

UT 1 - Multiproceso

Programación de Servicios y Procesos
Curso 2024-25

Profesor: Agustín González-Quel

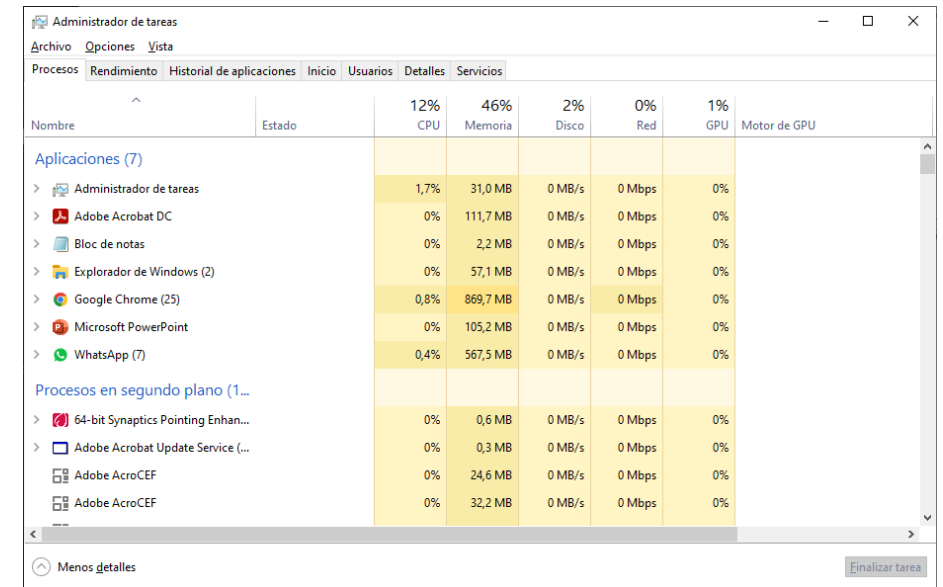
Programas y Procesos

Programa / Aplicación / App:

- Conjunto de instrucciones que resuelve una tarea específica en un ordenador.

• Proceso:

- Programa en ejecución.
- Se compone de
 - Instrucciones
 - Datos
 - Contexto / Registros del microprocesador.
 - Otros datos: prioridad, etc



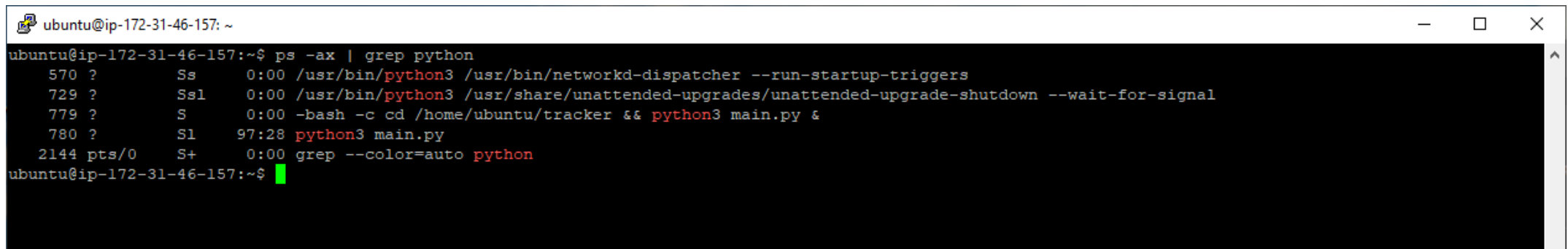
Administrador de tareas

Archivos Opciones Vista

Procesos Rendimiento Historial de aplicaciones Inicio Usuarios Detalles Servicios

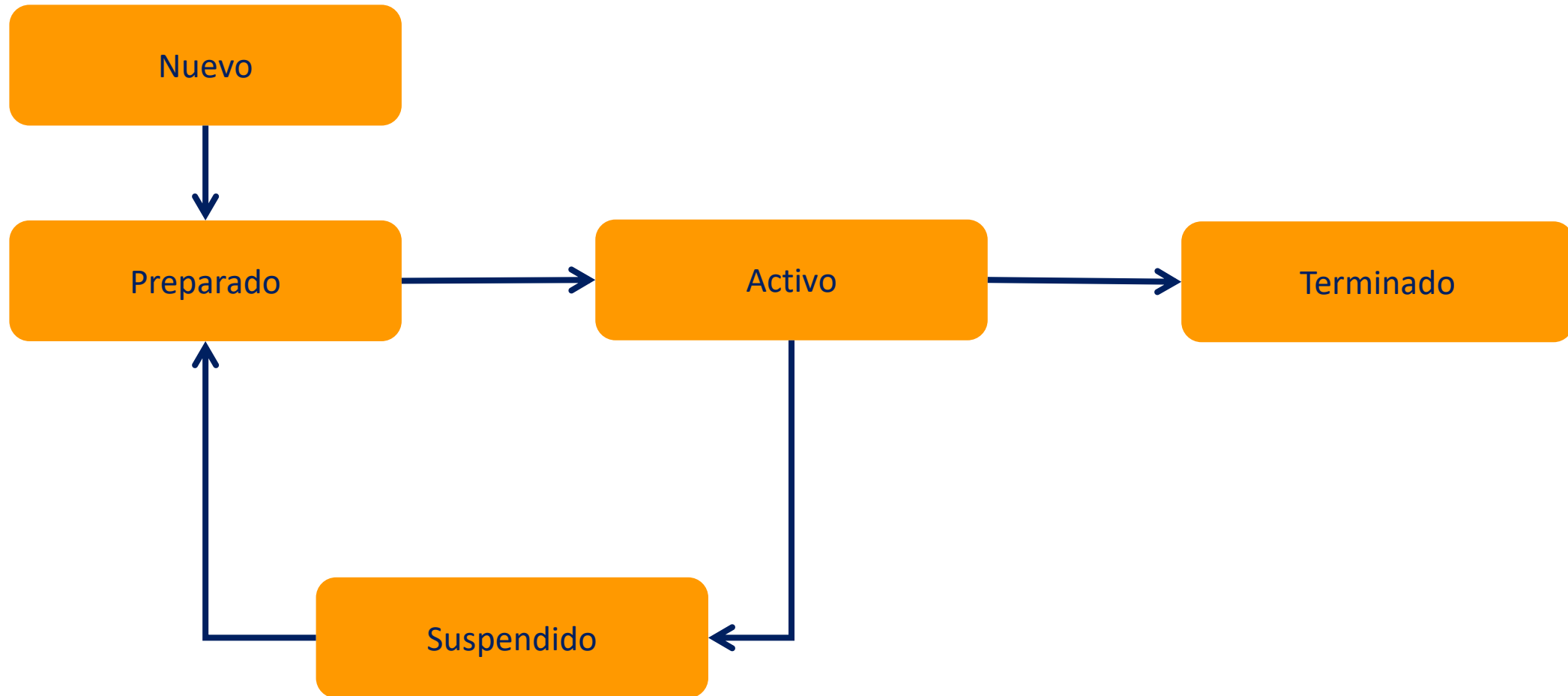
Nombre	Estado	12% CPU	46% Memoria	2% Disco	0% Red	1% GPU	Motor de GPU
Aplicaciones (7)							
Administrador de tareas		1,7%	31,0 MB	0 MB/s	0 Mbps	0%	
Adobe Acrobat DC		0%	111,7 MB	0 MB/s	0 Mbps	0%	
Bloc de notas		0%	2,2 MB	0 MB/s	0 Mbps	0%	
Explorador de Windows (2)		0%	57,1 MB	0 MB/s	0 Mbps	0%	
Google Chrome (25)		0,8%	869,7 MB	0 MB/s	0 Mbps	0%	
Microsoft PowerPoint		0%	105,2 MB	0 MB/s	0 Mbps	0%	
WhatsApp (7)		0,4%	567,5 MB	0 MB/s	0 Mbps	0%	
Procesos en segundo plano (1...)							
64-bit Synaptics Pointing Enhanc...		0%	0,6 MB	0 MB/s	0 Mbps	0%	
Adobe Acrobat Update Service (...)		0%	0,3 MB	0 MB/s	0 Mbps	0%	
Adobe AcroCEF		0%	24,6 MB	0 MB/s	0 Mbps	0%	
Adobe AcroCEF		0%	32,2 MB	0 MB/s	0 Mbps	0%	

Menos detalles Finalizar tareas



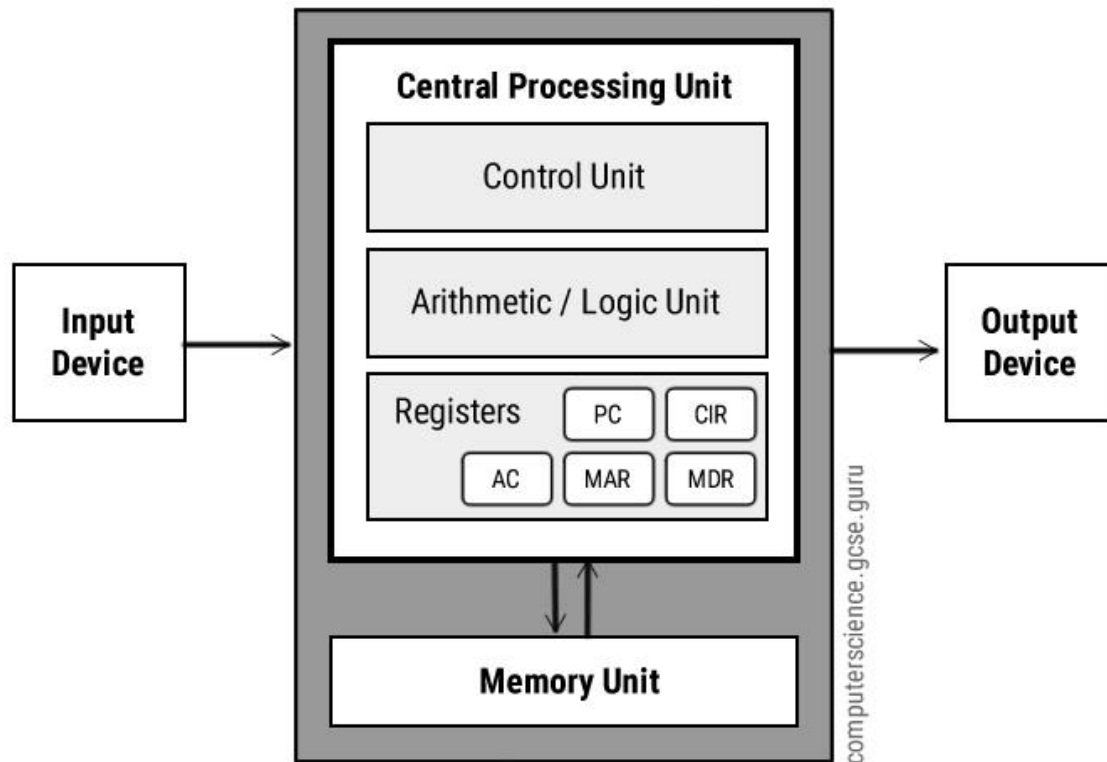
```
ubuntu@ip-172-31-46-157: ~  
ubuntu@ip-172-31-46-157:~$ ps -ax | grep python  
570 ?        Ss        0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers  
729 ?        Ss1       0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal  
779 ?        S         0:00 -bash -c cd /home/ubuntu/tracker && python3 main.py &  
780 ?        S1        97:28 python3 main.py  
2144 pts/0    S+        0:00 grep --color=auto python  
ubuntu@ip-172-31-46-157:~$
```

Estados de un proceso

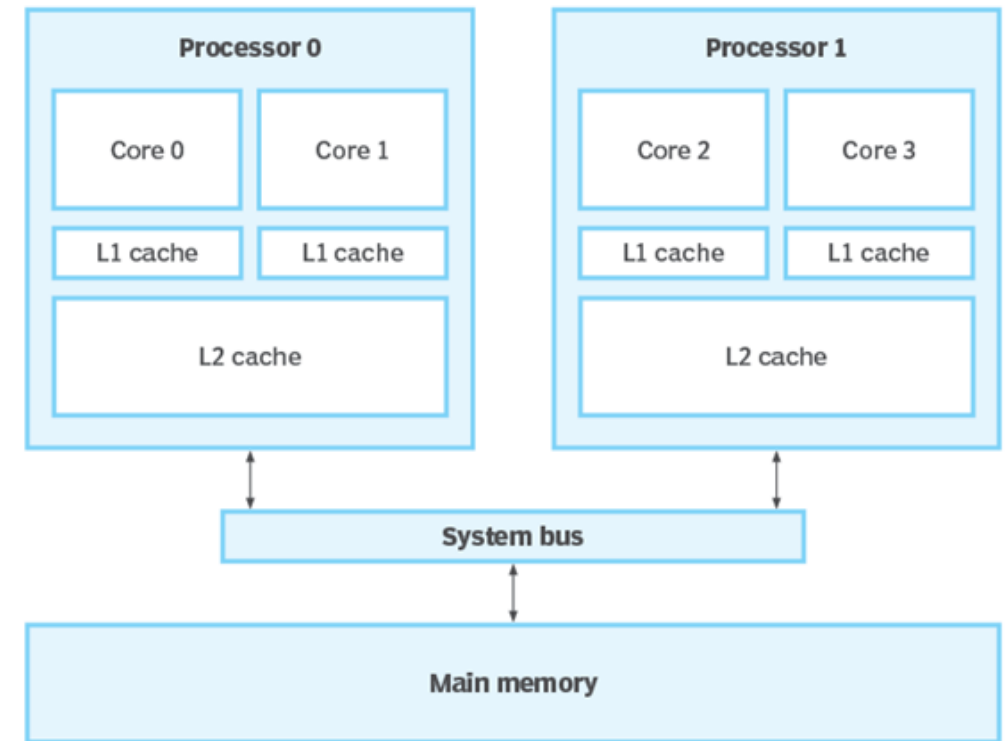


Estructura de un procesador

Arquitectura Von Neumann



Procesador con varios cores



Programación concurrente vs paralela

Concurrente

- El sistema **sólo ejecuta un proceso en cada momento.**
- El S.O. selecciona en cada momento uno de los procesos que deben ejecutarse.
- La percepción del usuario es que varios procesos se ejecutan a la vez
- Cada cambio de proceso requiere un movimiento de toda la información de contexto: datos, posición del ejecutable, etc.

Paralela

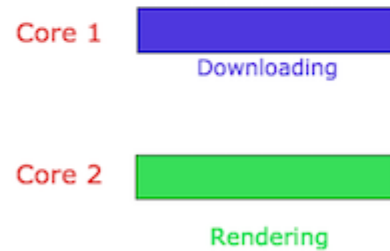
- El sistema ejecuta **varios procesos en un mismo instante de tiempo.**
- Varios núcleos en un mismo procesador o varios procesadores.
 - Cada uno de ellos ejecuta un proceso en el mismo instante de tiempo.
 - Cada núcleo tiene registros para almacenar la información de contexto.
- El S.O. planifica los procesos de cada núcleo.

Gráficamente

Concurrency



Parallelism



Cómo se consigue concurrencia / paralelismo

Procesador

- Número de cores (núcleo): Cada core ejecuta una instrucción en cada momento.

Sistema Operativo

- Los S.O. modernos son multiproceso.

Lenguaje de programación

- Dependerá de cómo se construya cada programa para aprovechar las capacidades de la máquina
- Existen mecanismos en lenguajes como C, Java, Python, etc. Para aprovechar los recursos de la máquina.

Temas a considerar

- Tiempo asignado a cada segmento de proceso, ni muy grande ni muy pequeño.
- Tiempo de cambio de contexto
- Bloqueo – dependencia entre procesos
- Planificación (Scheduler del SO): Se utilizan diferentes estrategias
 - FCFS (First Come First Served)
 - Prioridad
 - Short Job First. Problema, no se suele saber a ciencia cierta.
 - Round Robin
 - Combinación de los anteriores.
- Por qué entra o sale un proceso de la CPU
 - Se agota su tiempo asignado
 - Termina
 - Se suspende por espera en E/S o interrupción del sistema
 - Queda a la espera de un evento (sincronización)

En un sistema con varios cores ...

ACTIVIDAD:

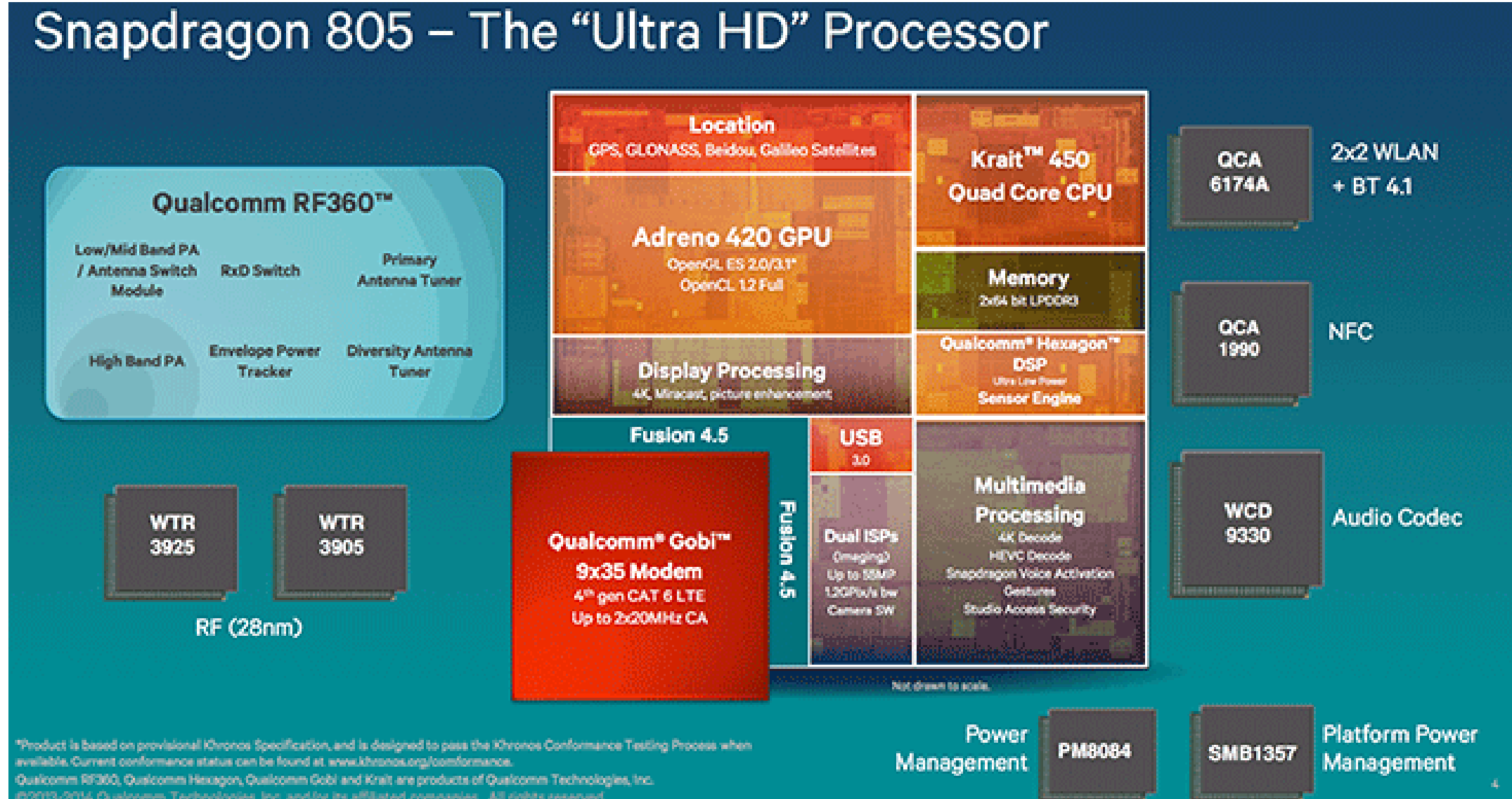
Menú: Filete a la plancha con ensalada y patatas fritas.

Pasos:

- Lavar la lechuga y el tomate (150 s)
- Trocear los tomates (90 s)
- Trocear la lechuga (120 s)
- Aliñar y mezclar (30 s)
- Freír el filete por un lado (120 s)
- Darle la vuelta al filete y freírlo por el otro lado (150 s)
- Pelar patatas (240 s)
- Freír patatas (900 s)



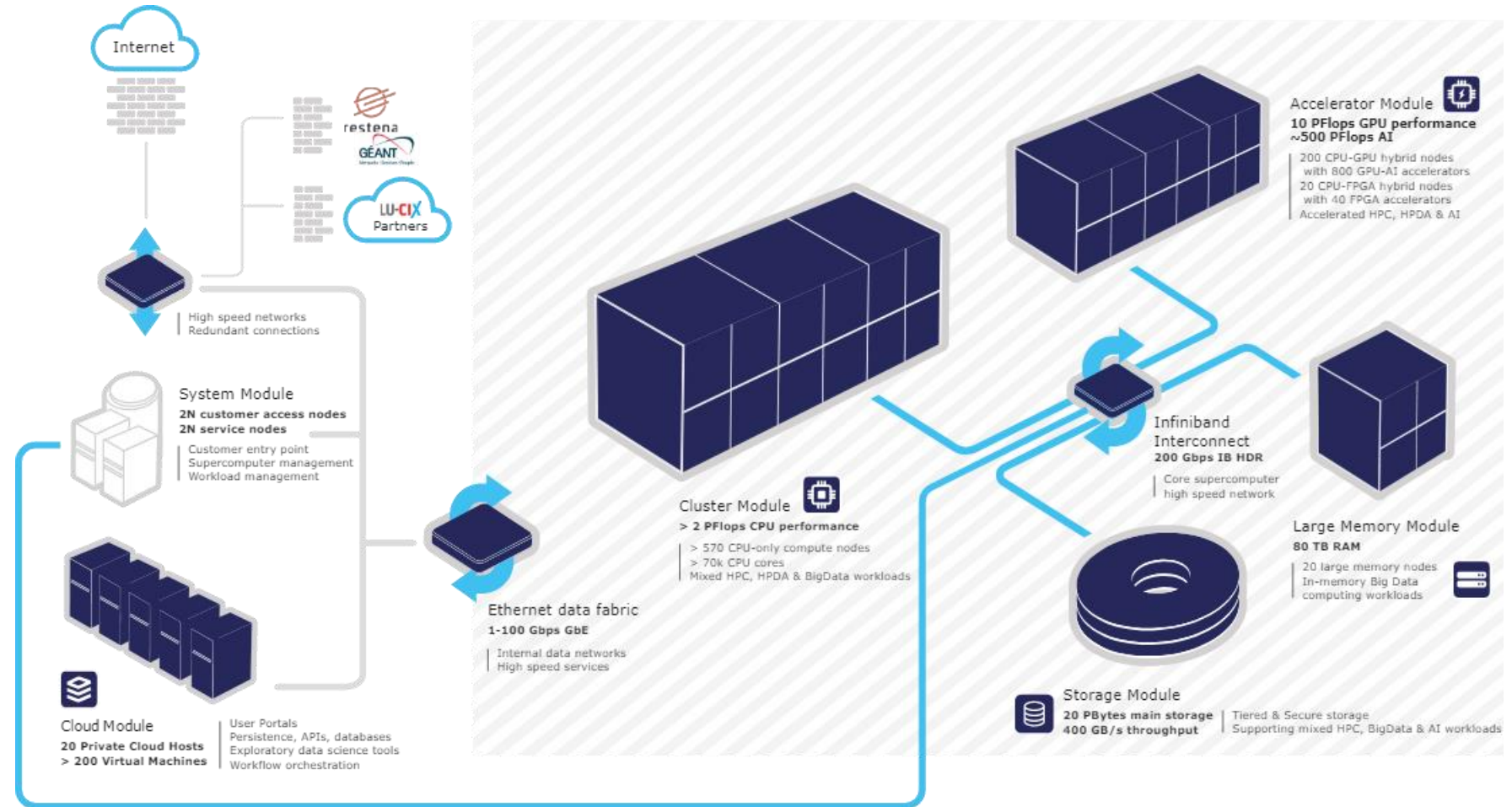
Arquitectura de un teléfono móvil



Ambos comparten la arquitectura



- <https://www.ibm.com/thought-leadership/summit-supercomputer/>



Resumen hasta la fecha

- Qué es un proceso
- Estados por los que pasa un proceso
- Tipos de ejecución
 - Concurrente
 - Paralela
- Procesadores multi-core
- Cómo se gestionan procesos
 - Asignación de recursos
 - Bloqueo entre procesos

Conocimientos teóricos

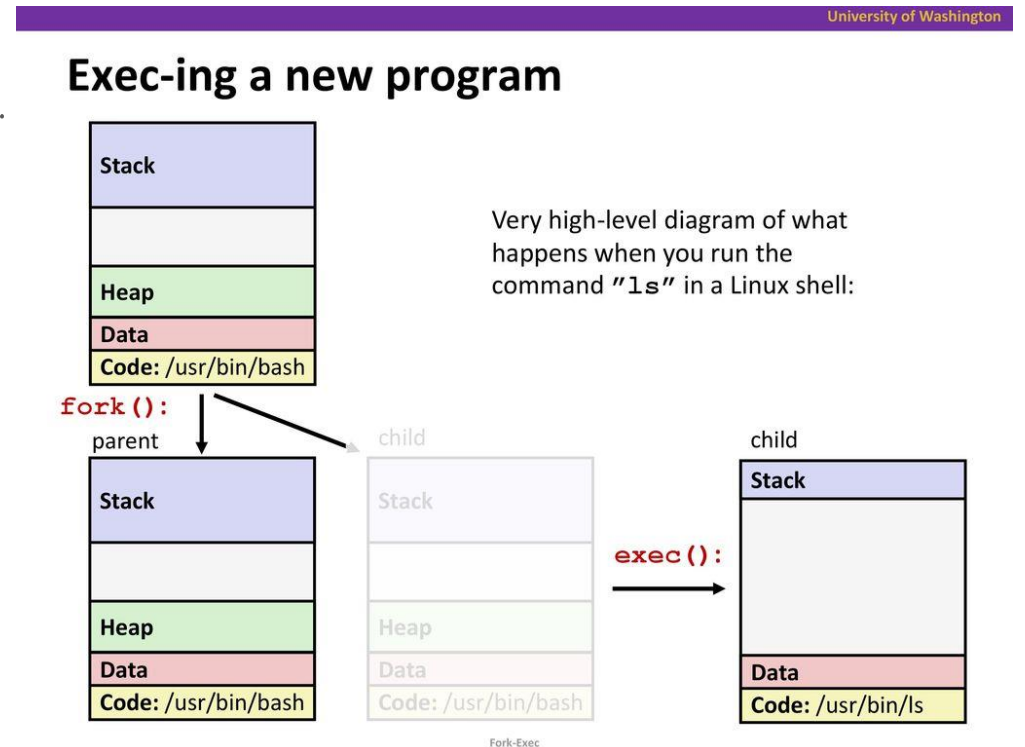
Sabemos responder a todas estas preguntas y si no es así repasamos la presentación.

Operaciones con procesos

1. Clonar un proceso: lanzar un proceso terminando el actual
2. Crear un proceso a partir de una función
 - Esperar a que acabe para continuar el flujo
 - Seguir con el programa de forma independiente
3. Crear un proceso desde un ejecutable
 - Esperar a que acabe para continuar.
 - Seguir con la ejecución independiente
4. Operaciones con procesos del sistema
 - Ver los procesos
 - Comprobar sus propiedades
 - Matar un proceso
5. Comunicación entre procesos

Independencia de procesos

- Cada proceso es una entidad independiente del Sistema Operativo
- Tiene por tanto su propio área de memoria a la que no podemos acceder desde otro proceso
- Si necesitamos compartir información será preciso un elemento externo: bases de datos, ficheros, colas de mensajes, sockets/websockets, etc.



Fuente: <https://github.com/lxeoz/Timer-Old-Implementation/blob/main/Procesos/Procesos.md>

1 - Clonar un proceso terminando el actual

- Puede hacerse con la librería os.
- Lanza un proceso que sustituye al actual que muere.

OS

<https://docs.python.org/3/library/os.html>

- Muy buena para manejar temas del SO como ficheros, carpetas y demás.
- Menos recursos para multiproceso. Permite arrancar procesos

Funciones

```
os.getpid() # Id de proceso
os.getppid() # Id del proceso padre
os.execvp() # Ejecuta un nuevo proceso (hay varias versiones)
os.exit() # Termina el proceso y devuelve un código al SO
```

Ejemplo: 01 - OS-lanzoExterno.py

```
os.execvp('notepad.exe', ['notepad.exe', 'C:/Users/profesor/Documents/mi fichero.txt'])
```

Fork de procesos

- En los sistemas operativos multitarea, los procesos necesitan una forma de crear nuevos procesos, por ejemplo, para ejecutar otros programas.
- La forma más extendida de hacerlo en sistemas tipo Unix (Debian, Ubuntu, Suse, Red Hat, etc.) es mediante el comando fork
 - Fork crea un espacio de direcciones independiente para el proceso hijo.
 - El proceso hijo arranca con una copia exacta de todos los segmentos de memoria del proceso padre.

```
import os
```

```
newpid = os.fork()
```

En el proceso hijo, fork devuelve 0.

Ejemplo: 01 - OS-fork.py

Importante: en MS Windows, no funciona.

2 - Crear un proceso a partir de una función

Multiprocess

multiprocessing: Más compleja y completa

<https://docs.python.org/3/library/multiprocessing.html>

Tiene varias clases, pero usaremos solamente Process

Funciones

```
p = Process(target=hijo)  # crea un nuevo proceso que ejecutará la func hijo
p.start()                # Arranca el proceso
p.join()                 # Detiene el flujo hasta que vuelve el proceso.
```

Ejemplo:

```
p = multiprocessing.Process(target=fun, args=...)
p.start()
```

Nota: args espera una tupla con los parámetros de entrada de la función hijo. Si solo tiene un parámetro, la tupla terminará en , → (4,)

2 – Creación de proceso a partir de función – ejemplo

Se arranca un proceso que ejecuta una función de nuestro programa

- Si queremos detener el proceso padre hasta que acabe el proceso hijo tenemos el método `join()`

En el momento de crear el proceso se hace una copia de memoria.

- Hay dos formas de hacerlo:
 - Spawn: crea un espacio de memoria limpio para el nuevo proceso de forma independiente. Es la opción por defecto en Linux y MacOS y la única disponible en Windows.
 - Fork: Copia el espacio de memoria en ese momento del proceso padre. Es la opción por defecto en Linux y no está disponible en Windows.

<https://docs.python.org/3/library/multiprocessing.html#contexts-and-start-methods>

```
import time
import multiprocessing as mp
import os

def hijo(num):
    c1 = mp.get_context()
    global valor
    print("Hijo antes de modificar: ", valor)
    valor = num
    print("Hijo: ", valor)
    time.sleep(num)
    print("Salgo del proceso hijo {}".format(os.getpid()))
    os._exit(0)

valor = 0
def main():
    global valor
    valor = -5
    p1 = mp.Process(target=hijo, args=(3,))
    p1.start()
    print("Proceso creado {}".format(p1.pid))
    #p1.join()
    print("Padre: ", valor)
    p2 = mp.Process(target=hijo, args=(1,))
    p2.start()
    print("Proceso creado {}".format(p2.pid))
    #p2.join()
    print("Padre: ", valor)

    print("Salgo del proceso padre {}".format(os.getpid()))

if __name__ == '__main__':
    mp.set_start_method('spawn')
    main()
```

3 - Crear un proceso desde un ejecutable del sistema

subprocess:

<https://docs.python.org/3/library/subprocess.html>

Lanzamiento de procesos con bloqueo: función `run`
`subprocess.run()`

- El proceso padre queda bloqueado hasta que el hijo acaba
- Hay que protegerse con try-except en caso de error

```
import subprocess
import time

def CrearProceso():
    try:
        proc = subprocess.Popen('notepad.exe')
        return proc
    except subprocess.CalledProcessError as e:
        print(e.output)
```

```
p = CrearProceso()
time.sleep(5)
```

3 - Crear un proceso desde un ejecutable del sistema

subprocess:

<https://docs.python.org/3/library/subprocess.html>

Lanzamiento de procesos sin bloqueo: Clase Popen y communicate

```
proc = subprocess.Popen('notepad.exe')  
subprocess.communicate() # Recoge la salida de un proceso.
```

- communicate bloquea hasta que no acaba el proceso llamado
- En ocasiones interesa protegerse con try-except.

3 - Crear un proceso desde un ejecutable del sistema: Recogiendo salida

- Recoger stdout, stderr → subprocess (Ejemplo)

```
import subprocess
import time

# Vigilo cada 1 segundos si un servicio Web está arriba
# codecs: https://docs.python.org/3/library/codecs.html#standard-encodings

servicio = "www.madrid.org"
error = False

while not error:
    try:
        subproc = subprocess.Popen(["ping", servicio, "-n", "1"], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        (standardout, standarderr) = subproc.communicate()
        salidaStr = standardout.decode("cp858")
        salidaErr = standarderr.decode("cp858")
    except:
        error = True
        salidaStr = "Error en comando"
    print(salidaStr)
```

`Popen.communicate(input=None, timeout=None)`

Interact with process: Send data to stdin. Read data from stdout and stderr, until end-of-file is reached. Wait for process to terminate and set the `returncode` attribute. The optional `input` argument should be data to be sent to the child process, or `None`, if no data should be sent to the child. If streams were opened in text mode, `input` must be a string. Otherwise, it must be bytes.

`communicate()` returns a tuple `(stdout_data, stderr_data)`. The data will be strings if streams were opened in text mode; otherwise, bytes.

Note that if you want to send data to the process's stdin, you need to create the Popen object with `stdin=PIPE`. Similarly, to get anything other than `None` in the result tuple, you need to give `stdout=PIPE` and/or `stderr=PIPE` too.

<https://docs.python.org/3.10/library/subprocess.html>

4 – Operaciones con procesos del sistema

Psutil: Información sobre procesos en ejecución y otros parámetros del sistema en funcionamiento.

<https://psutil.readthedocs.io/en/latest/>

- Ver procesos en ejecución
- Detener procesos
- Temperaturas del hardware
- Duración de la batería (`battery = psutil.sensors_battery()`)

Funciones

```
p.process_iter()
p.is_running()
p.kill() / p.suspend() / p.resume()
p.name() / p.exe() / p.nice()
p.openfiles()
p.parent()
```

4 – Operaciones con procesos del sistema, ejemplo

Identificar procesos del sistema y hacer operaciones con ellos.

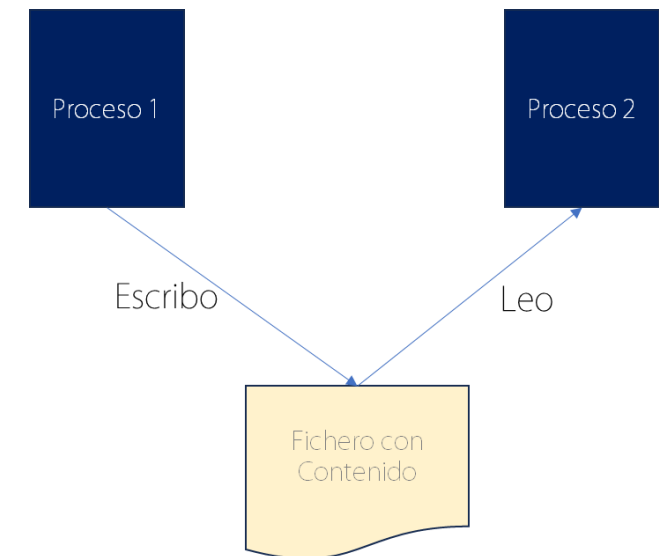
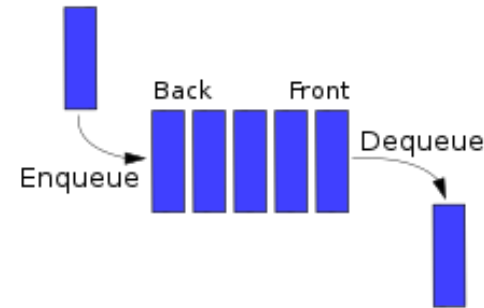
```
import psutil
import time

time.sleep(1)
for p in psutil.process_iter():
    if p.is_running():
        if p.name() == "Notepad.exe":
            print("Encontrado Notepad")
            while p.is_running():
                p.suspend()
                print("Suspendo")
                time.sleep(10)
                p.resume()
                print("Reactivo")
                time.sleep(5)
```

5 – Comunicación entre procesos

Existen diversos mecanismos de comunicación entre procesos

- E/S estándar: stdout/stderr
 - Clase subprocess
- Colas de mensajes
 - Clase Queue de la librería multiprocessing
- Ficheros
 - Visto en ejemplos en clase
- Sockets / websockets
 - Los veremos en temas posteriores



Multiprocessing - Colas

Funciones para el uso de colas entre procesos

From multiprocessing import Queue

- Creación de la cola
`miCola = Queue(maxsize= Tamaño máximo)`
- Comprobar si está vacía/llena: devuelve True-False
`miCola.empty()`
`miCola.full()`
- Añadir elemento: añade un elemento a la cola
`miCola.put(elemento)`
- Obtener elemento: Devuelve el primer elemento de la cola y lo elimina de la misma
`miCola.get()`

Ejemplo de uso de una Cola

```
from multiprocessing import Process, Queue
import os
import time

def leo(q):
    end = False
    cadena = ''
    while not end:
        texto = q.get()
        if texto == '--':
            end = True
        else:
            cadena += texto
            print (cadena)
    print("Salgo del proceso hijo {}".format(os.getpid()))
    os._exit(0)
```

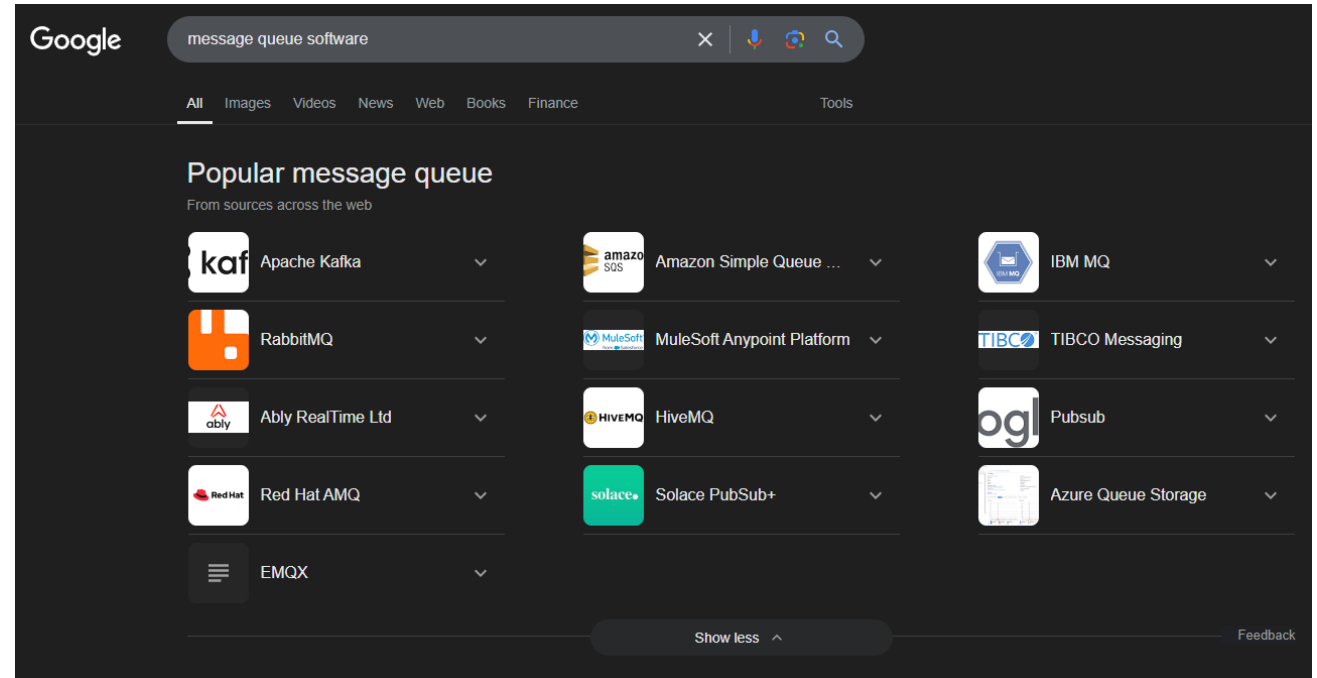
```
def main():
    myQ = Queue(maxsize=1000)
    end = False

    rp = Process(target=leo, args=(myQ, ))
    rp.start()
    #rp.join()

    while not end:
        line = input("Introduce mensaje: ")
        if line == '--':
            end = True
        myQ.put(line)
        time.sleep(3)
        print("Salgo del proceso padre {}".format(os.getpid()))

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

Sistemas comerciales de colas de mensajes



- https://en.wikipedia.org/wiki/Message_queue
- <https://icepanel.medium.com/top-6-message-queues-for-distributed-architectures-a3cbabf08993>
- <https://www.rabbitmq.com/>
- <https://kafka.apache.org/>

Fin UD1 Procesos