**1.3. TEMA: Monitorización**

**1.3.1. Introducción:**

4 elementos que observar para asegurar el buen funcionamiento de una aplicación:

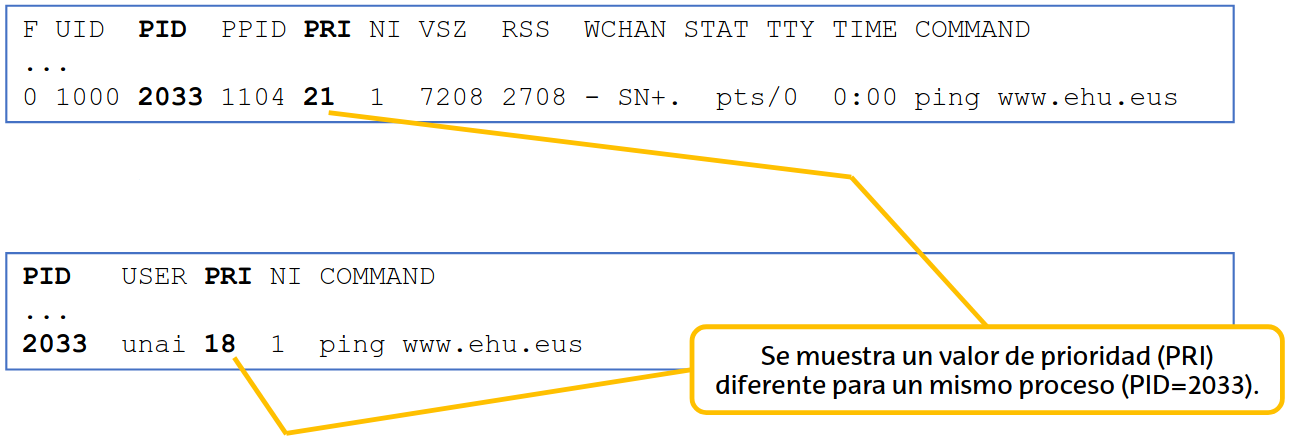
* Latencia:
  + El tiempo necesario para completar una petición.
  + Importante: Distinguir entre peticiones satisfactorias y fallidas.
  + **Cómo observar**: Monitorizar logs de aplicación.
* Tráfico:
  + Métrica de la demanda de un sistema.
  + Suelen ser específicas de un sistema.
  + Ejemplos:
    - Peticiones HTTP / segundo.
    - Transacciones / segundo.
  + **Cómo observar**: Monitorizar logs de aplicación y red.
* Errores:
  + Resultado de peticiones fallidas.
  + Pueden ser explícitos o implícitos
  + Ejemplo: Al solicitar una web a un servidor:
    - Error explícito: Error 404.
    - Error implícito: Recuperar una web diferente a la esperada.
  + **Cómo observar**: Monitorizar logs de aplicación y del sistema.
* Saturación:
  + Medida de la capacidad de un recurso en uso.
  + Importante observar los recursos más limitados.
  + La latencia puede ser un indicador de saturación.
  + **Cómo observar**: Monitorizar hardware (CPU, memoria, …) y red.

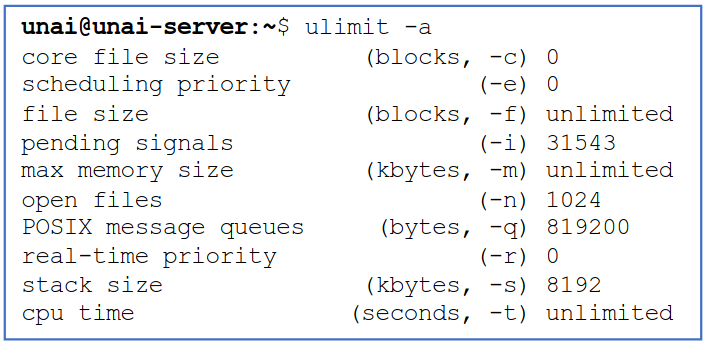
Monitorización de CPU

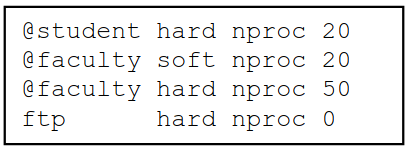
* Comando **top**:
  + Uso de recursos del sistema en tiempo real.
* Comando **ps**:
  + Listado de procesos y su uso de recursos.
* Comando **pstree**:
  + Árbol de procesos del sistema.

**1.3.2. Gestión de recursos:**

Gestión de procesos

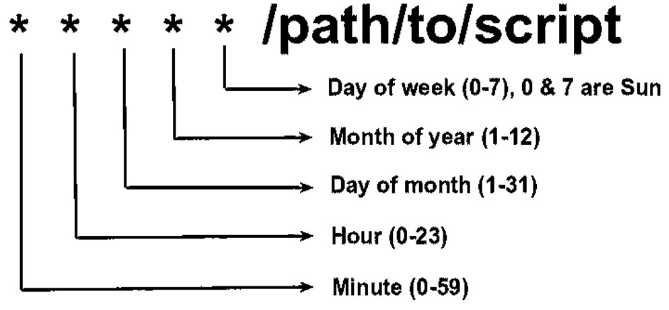
* Prioridades de los procesos:
  + El planificador del SSOO asigna intervalos de tiempo a los procesos según prioridad.
  + Se controla según 2 valores:
    - Prioridad (PR): Puede tomar valores en el rango -100 a 39.
    - Valor “nice” (NI): Puede tomar valores en el rango -20 a 19.
    - Columnas PR y NI en el comando Top.
    - Para ambos, cuanto más negativo el valor, mayor prioridad.
  + En Linux, los procesos se consideran de 2 tipos:
    - Procesos normales (La mayoría de los lanzados por usuarios).
    - Procesos de tiempo real (Generalmente, los esenciales para el SSOO).
* Prioridades de los procesos normales:
  + Se calcula: PR = 20 + NI
  + Ocupan el rango 0-39 de prioridades.
  + Por defecto, NI = 0.
  + Un usuario normal puede modificar el valor NI entre 0 y 19.
    - Puede reducir la prioridad sobre el resto de procesos del sistema.
  + El usuario root puede modificar el valor NI -20 y 19.
  + Comando **nice**:
    - Lanza un comando con un valor NI concreto
    - Sintaxis → nice -n [valor] [comando]
      * Valor es relativo (define cuánto más o menos)
    - Ejemplo → nice -n 10 ./miScript
* Prioridades de los procesos en tiempo real:
  + Se calcula: PR = – 1 - prioridad\_tiempo\_real
  + El valor prioridad\_tiempo\_real toma valores entre 1 y 99.
    - El valor NI da igual.
    - En el comando top, si PR = – 100, se muestra como ‘rt’ (real time).
  + Comando **chrt**:
    - Lanza un proceso con una prioridad de tiempo real.
    - Sintaxis → chrt --rr <prioridad\_tiempo\_real> <programa>
    - Ejemplo → chrt --rr 20 ./miPrograma
      * Lanzaría ./miPrograma con PR = – 21
* Comando **ps**:
  + Muestra datos en diferentes formatos.
    - Cada formato puede mostrar una misma prioridad con diferentes valores.
  + Formato BSD:
    - Comando → ps al
  + Formato Unix:
    - Comando → ps -u unai -o pid,user,pri,nice,args
* Comando **kill**:
  + Envía señales a procesos (no sólo matarlos).
  + Sintaxis → kill <opciones> PID
  + Opciones:
    - -l → Mostrar las señales disponibles.
    - -señal → Mandar una señal al proceso.
  + 3 formas de indicar una señal:
    - Con su número → -19
    - • Con el prefijo SIG → -SIGSTOP
    - • Sin el prefijo SIG → -STOP
  + Señales:
    - -STOP → Parar el proceso.
    - -CONT → Reanudar el proceso (parado con STOP).
    - -KILL → Matar el proceso.
* Comando **ulimit**:
  + Limitar el uso de recursos.
  + Los límites sirven para la Shell en uso.
  + Sintaxis → ulimit -<opción> [límite]
  + Opciones:
    - -a → Lista los límites establecidos.
    - -f → Máximo número de ficheros creados por la Shell.
    - -m → Máxima memoria disponible.
    - -t → Máximo tiempo de CPU (segundos).
  + Ejemplo:



* Fichero /etc/security/limits.conf:
  + Permite hacer una configuración permanente de límites
  + Cada línea tiene el siguiente formato → usuario/grupo tipo-de-límite ítem valor
    - usuario/grupo → Nombre del usuario o grupo (comienza con @).
    - tipo-de-limite → soft/hard.
    - item → Puede ser: cpu, nproc, maxlogins, fsize, …
    - valor → Valor para el ítem definido.
  + Ejemplos:
  + Mas información → man limits.conf
* Comando **cpulimit**:
  + Permite limitar el % de uso constante de CPU de un proceso.
    - ulimit y limits.conf sólo permiten limitar el tiempo total de uso CPU.
    - nice y renice permiten reducir la prioridad pero no fijar un umbral.
  + Está en los repositorios Debian
  + Uso → cpulimit --pid PID --limit <límite>
    - <límite> → Límite de % CPU máximo que queremos permitir.
  + Más información:
    - <https://www.tecmint.com/limit-cpu-usage-of-a-process-in-linux-with-cpulimit-tool/>

Planificación de tareas:

* Comando **crontab**:
  + Una línea por tarea programada.
  + Sintaxis → crontab <opciones>
  + Opciones:
    - -l → Mostrar las tareas programadas.
    - -e → Editar las tareas programadas.
    - -r → Elimina las tareas programadas.
  + Cada entrada de cron es una línea sigue la estructura.



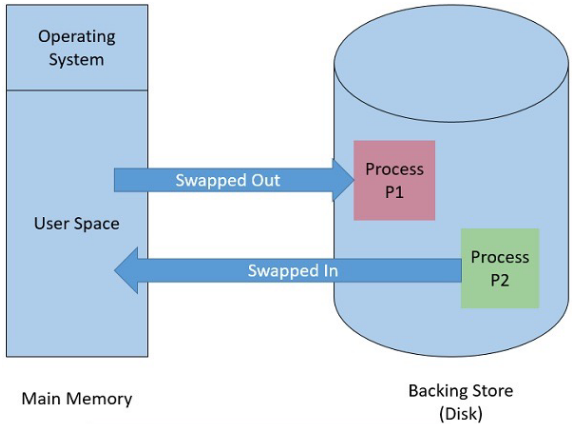
* + Editor online de entradas cron: <https://crontab.guru/>
* Comando **at**:
  + Controla las tareas a ejecutar por atd.
  + Para programar una tarea:
    - Desde Shell → at HORA
      * HORA → Una hora es formato HH:MM.
    - Se abre el Shell de at, escribir el/los comando(s) deseado(s):
      * P.e. ls /home/unai -l
    - Cerrar la Shell de at (Ctrl + D)
    - La salida estándar (stdout) se envía por mail usando sendmail.
      * Conviene revisar /var/spool/mail/<usuario>
  + Otras opciones (Desde Shell):
    - at -l → Listado de tareas pendientes.
    - at -d <ID> → Eliminar tarea (obtener ID con -l).

**1. ariketa:**

* sudo apt update
* sudo apt install stress-ng -y
* stress-ng --cpu 4 --timeout 2m
* top
* sudo renice 19 -p 2345 →2345 = PID
* sudo renice -20 -p 2345
* stress-ng --cpu 1 --timeout 3m
* sudo apt install cpulimit -y
* sudo cpulimit -p 2345 -l 50
* top

Gestión de memoria:

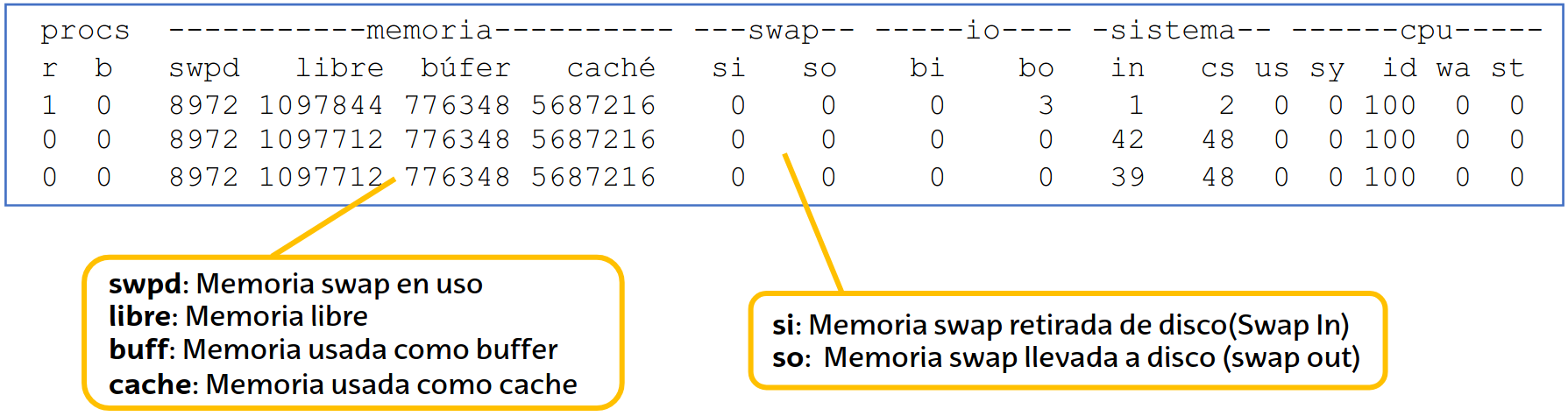
* La mayoría de sistemas operativos modernos utilizan memoria virtual.
  + Utilizan un espacio de disco como extensión de la memoria principal.
    - Espacio de intercambio o Swap.
  + Se organiza en páginas que se intercambian entre memoria y disco.



* + Un uso excesivo de Swap puede degradar el rendimiento.
  + Valores referencia de latencias en una jerarquía de memoria:

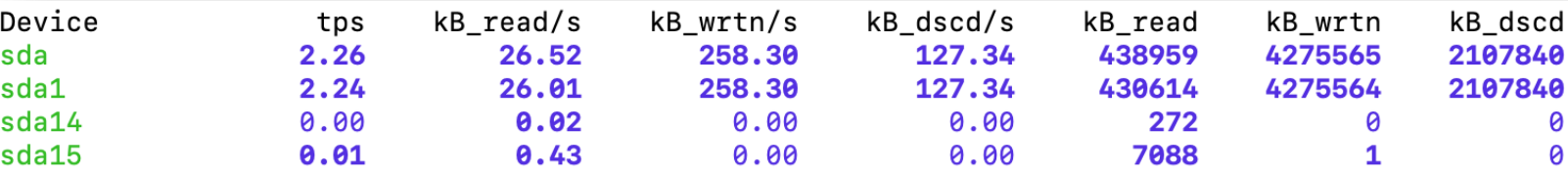
| **Tipo de memoria** | **Latencia** |
| --- | --- |
| Caché L1 en CPU | 1 ns |
| Caché L2 en CPU | 4 ns |
| RAM | 100 ns |
| SSD | 16.000 ns |
| HDD | 2.000.000 ns |

* Monitorizar la memoria:
  + Comando **top**:
    - Utilizar Shitf+m para ordenar por consumo descendente de memoria.
  + Comando **vmstat**:
    - Campos relativos a la memoria:



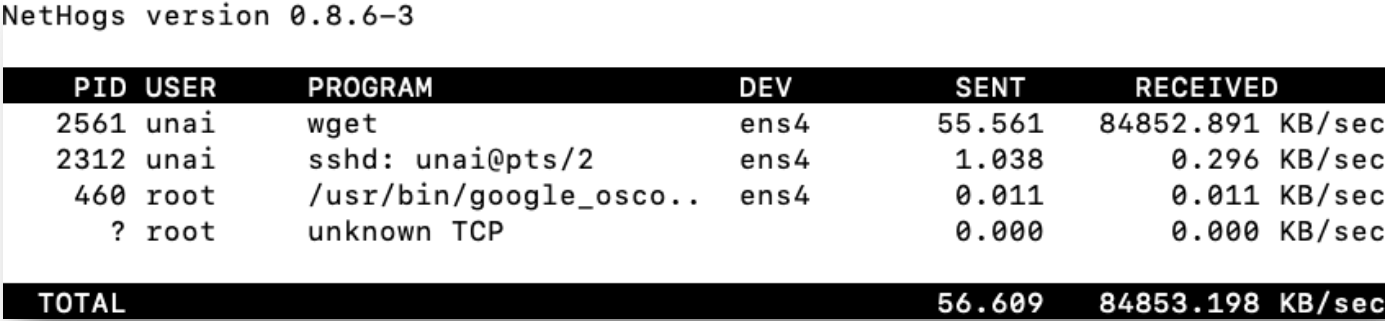
Gestión de discos y ficheros:

* Comando **df**:
  + Listado de sistemas de ficheros y espacio disponible.
  + Sintaxis → df <opciones>
  + Ejemplo → df -h
    - Muestra los tamaños en kB, MB, … en lugar de en bytes.
* Comando **du**:
  + Tamaño de una rama del sistema de ficheros (p.e., de un directorio).
  + Sintaxis → du <opciones> directorio
  + Ejemplo → du -sh /home
    - Muestra el tamaño total del directorio /home sin listar todo su contenido.
* Comando **lsof**:
  + Muestra los ficheros en uso por los procesos del sistema (list open files).
  + Útil para resolver el error “resource is busy” al desmontar una partición.
* Comando **iostat**:
  + Muestra estadísticas de uso y tasas de transferencia de los disp. de almacenamiento.
* Sintaxis → iostat –p <disco>
* Ejemplo → iostat –p /dev/sda



Gestión de red:

* Comando **netstat**:
  + Muestra información sobre las conexiones y rutas de red
  + Mostrar conexiones activas → netstat -a | more
  + Mostrar tabla de rutas → netstat -r
* Comando **nethogs**:
  + Muestra conexiones y ratio de tráfico enviado/recibido.
  + Requiere instalar el paquete sysstat.



* Comando **tcpdump**:
  + Es un analizador de tráfico para conexiones TCP/IP.
  + Uso más común: captura de tráfico para posterior análisis.
  + Comenzar a capturar tráfico y guardar en un fichero:
    - Sintaxis → tcpdump -i <interfaz> -Z <usuario> -w <ficheroCaptura>
    - Ejemplo → tcpdump -i ens4 -Z unai -w miCaptura
    - Las interfaces disponibles se pueden mostrar con → ip link
  + Visualizar un fichero de captura de tráfico:
    - Sintaxis → tcpdump -enr <ficheroCaptura>
    - Usar el parámetro -ttt para incluir la diferencia de tiempo entre cada paquete.
* Comando **telnet**:
  + Útil para comprobar si un servicio remoto está a la escucha.
  + Sintaxis → telnet <IP> <puerto>
* Comando **netcat**:
  + Herramienta para leer y escribir en conexiones de red.
  + Abrir una conexión a la escucha en un puerto → nc -l <puerto>
  + Conectarse a una IP/puerto y escribir en él → Sintaxis: nc <IP> <puerto>

**2. ariketa:**

**1.3.3. Análisis de rendimiento:**

Código Python de ejemplo:

* Implementación de la sucesión de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, …).
* Calcula el valor del número 34 en la sucesión.



Comando **time**:

* Mide cuánto tarda en ejecutarse un comando/script.
* Sintaxis → time <comando>
* Salida: real 0m2.345s

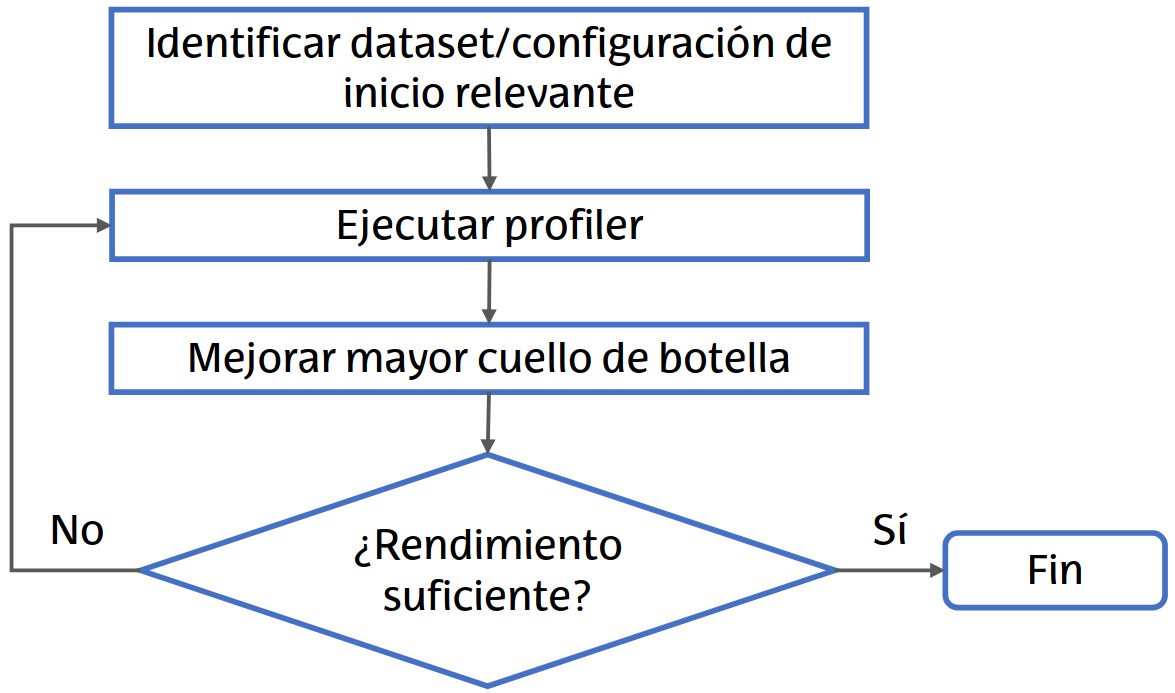
user 0m1.876s

sys 0m0.469s

* Interpretación:
  + real: Tiempo total de ejecución (como si fuese medido con un cronómetro).
  + user: Tiempo dedicado a ejecutar código de la aplicación (p.e. cálculos).
  + sys: Tiempo dedicado a llamadas del sistema (p.e. acceder a ficheros).

Profiler:

* Herramienta de análisis que mide el uso de recursos durante la ejecución de un software.
* Mide el tiempo que pasa un software en determinadas funciones.
* Ayuda a identificar cuellos de botella:
  + Porciones de código en las que el software pasa la mayor parte del tiempo.
* Muestra el flujo de llamadas entre funciones.
* Complementario a un debugger.
* Los profilers son específicos para 1 o varios lenguajes.
  + C / C++: gprof, perf
  + Python: cProfile, py-spy, Scalene.
    - cProfile → python3 -m cProfile fichero.py
  + Matlab: Profiler integrado.
  + Javascript: Chrome DevTools.
* 2 tipos principales:
  + Basados en trazas (Tracing profilers):
    - Registran cada llamada y retorno a funciones.
    - Suelen provocar overhead (incremento del tiempo de ejecución).
      * P.e. cProfile en Python.
  + De muestreo (Sampling profilers):
    - Inspeccionan el estado del programa en intervalo (p.e. cada 10 ms).
    - Menos precisos, poco impacto en el tiempo de ejecución.
      * P.e. Scalene en Python.
* Procedimiento habitual de mejora de rendimiento:



* El rendimiento de un programa está relacionado con su complejidad computacional:
  + La notación O grande define el rendimiento teórico.
  + Los profilers miden el rendimiento real y uso de recursos.
* Ejemplo:
  + Complejidad temporal de algoritmos de ordenación:

| **Algoritmo** | **Complejidad** | **Rendimiento** |
| --- | --- | --- |
| Bubble Sort | O(n²) | Muy lento |
| Quick Sort | O(n·log n) | Rápido en la mayoría de casos |
| Radix Sort | O(n·d) | Depende del tipo de datos |

**3. ariketa:**

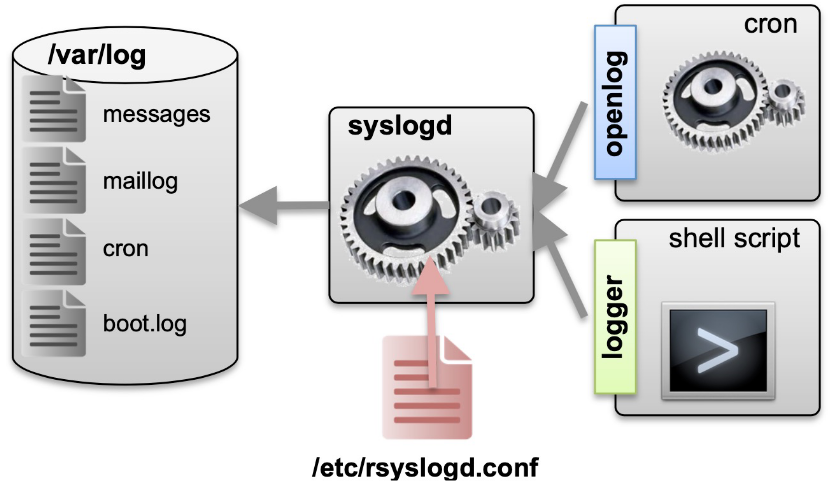
**1.3.4. Registros del sistema (Logs):**

Logs:

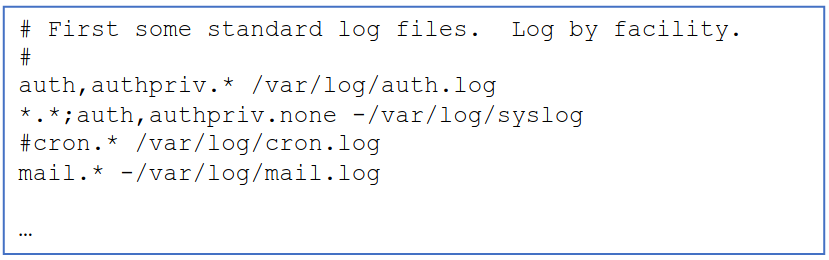
* El kernel de Linux, los servicios y las aplicaciones generan eventos constantemente:
  + Información sobre su estado, fallos/anomalías, …
  + Errores de arranque.
  + Acceso a información (seguridad).
* Una gestión adecuada de esta información es esencial para descubrir y solucionar problemas.
* Todos estos eventos suelen estar gestionados por un único servicio.
  + En Unix/Linux es syslog.

Syslog:

* Es el recolector de eventos empleado por el kernel, servicios y aplicaciones.
* Flexible, seguro y fácil de usar.
* Está compuesto por los siguientes elementos:



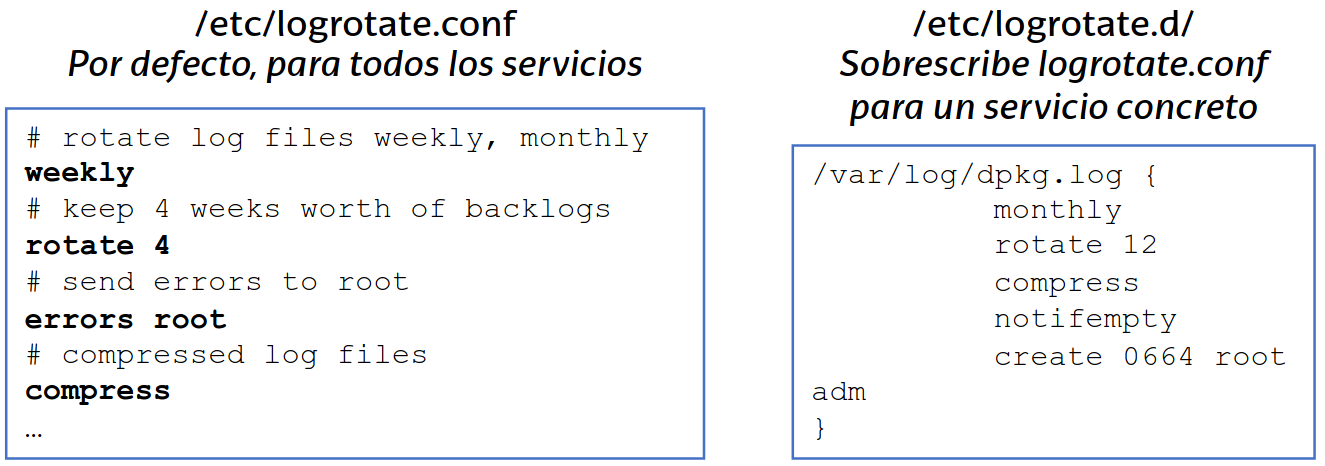
* Partes de syslog:
  + syslogd:
    - Servicio del sistema.
    - Recibe los mensajes del resto de servicios y apps y los añade al registro.
  + openlog:
    - Librerías para usar syslog desde una aplicación.
    - P.e., openlog (C/C++), sys::syslog(openlog(),syslog()) (Perl)
  + logger:
    - Comando del sistema para enviar mensajes a syslog.
  + rsyslogd.conf:
    - Fichero de configuración.
* rsyslogd.conf:
  + Listado de acciones a realizar en función de los mensajes recibidos.
  + Tiene una línea por acción, con el formato: entidad.nivel acción
  + Entidad: Lista de valores definidos por el sistema.
    - P.e.: Kern, user, daemon (otro servicio), auth (login, su, ssh), mail, cron, …
  + Nivel: Tipo de notificación.
    - emerg, alert, crit, err, warning, notice, info, debug, \* (todos los niveles).
  + Acción:
    - <nombre-de-fichero> → Escribir el mensaje a ese fichero.
    - <nombre-dominio>/<IP> → Enviar el mensaje al syslogd del nodo indicado.
    - <nombre-usuario> → Enviar mensaje al usuario, si está conectado.
    - \* → Enviar mensaje a todo usuario conectado.
  + Ejemplo:



* + En Ubuntu, por defecto es: /etc/rsyslog.d/50-default.conf.
  + Los cambios en este fichero requieren reiniciar el servicio rsyslog.
* Syslog escribe en ficheros en /var/log:
  + syslog → Eventos generales, ni críticos ni de depuración.
  + kern.log → Registros del kernel.
  + auth.log → Información de inicio/fin de sesiones.
* Gestión de logs:
  + Los logs son una herramienta fundamental para el control y reparación del sistema…
  + … pero cuanta más información de logs, mayor uso de disco.
    - Los logs pueden llegar a consumir un espacio significativo.
    - Puede ser costoso buscar información/datos concretos entre miles de líneas.
  + Rotación de logs:
    - Periódicamente cambiar el fichero donde se escriben los logs, cambiando a escribir en uno nuevo y borrando el más antiguo.
    - Se puede hacer de manera manual con un script:



* + - Se puede implementar la rotación con un servicio del sistema.
      * Evita errores humanos al crear scripts.
    - Servicio **logrotate**.
    - Se configura con los siguientes ficheros:



* + Analizando logs:
    - Para depuración:
      * Útil para obtener más información cuando algo va mal.
      * Activar modo verboso de las aplicación:
        + P.e. activar flag -d, /etc/init.d/ssh sshd -d
      * Importante: Desactivar el modo verboso al volver a producción.
    - Para monitorización:
      * Problema: Abundante información, de la que mucha puede ser inútil.
      * Utilizar herramientas para buscar los mensajes relevantes, p.e.:
        + Swatch: Programa Perl que busca patrones en los logs.
        + LogWatch: Genera resúmenes para su envío por e-mail.
        + Soluciones más complejas, p.e., pila ELK.

**3. ariketa:**