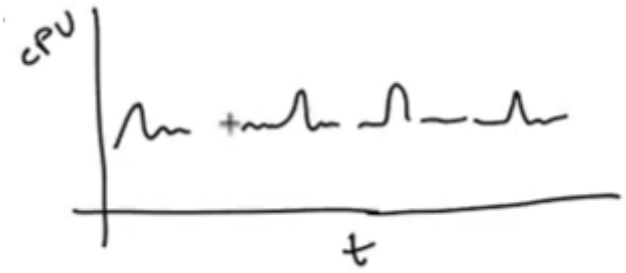
LECCION 1

En la P3 -> trabajo autónomo. Monitorización, luego le pasaremos un benchmark y veremos como se conforma. También Profiling pero no está dentro de la asignatura así que se ve muy por encima. No lo van a pedir como objetivos mínimos.

Vamos a conocer y a aprender a utilizar monitores. Si hay un panel de aplicaciones y vemos esto:



Y podemos interpretar que hay picos con periodicidad y mismas alturas. Entonces, a la hora de hacer la revisión, no sería en el punto álgido, sino cuando haya menos carga en la CPU. También se ve que hay momentos en los que no hay uso siquiera, eso se debe a desconexión de red. Si se cae la red de la máquina que lo monitoriza, deja de haber lecturas. Si el monitor está off, aunque la red esté enviando información, no va a recoger información. Son las dos teorías posibles. De una gráfica como esa se sacarían los valores medios con los que luego juegan para tener productividad máxima, respuesta mínima, etc.

No utilizamos monitores hardware porque usamos una máquina virtual, es todo software. Comandos útiles son listar (ls), lspci y lsusb (para saber si tenemos el dispositivo conectado o no). Es interesante porque podemos ver por qué se llaman nuestras interfaces np0s3 o cosas asi, y es porque lo ubica dentro de los buses que hay y cada bus puede tener varios slots.

Otro comando interesante para monitorizar los mensajes que nos envía el kernel es dmesg. Este comando pinta las líneas del kernel. Antiguamente se iba a un archivo que no está ya ni en Fedora. En cetos7 si.

En terminal suya local:

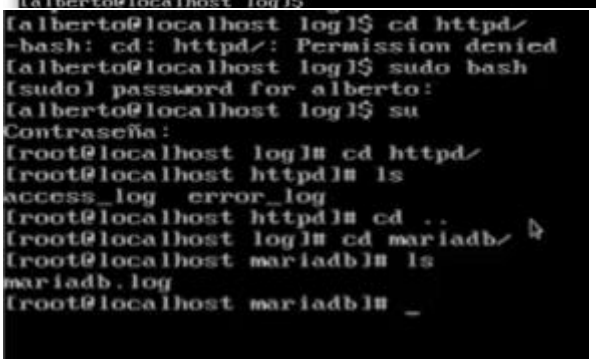
- Dmesg | tail -> Se puede añadir al final el número de líneas que queremos. Por defecto vienen 5. Le da un error con el usb. Así se puede ver el fabricante y todo eso si metemos una memoria por usb. También se puede usar “write through” para escribir directamente sin pasar por la offer cache(¿) y alerta si el dispositivo tiene algún problema (el sistema de archivos incorrecto y tal (fsck))

El kernel lidia con hw y sw, entonces es normal que tengamos monitorización híbrida. Monitores específicos para software: en Unix tenemos un directorio /proc que tiene tantos directorios como procesos. Además hay otros archivos en el stat y eso que nos da información sobre el sistema. Cuando queramos depurar podemos navegar por dentro de la estructura del proceso y explorar la lista de llamadas, archivos abiertos etc. El nivel de depuración y de monitorización que tenemos en Linux es excelente. Importante conocer la carpeta var para

encontrar errores y tal, relacionado con SELinux y cosas de ssh. Podemos definir nosotros las reglas. Si hay problemas en la instalación de un servicio entero mejor miramos a los archivos de log. Var/log tiene importancia de cara a monitorizar el estado de nuestra aplicación, servicios, etc. /proc/mdstat lo que nos da es el estado de los multidevice.

Terminal de CentOS:

- Cd /var/log
- Ls -> tenemos distintos logs, el estándar u otro creado para algún proceso concreto. Dentro tenemos los errores y los accesos.



-

Se mete en “raid.wiki.kernel.org/index.php/Mdstat” que tiene información sobre mdstat. Se aprende cómo se lee el archivo. LEER MEJOR.

APUNTES DE LO DE MDSTAT----------

APUNTES DE LO DE MDSTAT----------

Con strace lo que hacemos es una traza de las llamadas al sistema. Deja el enlace también “chadfowler.com/2014/01/26/the-magic-of-strace.html” para que le echemos un ojo a un ejemplo de una persona que monitoriza un proceso de httpd.

MONITORES GENERALES

Tenemos muchos monitores que monitorizan parte sw y hw y monitorizan aplicaciones y servicios concretos y vistazos al sistema en general. Munin significa memoria. Ocupa pocos recursos. Los problemas de la memorización es el overhead que generamos en el sistema.

Nagios es de los monitores mas famosos y potentes de la ultima década. Paso a ser software privativo hace unos 3 años y por ello algunos desarrolladores se fueron o los echaron. Los que se separaron crearon la versión libre naemon. La documentación a veces referencia a nagios que es raro.

ZABBIX es un monitor de sistemas que ofrece una funcionalidad muy completa, la documentación para todo el mundo es bastante buena, su API está bien hecha (gracias a ella automatizamos muchas tareas) y los distintos protocolos que nos permite (escuchas activas, pasivas, personalización de algunos pluggins...). Hay que instalar Zabbix para Ubuntu server para que Ubuntu se monitorice a si mismo y a centos comprobando los servicios ssh y http. También nos piden que hagamos una pequeña memoria con capturas de qué hacemos, de donde sacamos la info y todo eso. Podemos mandárselo antes para que le eche un vistazo.

AUTOMATIZACION

Los units de systemd. Cron ya no existe. En debian si lo siguen usando pero ahora todos usan systemd porque permite ir arrancando servicios en paralelo, con lo cual mejoramos el tiempo de arranque e incorpora un módulo de software que nos permite controlar la ejecución. Eso no quita que podamos editar cron en centos o redhat, pero es una capa de software. Tenemos que ir a nuestro archivo en /etc/systemd/system y definimos un unit. Así podemos monitorizar nuestro raid y vamos a tener un pequeño script de Python en /usr/bin/python3 /home/usuario/mon-raid.py que nos monitoriza el estado del raid. Hay que leer lo de los mdstat. Ese tipo de units (que son muy personalizables) se puede incrustar en un sistema de monitorización más general como zabbix, naemon, etc.

```
import re
f=open('/proc/mdstat')
for line in f:
  b=re.findall('\[[U]*[_]+[U]*\]',line)
  if(b!=[]):
     print("---ERROR en RAID---")
print("---OK Script---")
```

Importa una biblioteca que analiza expresiones regulares. Abrimos un archivo que se asigna a la variable f y abrimos la que nos da información sobre el estado de los multidevices (raid 1 lo es por ejemplo). Luego hace un for para cada línea y va buscando que haya uno o mas discos caidos y si lo hay pone lo de ERROR pero si tenemos uno o mas Ups en corchete dentro del string line, no hacemos nada. Al final cerramos y fin. El script se guarda en un archivo y en el unit que hay que crear para que lo ejecute le ponemos la ruta

```
ExecStart=/usr/bin/python3 /home/alberto/mon-raid.py
```

Para consultar la salida de ese comando es con journalctl

```
podemos monitorizar su ejecución gracias al comando jour
p.ej. journalctl -u mon-raid --since="yesterday".
```

Es un ejemplo para ver lo del dia anterior. Mdadm podemos configurar hasta nuestro correo.

Se puede probar a borrar un disco de los dos en el Ubuntu de virtual box

En Ubuntu:

- /proc/mdstat -> después de borrar uno de los discos

```
Personalities : [raid1] [linear] [multipath] [ra
md1 : active raid1 sda2[0]
      203776 blocks super 1.2 [2/1] [U_]

md0 : active raid1 sda3[0]
      10267648 blocks super 1.2 [2/1] [U_]
```

- A Jorge si que le da fallo:

```
[    1.612056] usbcore: registered new interface driver usbhid
[    1.612556] usbhid: USB HID core driver
[    1.617084] e1000 0000:00:03.0 enp0s3: renamed from eth0
[    1.619222] input: VirtualBox USB Tablet as /devices/pci0000:00/0000:00:06.0/usb1/1-1/1-1:1.0/000
3:80EE:0021.0001/input/input6
[    1.620679] hid-generic 0003:80EE:0021.0001: input,hidraw0: USB HID v1.10 Mouse [VirtualBox USB T
ablet] on usb-0000:00:06.0-1/input0
[    1.771659] ata4: SATA link down (SStatus 0 SControl 300)
Begin: Loading essential drivers ... [    1.926724] raid6: avx2x4   gen() 26829 MB/s
[    1.974947] raid6: avx2x4   xor() 16731 MB/s
[    2.022270] raid6: avx2x2   gen() 23025 MB/s
[    2.070672] raid6: avx2x2   xor() 14897 MB/s
[    2.118271] raid6: avx2x1   gen() 21206 MB/s
[    2.166346] raid6: avx2x1   xor() 14512 MB/s
[    2.214282] raid6: sse2x4   gen() 14970 MB/s
[    2.262480] raid6: sse2x4   xor()  9329 MB/s
[    2.310867] raid6: sse2x2   gen() 12365 MB/s
[    2.358264] raid6: sse2x2   xor()  8248 MB/s
[    2.406566] raid6: sse2x1   gen() 10862 MB/s
[    2.454262] raid6: sse2x1   xor()  7271 MB/s
[    2.454679] raid6: using algorithm avx2x4 gen() 26829 MB/s
[    2.455070] raid6: .... xor() 16731 MB/s, rmw enabled
[    2.455456] raid6: using avx2x2 recovery algorithm
[    2.457573] xor: automatically using best checksumming function   avx
[    2.458910] async_tx: api initialized (async)
done.
Begin: Running /scripts/init-premount ... done.
Begin: Mounting root file system ... Begin: Running /scripts/local-top ...   Volume group "vg0" not
found
  Cannot process volume group vg0
cryptsetup: Waiting for encrypted source device
    UUID=835d975f-0b04-48c9-a8e6-aa5ef75875e7...

practicas,ise
```

- Con esa información ya debemos ser capaces de saber si está funcionando el raid1 o no. Ha funcionado porque si ha llegado ahí , el raid1 que configuramos para /boot ha funcionado. Entonces, si no hubiéramos /boot dentro del raid y hubiésemos borrado el disco que tiene /boot, no habría ni arrancado. Después de unos 300 segundos nos manda a initramfs que es una consola que esta almacenado en ram durante el inicio. Es un filesystem que esta en ram durante el inicio. Pulsando tabulador dos veces se ven unos comandos relativos. Faltan muchos pero tenemos lvm y al probarlo y darle tabulador tabulador están los que conocemos. Podriamos monitorizar qué está ocurriendo (depurar). Para ver lo que nos dice el kernel usamos dmesg (ya esta en initramfs de nuevo con ctrl d). Ha paginado los errores y con Mayuscula+RePag nos movemos y lo vemos. El archivo que tiene información sobre el raid (sobre el multidevice) es mdstat (asi que cat /proc/mdstat)

```
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : inactive sda2[1](S)
      306176 blocks super 1.2

md1 : inactive sda3[1](S)
      10166272 blocks super 1.2

unused devices: <none>
(initramfs)
```

- Y vemos que tenemos nuestros dos multidevices e inactivos. Pulsamos tabulador dos veces y en la columna tercera reconocemos mdadm. Para ver como se activa un raid hacemos man mdadm , pero fuera, ahí no vale. Es mdadm -R con /dev/md0 y con /dev/md1 y ahora lo probamos con cat /proc/mdstat.

```
(initramfs) mdadm -R md0 md1
[  710.843485] md/raid1:md0: active with 1 out of 2 mirrors
[  710.843762] md0: detected capacity change from 0 to 313524224
mdadm: started array /dev/md0
[  710.847656] md/raid1:md1: active with 1 out of 2 mirrors
[  710.847991] md1: detected capacity change from 0 to 10410262528
mdadm: started array /dev/md1
(initramfs) cat proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active (auto-read-only) raid1 sda2[1]
      306176 blocks super 1.2 [2/1] [_U]

md1 : active (auto-read-only) raid1 sda3[1]
      10166272 blocks super 1.2 [2/1] [_U]

unused devices: <none>
(initramfs) _
```

- Y ahora vemos que tenemos la entrada para md0 y md1 activos, pero uno de ellos caido. Pulsamos ctrl+d y vemos como continua el arranque sin problemas. Es un bug que no sabe que hacer cuando el raid1 esta caido. Si vuelven a arrancar de nuevo, se queda por defecto que arranca bien. Buscamos con man mdadm como reconstruir el raid. Podemos monitorizar el proceso de reconstrucción del raid con el comando watch cada dos segundos que me ejecute proc/mdstat .Ahora tenemos que insertarle un disco nuevo y reconstruir el raid con el comando mdadm.

ALBERTO GUILLEN PERALES 15:36
warch -n 2 cat /proc/mdstat

watch -n 2 cat /proc/mdstat

ALBERTO GUILLEN PERALES 15:37
mdadm --add /dev/md9 /dev/sdb

mdadm --add /dev/md0 /dev/sdd1

mdadm --add /dev/md2 /dev/sdd2

- noseques