conocido como *EternalBlue* que fue publicado por un grupo de ciberdelincuentes conocido como *The Shadow Brokers*. Lo más importante sobre este caso en concreto es que Microsoft publicó el parche que solucionaba la falla de seguridad más de dos meses antes del ataque, demostrando la importancia radical de mantener actualizado nuestro sistema. El problema no se produjo únicamente a nivel privado, este *malware* infecto a grandes compañías de todo el mundo, en España, por ejemplo, afecto a Telefónica y al INS, y, a nivel mundial, afectó a más de 150 países.

Por último, hablar de ***CryptoLocker***, un ataque que se extendió durante el 2016 y 2017 basado en *phising* e ingeniería social y que tenía como objetivo atacar a grandes compañías a través de sus empleados. Así, este *malware* penetraba a través de un equipo y luego era capaz de infectar toda la red de la empresa. A pesar de que ya no se considera una amenaza actualmente, se calcula que los autores consiguieron unos 3 millones de dólares mediante la extorsión para conseguir las claves de desencriptado.

Métodos criptográficos

Podemos dividir los métodos en dos grandes bloques, **la criptografía simétrica**, que usa una única clave para el cifrado y el descifrado, y **la criptografía asimétrica**, que usa claves diferentes.

Criptografía simétrica

El sistema receptor y el sistema emisor usan una misma clave que ambos conocen previamente. Esta clave se transmite sin seguridad.

A continuación, se presentan los algoritmos de cifrado:

DES (Data Encryption Standard)

Se desarrollo en EEUU por parte de la NSA para proteger las comunicaciones de las empresas.

Cifra por bloques de una longitud de bits fija y lo transforma en una serie de operaciones en otro bloque de la misma longitud. El tamaño del bloque es de 64bits y la clave igual, pero 8 bits se usan para comprobar la paridad. Se compone de 16 fases, se conoce como esquema Feistel.

Ya no se usa ya que no se considera robusto.

AES (Advanced Encryption Standard) o Rijndael

Utiliza una matriz de 4 x4 bytes. Es cifrado de clave simétrica y de fácil implementación. Además, usa poca memoria. Utiliza una serie de bucles para funcionar → 10 ciclos para claves de 128 bits, 14 ciclos para claves de 256 bits.

Es una clave mucho más robusta ya que tiene una longitud variable.

Criptografía asimétrica

También conocido como de clave pública, es un sistema que usa una pareja de claves. El remitente usa una clave pública del destinatario y solo puede descifrarse el mensaje usando la clave privada que posee el destinatario.

La clave privada solo la posee el destinatario, y la clave pública la pueden poseer todos los usuarios.

Estos sistemas requieren más tiempo de procesado que las claves simétricas, el mensaje cifrado es mayor que el original y las claves deben ser más complejas.

RSA (Rivest, Shamir y Adleman)

Es un algoritmo que se basa en escoger dos números primos grandes elegidos de manera aleatoria y que se mantienen en secreto. Aplicando el uso de coprimos, pueden generarse las fórmulas para el encriptado y desencriptado de los datos.

Funciones Hash criptográficas

Una función Hash tiene una entrada A y una salida completamente distinta, B, que siempre cumple:

- Aunque la entrada A sea distinta, la salida B siempre tiene el mismo tamaño, esto se consigue mediante la compresión del código hash.
- Debe manejar las colisiones. Hay diversos métodos para evitarlo, los más comunes son el lineal y el uso de LinkedLists en cada posición de la tabla de hash.
- Para cada entrada se debe producir una salida única.
- Debe ser rápido y fácil de calcular.
- No hay retorno desde la salida, es decir, es imposible obtener el valor de entrada a través del valor de salida.

El algoritmo más usado es el **MD5**.

Programación multiproceso

Librería Multiprocessing

El módulo *multiprocessing* de Python permite la **creación y administración de procesos hijo desde nuestro programa principal**. Esta librería se incluyó ya en Python 2, aunque muchas veces los desarrolladores carecemos de un conocimiento profundo sobre sus posibilidades y cómo explotarlas.

Multiproceso

Procesos

Podemos identificar directamente un proceso como **un programa en ejecución en nuestro sistema**. Cada vez que ejecutamos un programa Python, este actúa como un proceso principal. Además, Python nos permite la creación de procesos hijo que hereden de nuestro programa *main* y que nos permitirán ejecutar tareas de forma paralela.

El módulo *multiprocessing* controla como se crean y administran estos procesos, adaptándose al Sistema Operativo y arquitectura del equipo, manejándolo todo a través del intérprete de Python. Además, los procesos pueden ejecutarse de forma paralela o de forma secuencial.

Adaptándonos tanto a las necesidades de nuestro programa, como a las particularidades del procesador que lo ejecute.

Diferencia ente hilos y procesos

Es muy importante no confundir los procesos y los hilos. **Un proceso es un programa en ejecución**, **es decir, cada instancia que genere nuestro interprete de Python**. Estos procesos tienen dos funciones simultáneas, contener los recursos de la aplicación y ejecutar sus instrucciones.

Cada proceso tendrá al menos un hilo, el principal y este siempre existirá dentro de nuestro proceso, y todos los hilos que generemos dentro del programa, pertenecerán al proceso. Una vez finalicen todos los hilos que genere el proceso, finalizará el proceso. Así, **un hilo es una secuencia de instrucciones que podemos ejecutar de forma independiente en el programa**, de manera independiente del proceso principal.

Así, **un proceso tendrá uno o más hilos y un programa generará uno o más procesos a través del intérprete de Python.**

En resumen,

- Multiproceso es la ejecución de diversos procesos de forma simultánea.
- Multihilo es la ejecución de diversas tareas, o hilos, dentro de un mismo proceso.

Hay que puntualizar que los procesadores actuales suelen ser de varios núcleos, con lo que permiten el multiproceso, y que cada núcleo es multihilo.

Diferencias entre procesos e hilos

Procesos	**Hilos**
Necesitan más recursos	Son procesos ligeros
No comparten memoria	Comparten memoria
No necesitan sincronización de memoria	Necesitan sincronización de memoria
Pueden existir individualmente	Siempre están ligados a un proceso
Si un proceso desaparece, todos sus hilos mueren con él	Cuando expira un hilo, su pila se puede recuperar, porque cada hilo tiene su propia pila
Pueden existir en diferentes CPU	Deben permanecer en la misma CPU

Diagrama multihilo

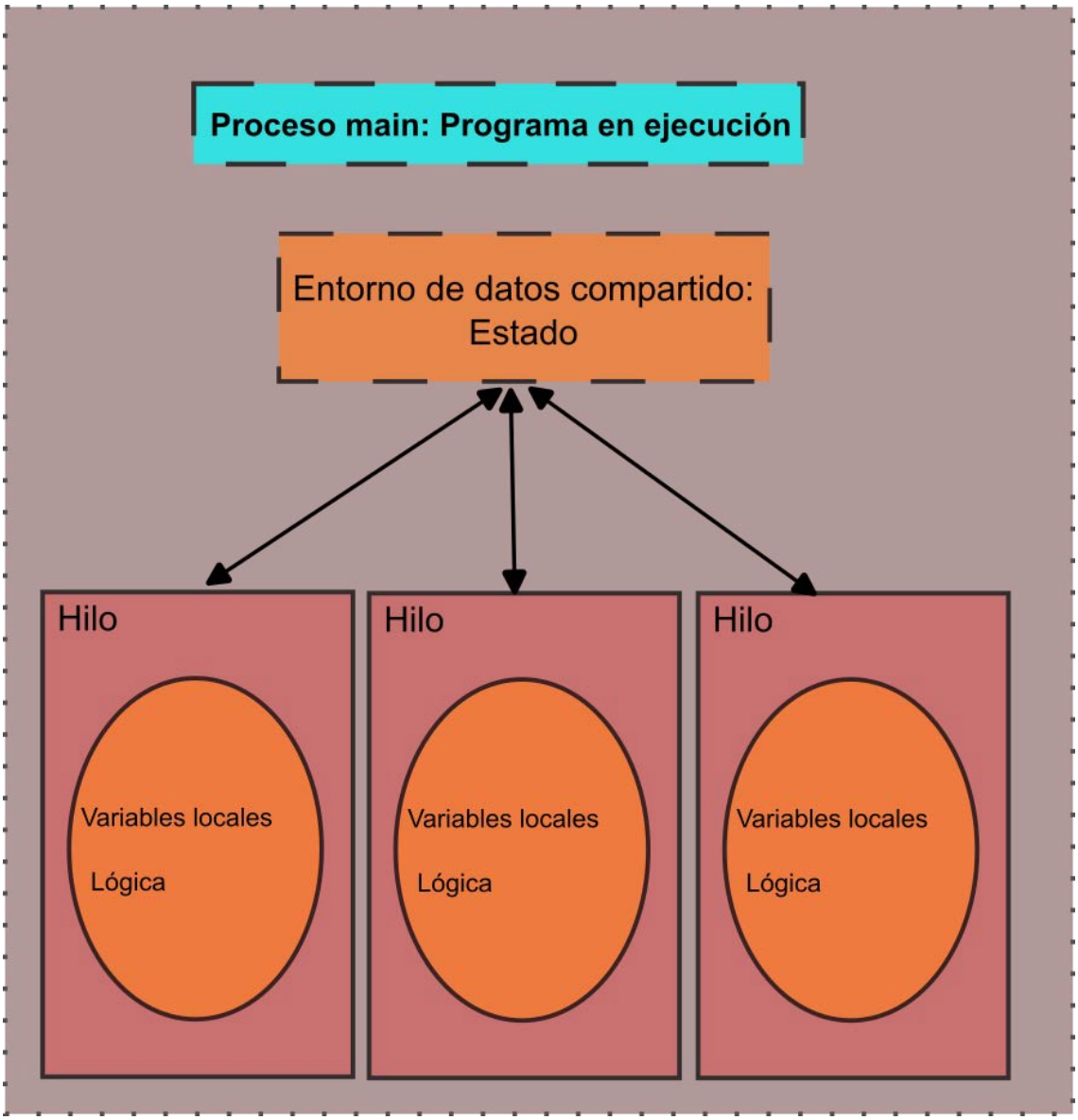


Diagrama multiproceso

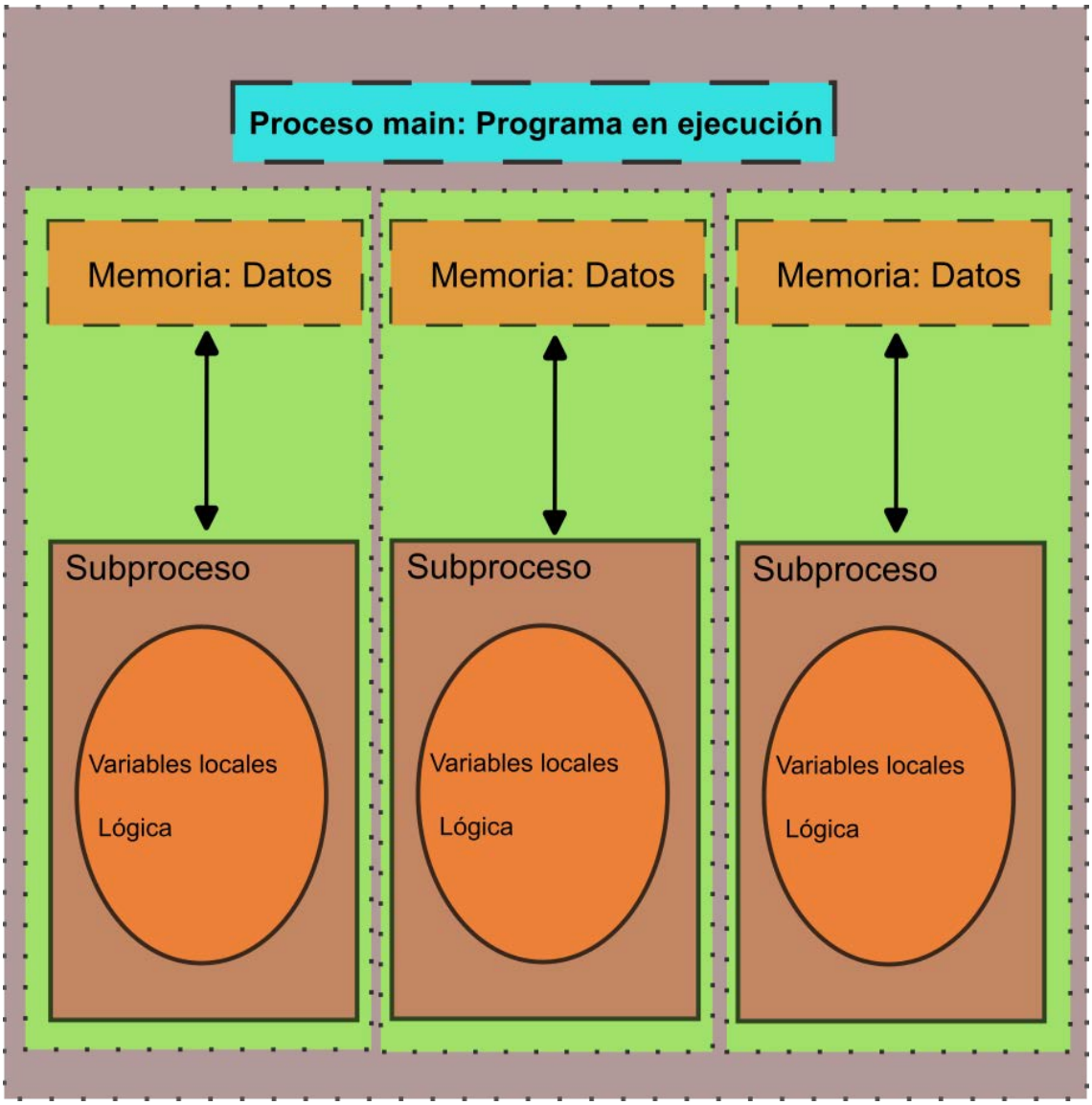
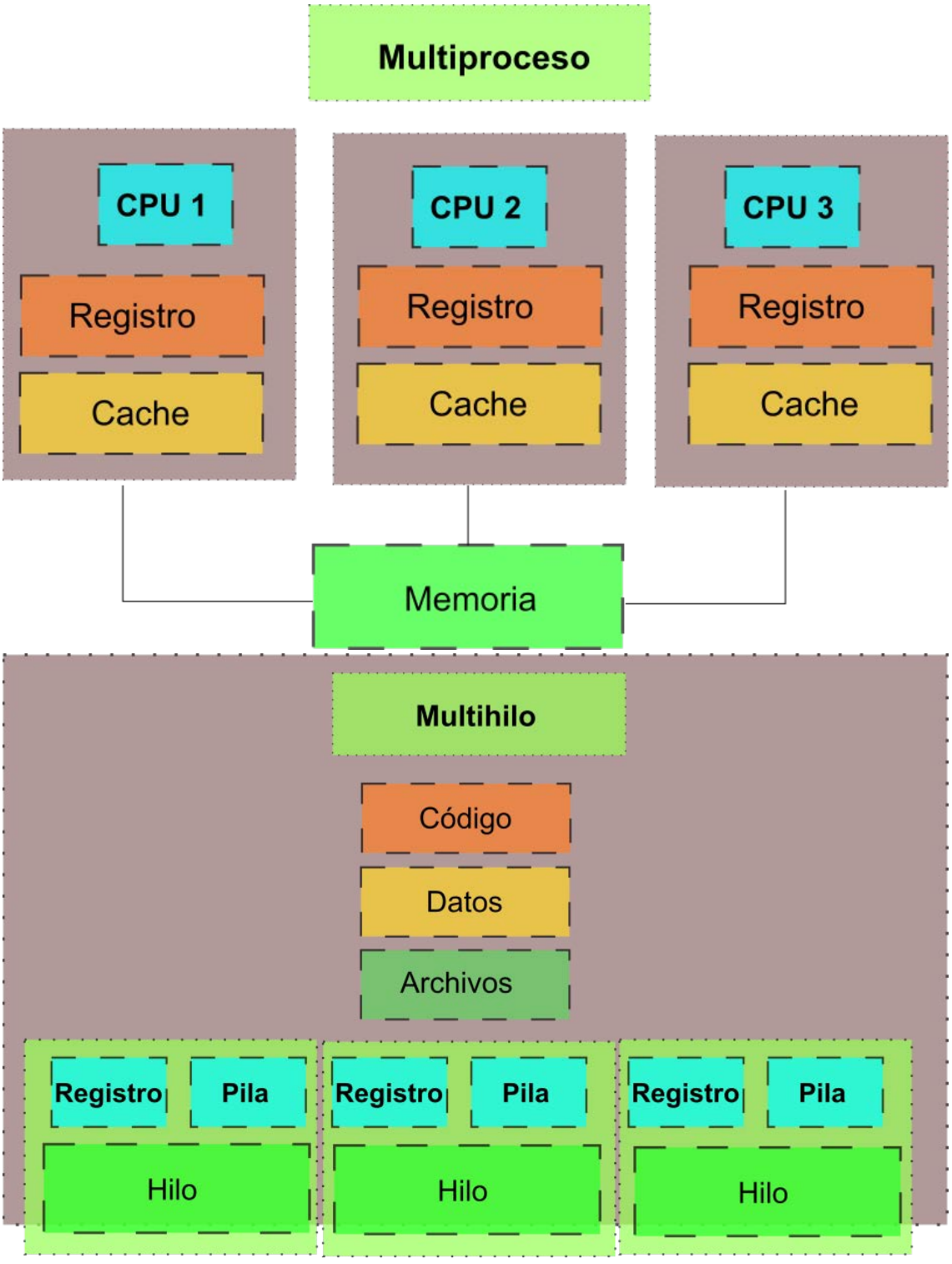


Diagrama de ejecución en Sistema



Ciclo de vida

Creación

Iniciamos el ciclo de vida de un proceso mediante el método: ***multiprocessing.Process()***

Este método instancia un nuevo proceso a través del intérprete que está relacionado, hereda, del proceso principal y captura la referencia a ambos.

Estados

Un proceso puede evolucionar a tres etapas diferentes en su ciclo de vida:

1) **Nuevo proceso hijo** → Iniciamos esta etapa cuando instanciamos el proceso.
2) **Proceso en ejecución** → Usamos el método: ***proceso.Start()*** para iniciar el estado de ejecución. implícitamente, esta acción también pasa a este estado al proceso padre.
   a) **Proceso bloqueado** → Un proceso puedo pasar a un estado de bloqueo cuando se encuentra a la espera de la ejecución de otros procesos, además, nuestro proceso hijo heredará los bloqueos del proceso principal. Los ***locks*** y semáforos sirven para bloquear procesos. Cuando el bloqueo finalice, nuestro proceso volverá al estado de ejecución.
3) **Proceso finalizado** → Un proceso pasa a este estado tanto cuando finaliza su ejecución, como cuando se produzca un error o excepción.
   a) Un proceso nunca finaliza hasta que todos sus hilos e hilos hijos no-demonios han concluido.

Paso de funciones

Para pasar la función que ha de ejecutar a un hilo usamos la siguiente sintaxis durante su instanciación:

```
nombre_proceso = processing.Process(target = función)
```

Además, como con los hilos, hemos de pasar los argumentos del proceso en un argumento independiente y con formo de ***tupla:***

```
nombre_proceso = processing.Process(target = función, args = (arg1,...))
```

Como hemos comentado, el intérprete se encargará de realizar el manejo del proceso adaptándose al sistema y el hardware concretos.

Ejemplo

```
# Importamos la librería
import multiprocessing

# Creamos una función que requiera algo de tiempo de ejecución
def count_primes():
        count = 0
        for i in range(2, 100000000):
```

```
                for x in range(2, i):
                        if(i % x == 0):
                                break
                        else:
                                count++;
        print(f"Proceso hijo: {count}")

# Creamos un proceso
counting_process = processing.Process(target = count_primes)
counting_process.start()
```

Ejecución secuencial

Podemos indicar explícitamente que debe esperarse a la finalización del proceso mediante el método **join()**.

```
counting_process.join()
```

Bloqueo de variables en procesos e hilos

Uno de los objetivos de esta actividad es entender cómo podemos realizar el bloqueo de una variable dentro de un proceso o hilo para mantener la integridad de los datos en nuestros programas. Así, **si no realizamos el bloqueo cuando sea necesario, se producirá lo que conocemos como efecto carrera y este producirá que los resultados de nuestras funciones sean inesperados**.

Veamos a continuación como podemos realizar el bloqueo del estado, recordando que es aplicable tanto a hilos como a procesos:

Una variable compartida por varios multiprocesos puede ver su valor alterado dependiendo de los tiempos y orden de ejecución de los hilos o procesos. Así, un valor puede variar en diferentes ejecuciones del script.

Para mantener el valor de la variable bloqueado para un solo *thread* hay que usar los comando *lock* y *release*.

- *lock* bloquea los valores de una variable hasta la finalización del proceso o hilo.
- *release* libera la variable para que pueda cambiar su estado en otros procesos o hilos.

Primero, tenemos que añadir un nuevo parámetro a nuestras funciones:

A continuación, podemos realizar los bloqueos dentro de estas, pero es muy importante que recordemos implementar el desbloqueo una vez finalizadas las operaciones:

```
def sumar_uno(variable_compartida, locker):
        lock.acquire()
        variable_compartida += 1;
        lock.release()
```

Dentro del *main*, declararemos el locker y lo pasaremos como argumentos de nuestra función cuando creemos el subproceso:

```
resultado = 0
locker = multiprocessing.Lock()
process_a = multiprocessing.Process(target=suma_uno, args=(resultado, locker))
```

Hilos Daemon y Non-Daemon

Non-Daemon

Por defecto, los *threads* son del tipo non-daemon, estos se ejecutan en primer plano, si el proceso finaliza, se espera para su terminación a que todos sus *threads* hayan concluido.

Es decir, si el *thread* principal finaliza, espera a que finalicen el resto de *thread* para cerrarse.

```
import threading
import time

def Worker_A:
        print("Starting A")
        time.sleep(2)
        print("Finishing A")

def Worker_B:
        print("Starting B")
        print("Finishing B")

a = threading.Thread(target = Worker_A)
b = threading.Thread(targe = Worker_B)
a.start()
b.start()
```

A no finaliza hasta que no acaba de ejecutarse *B*.

Usamos los *thread* regulares para tareas **críticas o principales** que normalmente ocupan un espacio concreto de tiempo y no se deben ejecutar de forma mantenida indefinidamente en segundo plano.

Daemon

Si declaramos un *thread* como tipo *daemon*, este se ejecuta en segundo plano, cuando el flujo del programa finaliza, se abortan los *threads* de este tipo, aunque estén en curso.

```
a = threading.Thread(target = Worker_A)
a.daemon = True
b = threading.Thread(targe = Worker_B)
a.start()
b.start()
```

A no finaliza hasta que no se cierra el programa. El programa no devuelve el texto de finalización.

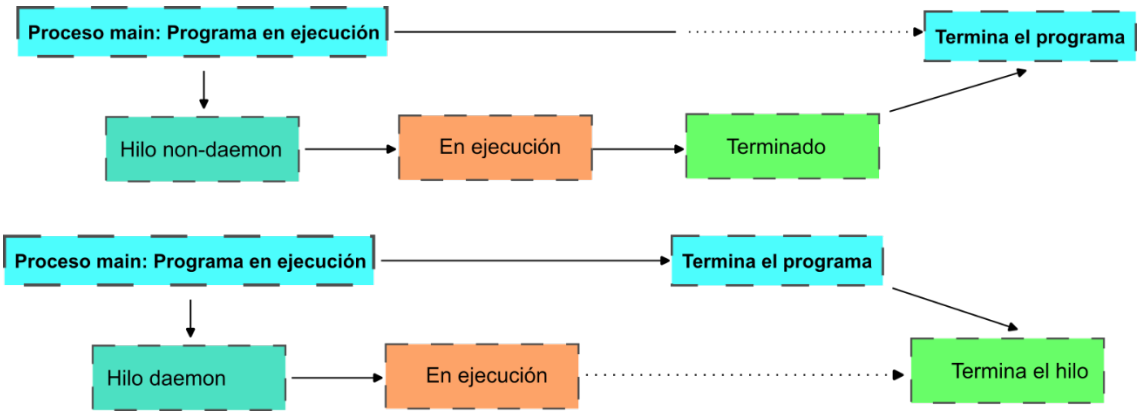
Así, por sus particularidades, estos hilos no deben usarse para ninguna tarea crítica, sino únicamente para tareas que puedan desarrollarse en segundo plano. Escogemos los *thread* de tipo *daemon* cuando necesitamos llevar a cabo tareas en segundo plano, pueden ser **esporádicas**, pero muchas veces usaremos estos hilos para ejecutar tareas **periódicas**, que tengan un **tiempo extenso de ejecución** o, sencillamente, deban permanece en ejecución un tiempo indefinido por necesidades de nuestro programa.

Usualmente, las tareas de los d*aemon* sirven como soporte para los hilos y procesos regulares. Por definición, este tipo de hilo no tiene control sobre su propia finalización, sino que son terminados al finalizar el hilo principal.

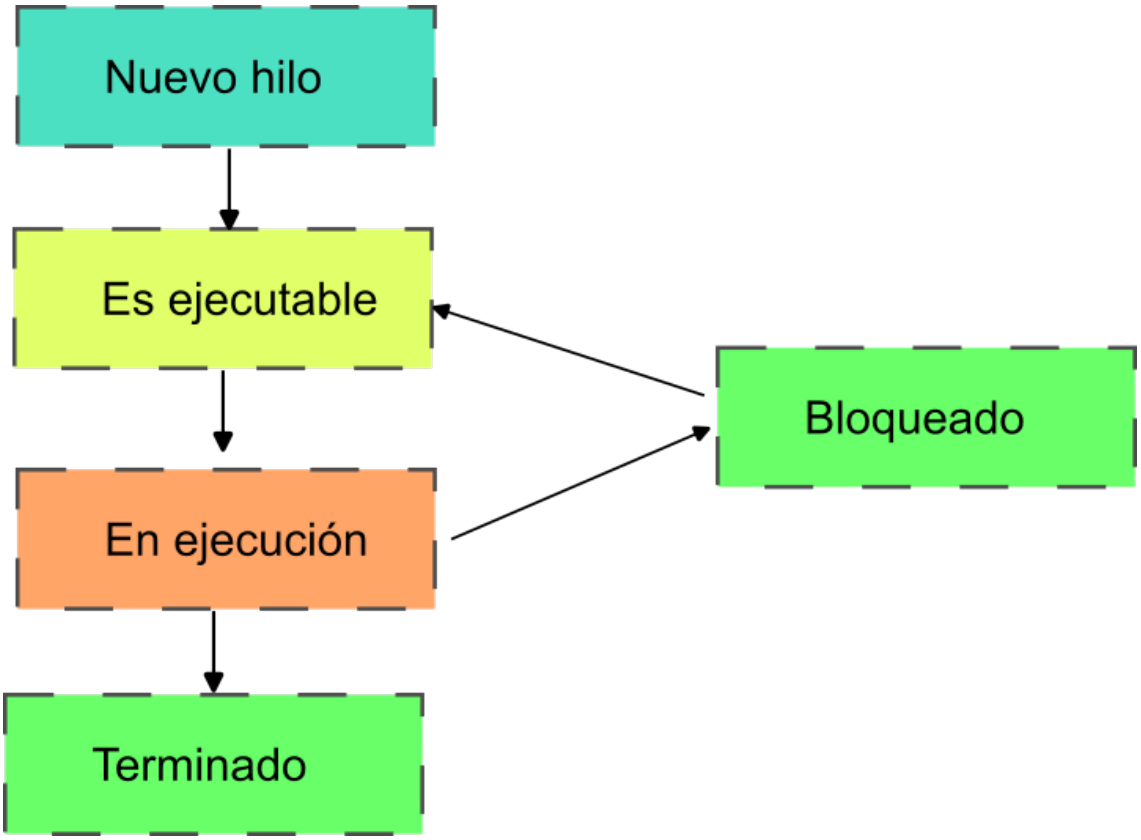
Diferencias

Non-Daemon o hilos de usuario	**Daemon**
El proceso espera a que todos los hilos finalicen.	El proceso no espera a que los hilos finalicen.
El proceso no obliga a los hilos a terminar.	El proceso obliga a los hilos daemon a terminar cuando todos los hilos de usuario han finalizado.
Realizan tareas específicas y críticas.	Realizan tareas de segundo plano.
Tienen alta prioridad.	Tienen baja prioridad.
Su ciclo de vida es independiente.	Su ciclo de vida dependen de los hilos del usuario.

Ciclo vida hilo non-daemon y Daemon

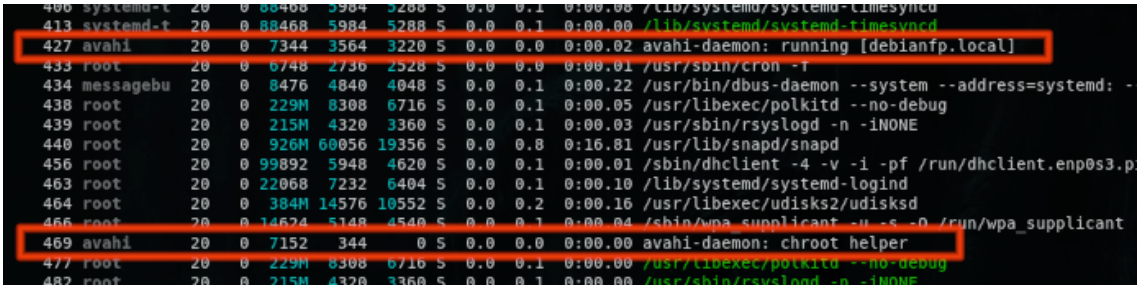


Ciclo de vida de un hilo



Identificación de un proceso *daemon* en el terminal de Debian

Si observamos la identficación de los procesos *daemon*, podremos ver que están creados por el proceso padre:



Práctica: Análisis y reimplementación del código de ejemplo

En el código de esta actividad se trabaja sobre la creación de subprocesos y con el bloqueo de estados de nuestra aplicación durante la ejecución de procesos. Para empezar, hay que decir

que, aunque vayamos a usar subprocesos, las operaciones de bloqueo se pueden realizar igualmente sobre procesos como sobre hilos.

El caso de estudio, como veremos a continuación, se realiza sobre un *script* donde la **falta de bloqueo de los subprocesos hará que, en las variables compartidas por los subprocesos**, **se produzca un efecto que se conoce como *race effect*,** este define los casos donde un sistema debe trabajar sobre un recurso compartido y las ordenes que deben ejecutarse de forma secuencial, acaban ejecutándose con un orden arbitrario y producen una salida no fiable, perdiendo la consistencia de los datos.

Estudio del programa original

El objetivo del *script* de ejemplo es simular la creación de dos subprocesos de nuestro programa, el proceso principal, que representan la entrada y salida de coches en un parking respectivamente. Al analizar el programa, nos vamos a dar cuenta de que realiza esto para que veamos la importancia del bloqueo del estado del programa, asegurando la consistencia de las variables que van a ser usadas por distintos subprocesos para poder mantener la integridad de los datos de nuestro programa.

De este modo, al ejecutar el programa, veremos que, **al no realizar un bloqueo en los subprocesos, los resultados que nos arrojan las funciones ejecutadas por los estos son inconsistentes**. En conclusión, las funciones de nuestra aplicación arrojarán resultados no previsibles, con lo que perderán su fiabilidad y, por ende, nuestro programa no tendrá consistencia de datos.

A continuación, podemos ver como distintas ejecuciones del programa arrojan resultados diferentes.

Análisis paso a paso del código de ejemplo

A continuación, se procede al análisis pormenorizado del código de ejemplo para analizar qué es lo que sucede y porqué se producen los errores:

```
""" _summary_ : Simular la entrada y salida de coches de un parking de 300 plazas. """

# Importamos la librería time para poder simular procesos con un tiempo de ejecución
concreto.
import time
# Importamos la librería multiprocessing para poder crear procesos.
import multiprocessing

""" El principal problema de este código es que la variable coches que van a compartir
los procesos es manipulado por estos sin orden secuencial ya que no están bloqueados,
por lo que puede ocurrir que se produzca una condición de carrera. Es decir, que los
```

```
procesos en ejecución paralela cambien el valor de la variable coches de forma
inesperada. """
def entrada(coches):
    """ _summary_ : Simula la entrada de coches al parking. """
    #simulamos la entrada de 200 coches
    for i in range(200):
        # Simulamos el tiempo de entrada de cada coche.
        time.sleep(0.01)
        # Aumentamos el contador de coches dentro del parking.
        coches.value = coches.value + 1

def salida(coches):
    """ _summary_ : Simula la salida de coches del parking."""
    #simulamos la salida de 200 coches
    for i in range(200):
        # Simulamos el tiempo de salida de cada coche.
        time.sleep(0.01)
        # Disminuimos el contador de coches dentro del parking.
        coches.value = coches.value - 1

# Creamos el proceso principal.
if __name__ == '__main__':
    # Creamos la variable coches que va a ser compartida por los procesos.
    coches = multiprocessing.Value('i', 300)
    # Creamos los procesos de entrada y salida de coches pasándole la variable coches
como argumento.
    entrada_coche = multiprocessing.Process(target=entrada, args=(coches,))
    salida_coche = multiprocessing.Process(target=salida, args=(coches,))

    # Iniciamos los procesos. Como hemos comentado, en estos no se realiza el bloqueo,
con lo que la variable será manipulada de forma desordenada.
    entrada_coche.start()
    salida_coche.start()

    # Esperamos a que los procesos terminen su ejecución. Veremos que el resultado no
es el esperado.
    entrada_coche.join()
    salida_coche.join()

    # Mostramos la variable por pantalla para comprobar que el resultado no es el
esperado.
    print("Coches dentro del parking: ", coches.value)
```

Reimplementación del código

Para asegurar la consistencia de los datos, tenemos que **realizar un bloqueo dentro de las funciones que ejecutarán los subprocesos con el fin de asegurarnos que la manipulación de la variable compartida se produce de forma secuencial**.

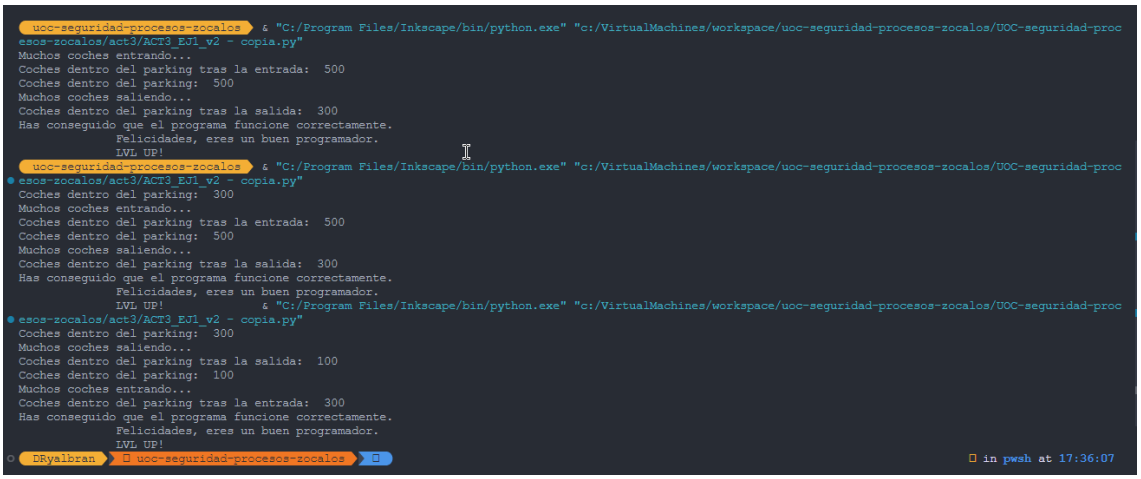
Para ello, deberemos crear una nueva variable *locker* que bloquee las funciones que están ejecutando los subprocesos y obligue a que se realice el acceso y modificación de la variable compartida por estos de forma secuencial y no en paralelo. Será necesario que añadamos un

nuevo parámetro en las funciones para recibir el bloqueo y que creemos la variable en el *main* y se la pasemos a las funciones de los subprocesos cuando los definimos como argumento.

Por último, en los bloques de implementación de las funciones, deberemos usar los métodos de bloqueo y desbloqueo:

- **locker.acquire()** → Para iniciar el bloqueo.
- **locker.release()** → Para liberar el proceso.

De este modo, conseguiremos asegurar la integridad de nuestros datos y tener una salida consistente en nuestro programa. Como vemos a continuación, cada vez que ejecutamos el *script*, obtenemos el mismo resultado esperado:



Análisis paso a paso del código reimplementado

A continuación, se muestra la descripción paso a paso de todos los procesos dentro del código:

```
""" _summary_ : Simular la entrada y salida de coches de un parking de 300 plazas. """

# Importamos la librería time para poder simular procesos con un tiempo de ejecución concreto.
import time
# Importamos la librería multiprocessing para poder crear procesos.
import multiprocessing

""" Refactorizamos el código para evitar el efecto carrera.
Para ello, realizaremos el bloqueo del estado dentro dentro de nuestras funciones para asegurar la consistencia de la variable compartida por los subprocesos. """

""" Añadimos como parámetro un locker para bloquearla. """


def entrada(coches, locker):
    """ _summary_ : Simula la entrada de coches al parking. """
    # Bloqueamos la variable coches.
    locker.acquire()
    # Informamos del número de coches dentro del parking.
    print("Coches dentro del parking: ", coches.value)
```

```
    # Simulamos la entrada de 200 coches
    # Informamos el inicio de la simulación.
    print("Muchos coches entrando...")
    for i in range(200):
        # Simulamos el tiempo de entrada de cada coche.
        time.sleep(0.01)
        # Aumentamos el contador de coches denro del parking.
        coches.value = coches.value + 1
    # Mostramos por pantalla el número de coches dentro del parking.
    print("Coches dentro del parking tras la entrada: ", coches.value)
    # Liberamos la variable coches.
    locker.release()


""" Añadimos como parámetro un locker para bloquearla. """


def salida(coches, locker):
    """ _summary_ : Simula la salida de coches del parking."""
    # Bloqueamos la variable coches.
    locker.acquire()
    # Informamos de los coches dentro del parking.
    print("Coches dentro del parking: ", coches.value)
    # Simulamos la salida de 200 coches
    # Informamos del inicio de la simulación.
    print("Muchos coches saliendo...")
    for i in range(200):
        # Simulamos el tiempo de salida de cada coche.
        time.sleep(0.01)
        # Disminuimos el contador de coches denro del parking.
        coches.value = coches.value - 1
    # Mostramos por pantalla el número de coches dentro del parking.
    print("Coches dentro del parking tras la salida: ", coches.value)
    # Liberamos la variable coches.
    locker.release()


# Creamos el proceso principal.
if __name__ == '__main__':
    # Variable compartida por nuestros procesos.
    coches = multiprocessing.Value('i', 300)
    # Creamos el locker para bloquear la variable coches.
    locker = multiprocessing.Lock()
    # Creamos los procesos de entrada y salida de coches pasándole la variable coches
como argumento.
    entrada_coche = multiprocessing.Process(
        target=entrada, args=(coches, locker))
    salida_coche = multiprocessing.Process(
        target=salida, args=(coches, locker))

    # Iniciamos los procesos. Como hemos comentado, estos no realizan el bloqueo de la
variable.
    entrada_coche.start()
    salida_coche.start()
```

```
    # Esperamos a que los procesos terminen su ejecución. Veremos que el resultado no
es el esperado.
    entrada_coche.join()
    salida_coche.join()

    """ Vamos a jugar un poco con la omnisciencia del programador.
    Así haremos más divertido el proceso. Sabemos que si entran 200 coches y
    salen 200 coches, debemos mantener los 300 coches dentro del parking.
    Así que comprobaremos que los datos son correctos y no se ha
    producido el efecto carrera. """
    if coches.value == 300:
        print("""Has conseguido que el programa funcione correctamente.
              Felicidades, eres un buen programador.
              LVL UP!""")
    else:
        print("""Has conseguido que el programa funcione incorrectamente.
              Debes mejorar tus habilidades de programación.
              LVL DOWN!""")
```

Recursos

Python Multiprocessing: The Complete Guide. (2023, 16 marzo). Super Fast Python. https://superfastpython.com/multiprocessing-in-python

Hola Mundo. (s. f.). Hola Mundo. https://academia.holamundo.io/courses/take/ultimate-python

How to Use Daemon Threads in Python. (2022, 11 septiembre). Super Fast Python. https://superfastpython.com/daemon-threads-in-python/

Frodo | F-Secure Labs. (s. f.). https://www.f-secure.com/v-descs/frodo.shtml

McAfee. (2020). ¿Qué es malware? *McAfee*. https://www.mcafee.com/es-cl/antivirus/malware.html

Andrés, R. (2020, 5 enero). Los 7 mayores ataques DDoS de la historia de Internet. *Computer Hoy*. https://computerhoy.com/listas/tecnologia/mayores-ataques-ddos-historia-internet-555187

¿Qué es el ransomware WannaCry? (2023, 19 abril). www.kaspersky.es. https://www.kaspersky.es/resource-center/threats/ransomware-wannacry

Tipos de malware y ejemplos. (2023, 19 abril). www.kaspersky.es. https://www.kaspersky.es/resource-center/threats/types-of-malware

El virus de la Policía Nacional ha vuelto. (s. f.). https://www.ccn-cert.cni.es/ca/gestion-de-incidentes/lucia/23-noticias/1791-el-virus-de-la-policia-nacional-ha-vuelto.html

buzonuv@uv.mx. (s. f.). *Conocimientos generales: ¿Qué es el malware y cómo se clasifica? – Seguridad de la información*. https://www.uv.mx/infosegura/general/conocimientos_virus-2/