Practica – Sistemas Operativos

PRÁCTICA: EXPLORACIÓN DEL SISTEMA OPERATIVO

Identificación de Kernel, Procesos y Componentes del SO

PARTE 1: IDENTIFICACIÓN DEL KERNEL

Objetivo: Distinguir qué información proviene del kernel vs del SO.

Comandos básicos:

uname -a
Muestra: Kernel Linux, versión 5.15.90.1-microsoft-standard-WSL2, arquitectura x86_64
Esto es información del KERNEL

uname -r
Muestra solo la versión del kernel
Esto es información del KERNEL

cat /proc/version
Muestra versión del kernel más detalles de compilación
Esto es información del KERNEL

hostnamectl
Muestra kernel + distribución del SO
Esto es información MIXTA

lsb_release -a
Muestra: Distributor ID: Ubuntu, Description: Ubuntu 22.04.3 LTS
Esto es información del SO (no del kernel)

cat /etc/os-release
Muestra nombre, versión e ID del SO
Esto es información del SO (no del kernel)

Ejercicio de análisis:

Ejecuta estos comandos y completa:

uname -a
Resultado ejemplo: Linux laptop 5.15.90.1-microsoft-standard-WSL2 #1 SMP ...

lsb_release -a
Resultado ejemplo: Distributor ID: Ubuntu, Description: Ubuntu 22.04.3 LTS

Preguntas para responder:

Versión del kernel: 5.15.90.1-microsoft-standard-WSL2
Esto es: KERNEL

Distribución: Ubuntu
Esto es: SO

Versión del SO: 22.04.3 LTS
Esto es: SO

Arquitectura: x86_64
Esto es: KERNEL/HARDWARE

Pregunta de reflexión:
Si cambias de Ubuntu 22.04 a Fedora 38, pero mantienes el kernel 5.15, ¿qué cambia?

Respuesta: Solo cambia el SO (herramientas, bibliotecas, gestor de paquetes). El kernel puede ser el mismo si es compatible.

PARTE 2: EXPLORACIÓN DE PROCESOS

Objetivo: Identificar quién crea procesos, quién los gestiona, y cómo se relacionan.

A. Identificar el init y procesos del sistema

ps -p 1
Muestra el proceso init (PID 1), iniciado directamente por el kernel
PPID es 0 porque el kernel lo crea

ps -p 1 -o pid,ppid,cmd
Muestra PID, PPID y comando del init
Verás que PPID es 0 (kernel)

pstree
Muestra árbol de procesos: init → servicios → aplicaciones
Permite ver la jerarquía completa

systemctl status
Muestra estado del init moderno (systemd en la mayoría de distribuciones)

Ejemplo de salida esperado:

ps -p 1
PID TTY TIME CMD
1 ? 00:00:01 init

O con systemd:
ps -p 1
PID TTY TIME CMD
1 ? 00:00:02 systemd

B. Identificar procesos kernel vs procesos de usuario

Procesos del kernel (threads internos):

ps aux | grep "^\[" | head -5

Salida típica:
[rcu_gp]
[rcu_par_gp]
[kworker/0:0]
[kworker/0:0H]
[mm_percpu_wq]

Estos procesos con corchetes [] son threads del kernel, no aplicaciones de usuario.

Procesos de usuario (normales):
ps aux | grep -v "^\[" | head -5

Salida típica:
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.0 2384 1424 ? Ss 10:00 0:01 /init
root 7 0.0 0.0 2400 1500 ? Ss 10:00 0:00 /init
usuario 123 0.1 0.5 12500 8900 pts/0 Ss 10:05 0:02 bash
usuario 456 0.0 0.3 8900 4500 pts/0 R+ 10:30 0:00 ps aux

C. Jerarquía de procesos: quién inicia a quién

pstree -p

Ejemplo de salida:
```
init(1)─┬─init(7)───bash(123)───pstree(789)
        ├─systemd-journal(45)
        ├─sshd(67)───sshd(234)───bash(235)
        ├─dockerd(89)───containerd(90)───...
        └─...
```

Identificación en el árbol:
- init(1) iniciado por kernel (PPID=0)

- bash(123) iniciado por init(7)
- pstree(789) iniciado por bash(123)

D. Estados de procesos

Estados posibles:

R = Running (ejecutando o esperando CPU)
S = Sleeping (dormido, esperando evento)
D = Uninterruptible sleep (dormido profundo, usualmente I/O)
Z = Zombie (terminado, padre no ha leído su status)
T = Stopped (detenido por señal SIGSTOP)

Modificadores (caracteres adicionales)

s = Session leader (líder de sesión, usualmente shell)
l = Multi-threaded (tiene múltiples threads)
+ = Foreground process (en primer plano, terminal actual)
< = High priority (prioridad alta, nice negativo)
N = Low priority (prioridad baja, nice positivo)
L= Has pages locked in memory (bloqueado en RAM)

Ver distribución de estados:

ps aux | awk '{print $8}' | sort | uniq -c | sort -rn

Salida típica:

```
 13 Ss
  9 S
  5 Ssl
  4 S+
  2 Ss+
  2 Sl
  2 R+
  1 STAT
  1 S<s
```

Explicacion:

13 Ss

S = Sleeping (dormido)
s = Session leader
Qué son: Procesos que son líderes de sesión, probablemente shells (bash, zsh) o demonios que iniciaron sesión. Son 13 procesos que actúan como "padres" de otros procesos en sus sesiones.

9 S

S = Sleeping puro, sin modificadores

Qué son: Procesos normales dormidos, sin características especiales. Esperan eventos (teclado, red, timer).

5 Ssl

S = Sleeping
s = Session leader
l = Multi-threaded
Qué son: Líderes de sesión con múltiples threads. Probablemente servicios del sistema o aplicaciones complejas (bases de datos, navegadores).

4 S+

S = Sleeping
+ = Foreground (primer plano)
Qué son: Procesos dormidos pero en la terminal actual que estás usando. Pueden ser editores, comandos pausados, etc.

2 Ss+

S = Sleeping
s = Session leader
+ = Foreground
Qué son: Líderes de sesión en primer plano. Probablemente tu shell actual (bash) y algún proceso hijo importante.

2 Sl

S = Sleeping
l = Multi-threaded
Qué son: Procesos multi-thread dormidos, no son líderes de sesión. Aplicaciones con varios threads de trabajo.

2 R+

R = Running
+ = Foreground
Qué son: Procesos ejecutándose activamente en tu terminal. Probablemente uno es el awk del propio comando, otro puede ser algo en bucle o interactivo.

1 STAT

Esto es el header de ps aux
La primera línea de ps aux tiene encabezados: USER, PID, %CPU, %MEM, VSZ, RSS, TTY, STAT, START, TIME, COMMAND. El awk tomó la palabra "STAT" como si fuera un estado. Es un artefacto del comando, no un proceso real.

1 S<s

S = Sleeping
< = High priority (nice < 0)
s = Session leader
Qué es: Un proceso líder de sesión con prioridad alta. Probablemente un servicio crítico del sistema que necesita CPU preferente.-

B. Gestión de memoria (Kernel + SO colaborando)

Nivel kernel (vista cruda):

cat /proc/meminfo | head -5

Salida típica:

```
MemTotal:       16384000 kB
MemFree:         2345678 kB
MemAvailable:    8901234 kB
Buffers:          123456 kB
Cached:          5678901 kB
```

Buffers (en este contexto) son memoria RAM usada para almacenar datos de bloques de disco antes de escribirlos o después de leerlos, cuando aún no están organizados como archivos completos.

Diferencia clave: Buffers vs Cached

Concepto	**Buffers**	**Cached**
Qué almacena	Datos de bloques de disco "crudos"	Contenido de archivos completos
Nivel	Bloques del filesystem (inodos, metadatos)	Archivos enteros (lectura de documentos, binarios)
Cuándo se usa	Operaciones de filesystem, escrituras pendientes	Lectura repetida de archivos
Se libera fácil	Sí, cuando se necesita RAM	Sí, con echo 3 > /proc/sys/vm/drop_caches

Esto es información expuesta por el kernel.

Nivel SO (presentación legible):

free -h

Salida típica:

```
               total        used        free      shared  buff/cache   available
Mem:            15Gi       6.5Gi       2.2Gi       512Mi       6.8Gi       8.5Gi
Swap:          2.0Gi       0.0Ki       2.0Gi
```

Esta presentación amigable es trabajo del SO.

Mapa de memoria de un proceso:

pmap $$

Muestra cómo el kernel ha mapeado la memoria virtual del shell actual.

C. Gestión de procesos (Kernel ejecuta, SO interfacea)

Acción: Crear proceso
Comando del SO: ./programa o bash
Syscall del kernel: fork() + execve()

Acción: Listar procesos
Comando del SO: ps aux
Origen de datos: /proc (expuesto por kernel)

Acción: Cambiar prioridad
Comando del SO: nice -n 10 ./prog
Syscall del kernel: setpriority()

Acción: Terminar proceso
Comando del SO: kill 1234
Syscall del kernel: kill() con señal

Acción: Ver recursos
Comando del SO: top, htop
Origen de datos: /proc/[pid]/stat, /proc/[pid]/status

Demostración de kill:

sleep 1000 &
[1] 5678

kill -STOP 5678
[1]+ Stopped sleep 1000

kill -CONT 5678
[1]+ Running sleep 1000 &

kill 5678
[1]+ Terminated sleep 1000

El SO (comando kill) envía la señal. El kernel recibe la señal y modifica el estado del proceso.

D. Identificación de llamadas al sistema (syscalls)

strace -c ls /

Salida típica:

% time	seconds	usecs/call	calls	errors	syscall
0.00	0.000000	0	1		read
0.00	0.000000	0	2		openat

```
  0.00    0.000000           0         2           close
  0.00    0.000000           0         1           fstat
  0.00    0.000000           0        11           getdents64
  0.00    0.000000           0         1           write
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                     9         0 total
```

Interpretación:
- ls es un comando del SO
- Pero usa syscalls del kernel para: abrir archivos, leer directorios, escribir output
- getdents64 = leer entradas de directorio
- write = escribir a terminal

PARTE 3: COMPARATIVA WINDOWS

Versión kernel en PowerShell:

[System.Environment]::OSVersion.Version

Versión SO en PowerShell:

Get-ComputerInfo | Select OsName, OsVersion

Procesos en PowerShell:

Get-Process

o

tasklist

Árbol de procesos en PowerShell:

Get-Process | Select-Object Name, Parent

Memoria en PowerShell:

Get-CimInstance Win32_OperatingSystem | Select TotalVisibleMemorySize, FreePhysicalMemory

Archivos abiertos en PowerShell (limitado):
Get-Process | Get-ChildItem

Syscalls en Windows:
No hay strace nativo. Requiere Process Monitor de Sysinternals.

Ejemplo completo Windows:

[System.Environment]::OSVersion

Salida:
Platform   : Win32NT
Version   : 10.0.22631.0
ServicePack   :
VersionString : Microsoft Windows NT 10.0.22631.0

Get-ComputerInfo | Select OsName, WindowsVersion, TotalPhysicalMemory

Salida:
OsName: Microsoft Windows 11 Pro
WindowsVersion: 2009
TotalPhysicalMemory: 17179869184

Matar ambiente grafico Windows

1. Abre Administrador de Tareas (Ctrl+Shift+Esc)

2. Busca "Explorador de Windows" o "Windows Explorer"

3. Clic derecho → "Finalizar tarea"

4. ¿Qué pasa?
- Pantalla negra (o fondo de pantalla visible)
- Sin barra de tareas
- Sin menú inicio
- Sin íconos de escritorio
- PERO: El sistema sigue funcionando

5. Para recuperar:
- Administrador de Tareas → Archivo → Ejecutar nueva tarea
- Escribir: explorer.exe
- Todo vuelve a la normalidad

PARTE 5: EJERCICIO DE SÍNTESIS

Escenario: Identificar quién hace qué en cada situación.

Situación 1: Escribes ls y presionas Enter
Quién actúa: SO (shell interpreta), luego kernel (ejecuta syscalls)
Justificación: Shell es parte del SO. El acceso a disco lo hace el kernel.

Situación 2: El mouse se congela 2 segundos
Quién actúa: Kernel (driver de USB/PS2 procesando interrupción)
Justificación: Los drivers de hardware corren en kernel space.

Situación 3: Aparece ventana "Disco lleno"
Quién actúa: SO (daemon de notificación)
Justificación: Es una aplicación de usuario, no el kernel directamente.

Situación 4: La batería baja a 10%
Quién actúa: Kernel (driver ACPI lee sensor), luego SO (notifica)
Justificación: Hardware → Kernel → SO → Usuario

Situación 5: Abres una foto
Quién actúa: SO (file manager), kernel (read syscall), SO (viewer app)
Justificación: Múltiples capas colaborando.

DESAFÍO FINAL: Script de síntesis

Crea un script que identifique:
1. Versión de kernel
2. Nombre del SO
3. Número total de procesos
4. Cuántos son kernel threads ([])
5. Cuántos son procesos de usuario
6. Porcentaje de memoria usada

Script modelo:

```
#!/bin/bash

echo "=== IDENTIFICACIÓN DEL SISTEMA ==="
echo "Kernel: $(uname -r)"
echo "SO: $(lsb_release -ds 2>/dev/null || cat /etc/os-release | grep PRETTY_NAME | cut -d"" -f2)"
echo ""

echo "=== PROCESOS ==="
total=$(ps aux | wc -l)
kernel_threads=$(ps aux | grep "^\[" | wc -l)
user_processes=$((total - kernel_threads - 1))

echo "Total líneas (aprox): $total"
echo "Kernel threads ([]): $kernel_threads"
echo "Procesos de usuario: $user_processes"
echo ""

echo "=== MEMORIA ==="
free -h | grep Mem
echo ""

echo "=== CONCLUSIÓN ==="
echo "El kernel $(uname -r) gestiona $kernel_threads threads internos"
echo "El SO $(lsb_release -is 2>/dev/null) proporciona $user_processes herramientas y servicios"
```