**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

**Evaluación: ps, grep, pipes linux, bash, awk**

**Objetivos**

• Aprender a listar y filtrar procesos activos en un sistema.

• Entender cómo identificar procesos por PID, usuario, uso de recursos y otros criterios.

• Utilizar **ps** para monitorear la salud y el rendimiento de aplicaciones paralelas y distribuidas.

• Aplicar **grep** para analizar logs de aplicaciones y sistemas, facilitando la depuración y el monitoreo.

• Utilizar pipes para crear cadenas de procesamiento de datos eficientes y scripts de análisis

• Aprender a escribir scripts de shell para automatizar tareas de administración y despliegue.

• Entender el control de flujo, manejo de variables, y funciones en Bash. • Desarrollar habilidades para la automatización de pruebas y despliegues en entornos de computación distribuida.

• Aprender a utilizar **awk** para el filtrado y transformación de datos complejos en scripts de shell.

**Entregable:**

Presenta el código completo y tus respuestas desarrollado en tu repositorio personal hasta el dia 16 de abril (8:00 PM). Recuerda presentar tus resultados en formato markdown y código si es que se ha realizado.***E***

**El comando ps**

El comando **ps** en Linux y otros sistemas tipo Unix es una herramienta de línea de comandos utilizada para mostrar información sobre los procesos activos en un sistema. **ps** es el acrónimo de "process status" o estado del proceso. Proporciona una instantánea de los procesos corriendo en ese momento, incluyendo detalles como el ID del proceso (PID), el usuario propietario del proceso, el uso de CPU, el uso de memoria, el tiempo de ejecución, el comando que inició el proceso, entre otros.

En un curso de computación paralela, concurrente y distributiva, el comando **ps** puede ser aplicado de diversas maneras para facilitar la comprensión y gestión de los procesos y la ejecución de programas en estos entornos:

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

• **Monitoreo de procesos**: **ps** puede ser usado para enseñar cómo identificar y monitorear procesos individuales o grupos de procesos relacionados con aplicaciones paralelas y concurrentes.

• **Gestión de recursos**: Utilizando **ps** junto con otras herramientas, se puede enseñar a observar el uso de CPU y memoria, lo cual es crucial para la optimización de aplicaciones en entornos paralelos y distribuidos.

• **Depuración y diagnóstico**: En la computación paralela y concurrente, identificar procesos bloqueados, zombies o que consumen recursos excesivamente es fundamental para la depuración y el mantenimiento del rendimiento del sistema. **ps** permite identificar rápidamente tales procesos.

• **Automatización y scripting**: **ps** se puede usar en scripts de shell para automatizar la supervisión y gestión de aplicaciones paralelas y distribuidas.

• **Estudio de casos**: Análisis de casos de estudio donde se requiere la identificación y gestión de procesos en sistemas de computación distribuida. Por ejemplo, cómo gestionar de manera eficiente múltiples instancias de un servicio web distribuido en un cluster de servidores.

**Ejercicios**

1. Listar todos los procesos con detalles completos

2. Buscar procesos específicos por nombre:

3. Mostrar procesos en un árbol jerárquico (útil para ver relaciones padre-hijo en procesos concurrentes):

4. Mostrar procesos de un usuario específico:

5. Escribe un script para verificar y reiniciar automáticamente un proceso si no está corriendo.

En estos scripts (recuerda son archivos Bash, terminan en .sh) verifica que efectivamente hacen lo que se indica:

**Monitoreo de procesos por uso excesivo de CPU**

#!/bin/bash

ps -eo pid,ppid,%cpu,cmd --sort=-%cpu | head -10 | while read pid ppid cpu cmd; do if (( $(echo "$cpu > 80.0" | bc -l) )); then

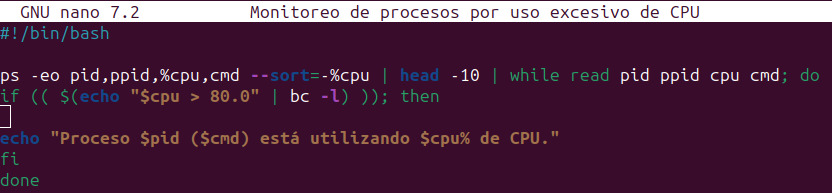
Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

echo "Proceso $pid ($cmd) está utilizando $cpu% de CPU."

fi

done



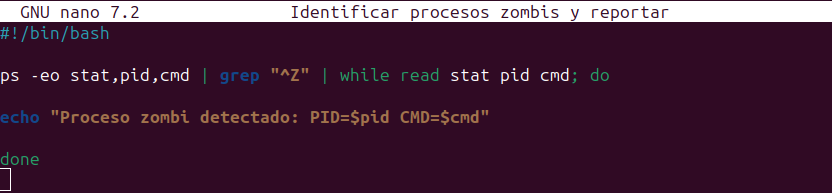
**Identificar procesos zombis y reportar**

#!/bin/bash

ps -eo stat,pid,cmd | grep "^Z" | while read stat pid cmd; do

echo "Proceso zombi detectado: PID=$pid CMD=$cmd"

done



**Reiniciar automáticamente un servicio no está corriendo**

#!/bin/bash

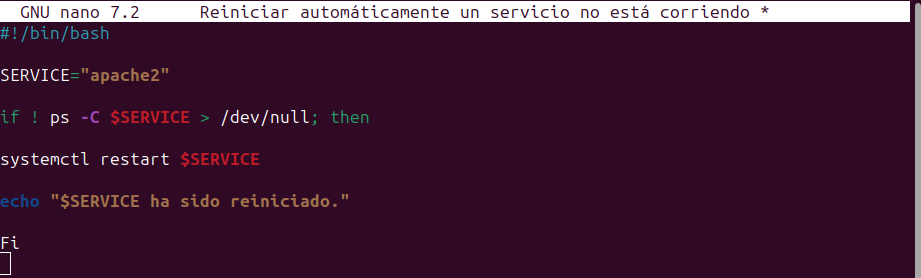
SERVICE="apache2"

if ! ps -C $SERVICE > /dev/null; then

systemctl restart $SERVICE

echo "$SERVICE ha sido reiniciado."

Fi



**Verificar la cantidad de instancias de un proceso y actuar si supera un umbral**

#!/bin/bash

PROCESS\_NAME="httpd"

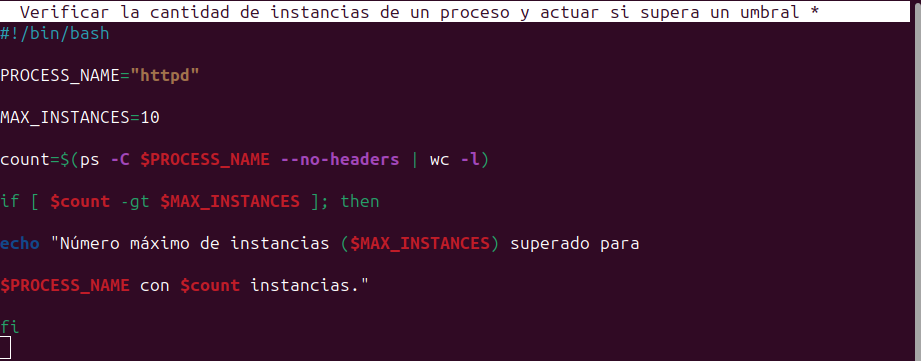
MAX\_INSTANCES=10

count=$(ps -C $PROCESS\_NAME --no-headers | wc -l)

if [ $count -gt $MAX\_INSTANCES ]; then

echo "Número máximo de instancias ($MAX\_INSTANCES) superado para $PROCESS\_NAME con $count instancias."

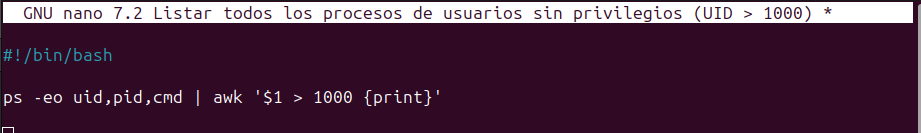
fi



**Listar todos los procesos de usuarios sin privilegios (UID > 1000)**

#!/bin/bash

ps -eo uid,pid,cmd | awk '$1 > 1000 {print}'



**Alertar sobre procesos que han estado corriendo durante más de X horas**

#!/bin/bash

MAX\_HOURS=24

ps -eo pid,etime | while read pid time; do

days=$(echo $time | grep -oP '^\d+-' | sed 's/-//')

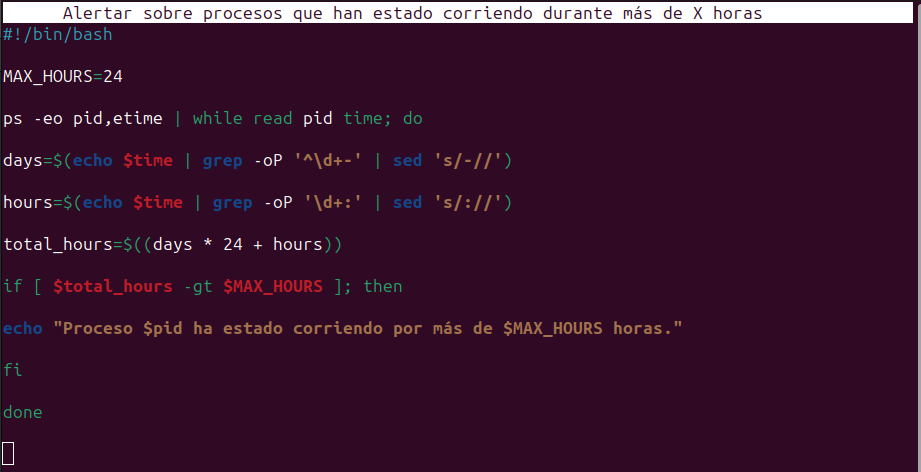
hours=$(echo $time | grep -oP '\d+:' | sed 's/://')

total\_hours=$((days \* 24 + hours))

if [ $total\_hours -gt $MAX\_HOURS ]; then

echo "Proceso $pid ha estado corriendo por más de $MAX\_HOURS horas." fi

done



**Encontrar y listar todos los procesos que escuchan en un puerto específico**

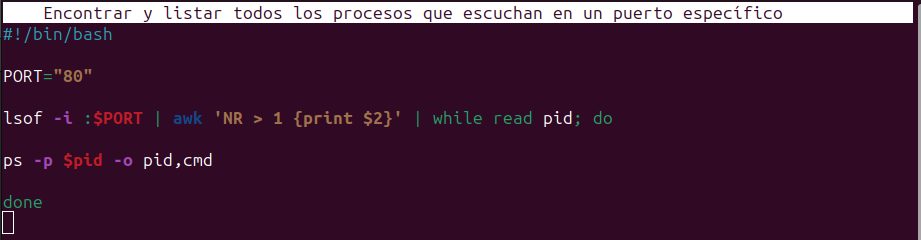
#!/bin/bash

PORT="80"

lsof -i :$PORT | awk 'NR > 1 {print $2}' | while read pid; do

ps -p $pid -o pid,cmd

done



**Monitorear la memoria utilizada por un conjunto de procesos y alertar si supera un umbral**

#!/bin/bash

PROCESS\_NAME="mysqld"

MAX\_MEM=1024 # 1GB en MB

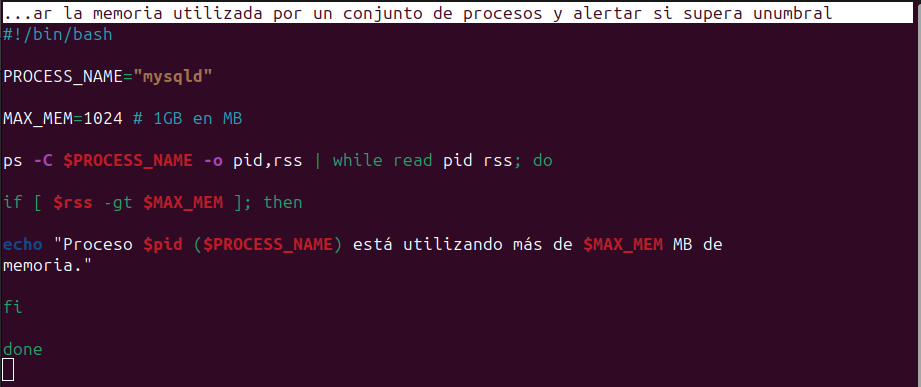
ps -C $PROCESS\_NAME -o pid,rss | while read pid rss; do

if [ $rss -gt $MAX\_MEM ]; then

echo "Proceso $pid ($PROCESS\_NAME) está utilizando más de $MAX\_MEM MB de memoria."

fi

done

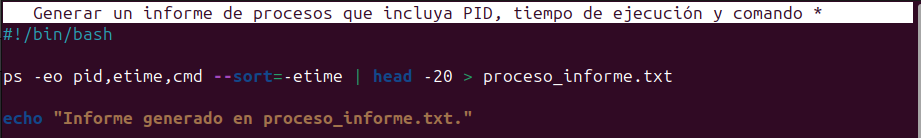


**Generar un informe de procesos que incluya PID, tiempo de ejecución y comando**

#!/bin/bash

ps -eo pid,etime,cmd --sort=-etime | head -20 > proceso\_informe.txt

echo "Informe generado en proceso\_informe.txt."



**El comando grep**

El comando **grep** es una herramienta de línea de comandos disponible en sistemas Unix y Linux utilizada para buscar texto dentro de archivos o flujos de datos. El nombre **grep** proviene de "global regular expression print", refiriéndose a su capacidad para filtrar líneas de texto que coinciden con expresiones regulares especificadas.

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

**grep** es extremadamente útil para analizar archivos de log, buscar ocurrencias de cadenas de texto en archivos de código, filtrar output de otros comandos, y muchas otras tareas de búsqueda de texto.

En el contexto de la computación paralela, concurrente y distributiva, así como en la automatización, **grep** se puede aplicar de diversas formas:

• **Análisis de logs de aplicaciones distribuidas**: **grep** puede ser utilizado para buscar rápidamente mensajes de error, advertencias o eventos específicos dentro de grandes volúmenes de archivos de log generados por aplicaciones distribuidas, facilitando el diagnóstico de problemas.

• **Monitoreo de salud del sistema**: Al integrarse en scripts de shell, **grep** puede automatizar el monitoreo del estado de servicios y procesos críticos, extrayendo información relevante de comandos como **ps**, **netstat**, o archivos como **/proc/meminfo**.

• **Validación de configuraciones en clusters**: **grep** puede ser utilizado para verificar rápidamente la consistencia de configuraciones de software en nodos de un cluster, buscando discrepancias o configuraciones erróneas en archivos distribuidos.

• **Automatización de tareas de gestión**: Integrado en scripts de shell, **grep** puede automatizar la gestión de recursos computacionales, por ejemplo, identificando y respondiendo a condiciones específicas detectadas en logs o salidas de comandos.

• **Análisis de rendimiento y carga de trabajo**: **grep** es útil para filtrar datos específicos de rendimiento y carga de trabajo de herramientas de monitoreo y métricas, permitiendo a los desarrolladores y administradores centrarse en información relevante para la optimización.

**Ejercicios**

En estos scripts (recuerda son archivos Bash, terminan en .sh) verifica que efectivamente hacen lo que se indica:

**Filtrar errores específicos en logs de aplicaciones paralelas:**

grep "ERROR" /var/log/myapp/\*.log

**Verificar la presencia de un proceso en múltiples nodos**:

pdsh -w nodo[1-10] "ps aux | grep 'my\_process' | grep -v grep"

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

**Contar el número de ocurriencias de condiciones de carrera registradas**: grep -c "race condition" /var/log/myapp.log

**Extraer IPs que han accedido concurrentemente a un recurso**:

grep "accessed resource" /var/log/webserver.log | awk '{print $1}' | sort | uniq

**Automatizar la alerta de sobrecarga en un servicio distribuido**:

grep "out of memory" /var/log/services/\*.log && mail -s "Alerta de Memoria" admin@example.com < /dev/null

**Monitorear errores de conexión en aplicaciones concurrentes**:

grep -i "connection error" /var/log/myapp\_error.log | mail -s "Errores de Conexión Detectados" admin@example.com

**Validar la correcta sincronización en operaciones distribuidas**:

grep "operation completed" /var/logs/distributed\_app/\*.log | awk '{print $5, $NF}' | sort

**Monitorizar la creación de procesos no autorizados**

#!/bin/bash

watch -n 5 'ps aux | grep -vE "(root|daemon|nobody)" | grep -v grep'

**Detectar y alertar sobre ataques de fuerza bruta SSH**

grep "Failed password" /var/log/auth.log | cut -d' ' -f11 | sort | uniq -c | sort -nr | head

**Identificar uso no autorizado de recursos en clústeres de computación**

#!/bin/bash

for host in $(cat hosts.txt); do

ssh $host "ps aux | grep -vE '(ALLOWED\_PROCESS\_1|ALLOWED\_PROCESS\_2)' | grep - v grep"

done

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

**Pipes**

Un "pipe" en Linux, simbolizado por |, es una poderosa característica de la línea de comandos que permite pasar la salida (output) de un comando directamente como entrada (input) a otro comando. Esto facilita la creación de secuencias de comandos o pipelines donde el resultado de un proceso es inmediatamente utilizado por otro, permitiendo una manipulación de datos eficiente y flexible sin necesidad de archivos intermedios.

En el contexto de la computación paralela, concurrente y distributiva, los pipes son fundamentales para procesar y analizar datos generados por múltiples procesos, monitorizar el rendimiento y estado de sistemas distribuidos y automatizar tareas administrativas complejas. Al combinar **ps**, **grep** y otros comandos de Linux, se pueden crear pipelines eficientes para la gestión y análisis de sistemas.

**Ejercicios**

Indica las actividades que realizan cada uno de los scripts (recuerda son archivos Bash y por tanto terminan en .sh y cada línea representa un script diferente)

watch "ps aux | grep '[a]pache2' | awk '{print \$1, \$2, \$3, \$4, \$11}'"

cat /var/log/myapp.log | grep "ERROR" | awk '{print $NF}' | sort | uniq -c | sort -nr systemctl --failed | grep "loaded units listed" || systemctl restart $(awk '{print $1}')

ps -eo pid,ppid,%cpu,cmd --sort=-%cpu | awk '$3 > 80 {print "Alto uso de CPU: ", $1}' | mail - s "Alerta CPU" admin@example.com

ls /var/log/\*.log | xargs -n 1 -P 5 -I {} ssh nodo\_remoto "grep 'ERROR' {} > errores\_{}.txt"

echo "8.8.8.8 www.example.com" | xargs -n 1 ping -c 1 | grep "bytes from" || echo "$(date) Fallo de ping" >> fallos\_ping.txt

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

ps -eo user,%cpu,%mem,cmd | awk '/httpd/ {cpu+=$2; mem+=$3; count++} END {print "Apache - CPU:", cpu/count, "Mem:", mem/count}'

df /home | awk '$5 > 80 {print $1}' | xargs -I {} tar -czf "{}\_$(date +%F).tar.gz" {} import subprocess

# Ejecutar el comando ps y obtener la salida

result = subprocess.run(['ps', '-eo', '%cpu,pid,cmd'], stdout=subprocess.PIPE) lines = result.stdout.decode('utf-8').strip().split('\n')

# Analizar cada línea de la salida de ps

for line in lines[1:]: # Saltar la primera línea que es la cabecera

cpu\_usage, pid, cmd = line.split(None, 2)

if float(cpu\_usage) > 80.0: # Umbral de uso de CPU

print(f"Alerta: Proceso {pid} ({cmd}) está utilizando {cpu\_usage}% de CPU") .................................................................................................................................. import subprocess

# Filtrar líneas con errores de un archivo de log

cmd = "grep 'ERROR' /var/log/myapp.log"

errors = subprocess.check\_output(cmd, shell=True).decode('utf-8').split('\n')

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

# Analizar errores y contar ocurrencias

error\_counts = {}

for error in errors:

if error in error\_counts:

error\_counts[error] += 1

else:

error\_counts[error] = 1

# Imprimir el recuento de errores

for error, count in error\_counts.items():

print(f"{error}: {count}")

..................................................................................................................................

from multiprocessing import Pool

import subprocess

def analyze\_log(file\_part):

"""Función para analizar una parte del archivo de log."""

with open(file\_part) as f:

return f.read().count('ERROR')

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

# Dividir el archivo de log en partes

subprocess.run(['split', '-l', '1000', 'large\_log.log', 'log\_part\_'])

# Lista de archivos divididos

parts = subprocess.check\_output('ls log\_part\_\*', shell=True).decode().split()

# Utilizar multiprocessing para analizar las partes en paralelo

with Pool(4) as p:

results = p.map(analyze\_log, parts)

print("Total de errores encontrados:", sum(results))

.................................................................................................................................. import subprocess

import time

previous\_ports = set()

while True:

# Ejecutar netstat y capturar la salida

result = subprocess.run(['netstat', '-tuln'], stdout=subprocess.PIPE) ports = set(line.split()[3] for line in result.stdout.decode().split('\n') if 'LISTEN' in line)

# Detectar cambios en puertos abiertos

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

new\_ports = ports - previous\_ports

closed\_ports = previous\_ports - ports

if new\_ports or closed\_ports:

print(f"Nuevos puertos abiertos: {new\_ports}, Puertos cerrados: {closed\_ports}") previous\_ports = ports

time.sleep(60) # Esperar un minuto antes de volver a verificar .................................................................................................................................. import subprocess

# Obtener uso de memoria por proceso

result = subprocess.run(['ps', '-eo', 'user,rss'], stdout=subprocess.PIPE) lines = result.stdout.decode().split('\n')

# Calcular el uso total de memoria por usuario

memory\_usage = {}

for line in lines[1:-1]: # Ignorar la primera y última línea (cabecera y línea vacía) user, rss = line.split(None, 1)

memory\_usage[user] = memory\_usage.get(user, 0) + int(rss)

for user, rss in memory\_usage.items():

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

print(f"Usuario: {user}, Memoria RSS total: {rss} KB")

..................................................................................................................................

import subprocess

import datetime

snapshot\_interval = 60 # en segundos

while True:

timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

cpu\_usage = subprocess.check\_output("top -b -n1 | awk '/Cpu\(s\):/ {print $2}'", shell=True).decode().strip()

memory\_usage = subprocess.check\_output("free | grep Mem | awk '{print $3/$2 \* 100.0}'", shell=True).decode().strip()

with open("system\_performance.log", "a") as log\_file:

log\_file.write(f"{timestamp}, CPU: {cpu\_usage}%, Memoria: {memory\_usage}%\n")

time.sleep(snapshot\_interval)

.................................................................................................................................. #!/bin/bash

while true; do

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

ps -eo %cpu,pid,cmd --sort=-%cpu | head -n 10 | awk '$1 > 80.0 { printf("Alto uso de CPU (%s%%) por PID %s: %s\n", $1, $2, $3); }' | while read LINE; do

echo "$LINE" | mail -s "Alerta de CPU" admin@domain.com done

sleep 60

done

..................................................................................................................................

#!/bin/bash

while true; do

ps -eo %cpu,pid,cmd --sort=-%cpu | head -n 10 | awk '$1 > 80.0 { printf("Alto uso de CPU (%s%%) por PID %s: %s\n", $1, $2, $3); }' | while read LINE; do

echo "$LINE" | mail -s "Alerta de CPU" admin@domain.com done

sleep 60

done

.................................................................................................................................. #!/bin/bash

ps -eo user,rss | awk '{arr[$1]+=$2} END {

for (user in arr) {

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

print user, arr[user] " KB";

}

}' | sort -nrk 2 > /tmp/memory\_usage\_by\_user.txt

echo "Uso de memoria por usuario guardado en /tmp/memory\_usage\_by\_user.txt." .................................................................................................................................. #!/bin/bash

echo "Top CPU y Memoria por Usuario"

ps -eo user,%cpu,%mem --sort=-%cpu | awk 'NR==1 {print $0; next} !seen[$1]++' | while read USER CPU MEM; do

echo "Usuario: $USER, CPU: $CPU%, Mem: $MEM%"

done

.................................................................................................................................. #!/bin/bash

PROCESS\_NAME="java"

echo "Reporte de Memoria para procesos $PROCESS\_NAME"

ps -C $PROCESS\_NAME -o pid,user,%mem,cmd --sort=-%mem | awk 'NR==1; NR>1 {print $0; total+=$3} END {print "Memoria Total Usada:", total "%"}'

.................................................................................................................................. #!/bin/bash

LOG="/var/log/httpd/access\_log"

echo "Top IPs"

awk '{print $1}' $LOG | sort | uniq -c | sort -nr | head -5 | while read COUNT IP; do LOCATION=$(geoiplookup $IP | cut -d, -f2)

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

echo "$IP ($COUNT accesos) - $LOCATION"

done

..................................................................................................................................

#!/bin/bash

NET\_DEV="eth0"

echo "Estadísticas de red para $NET\_DEV"

rx\_prev=$(cat /sys/class/net/$NET\_DEV/statistics/rx\_bytes)

tx\_prev=$(cat /sys/class/net/$NET\_DEV/statistics/tx\_bytes)

sleep 5

rx\_now=$(cat /sys/class/net/$NET\_DEV/statistics/rx\_bytes)

tx\_now=$(cat /sys/class/net/$NET\_DEV/statistics/tx\_bytes)

rx\_rate=$(( ($rx\_now - $rx\_prev) / 5 ))

tx\_rate=$(( ($tx\_now - $tx\_prev) / 5 ))

echo "RX Rate: $rx\_rate bytes/sec"

echo "TX Rate: $tx\_rate bytes/sec"

**Bash**

Para profundizar en el aprendizaje y comprensión de Bash en el contexto de computación paralela, concurrente y distribuida, necesitarán una base sólida en varios conceptos y herramientas de línea de comandos. A continuación, les presento una lista de referencias y recursos que pueden ser útiles permitiéndoles no solo entender los scripts proporcionados aquí, sino también desarrollar sus propios scripts para resolver problemas complejos en estos entornos.

"The Linux Command Line" por William Shotts https://linuxcommand.org/tlcl.php

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

La documentación oficial de Bash es un recurso indispensable para comprender todas las características y capacidades del shell. https://www.gnu.org/software/bash/manual/

Explainshell : https://explainshell.com/

Un sitio web que desglosa comandos de Bash, mostrando una explicación detallada de cada parte de un comando.

Ryan's Tutorials - Bash Scripting Tutorial https://ryanstutorials.net/bash-scripting-tutorial/

Una herramienta de linting para scripts de Bash que ayuda a encontrar errores y problemas comunes en el código. ShellCheck: https://www.shellcheck.net/

Una herramienta para ejecutar tareas en paralelo utilizando la línea de comandos. Es muy útil para procesamiento de datos y tareas computacionales que pueden ser paralelizadas. GNU Parallel https://www.gnu.org/software/parallel/

Para escribir scripts efectivos en Bash, especialmente en el contexto de computación paralela, concurrente y distribuida, es esencial dominar ciertos fundamentos y conceptos avanzados de Bash. A continuación, presento un resumen de los aspectos clave de Bash que debes conocer, acompañados de ejemplos ilustrativos.

**Variables**: Almacenar y manipular datos.

nombre="Mundo"

echo "Hola, $nombre"

**Estructuras de Control**: Permiten tomar decisiones y repetir acciones. # If statement

if [ "$nombre" == "Mundo" ]; then

echo "Correcto"

fi

# Loop

for i in {1..5}; do

echo "Iteración $i"

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

done

**Funciones**: Agrupar código para reutilizar.

saludo() {

echo "Hola, $1"

}

saludo "Mundo"

**Comandos comunes** (**grep**, **awk**, **sed**, **cut**, **sort**, **uniq**): Procesamiento de texto y datos. echo -e "manzana\nbanana\nmanzana" | sort | uniq

**Pipes y redirecciones**: Conectar la salida de un comando con la entrada de otro. cat archivo.txt | grep "algo" > resultado.txt

**Expresiones regulares**: Patrones para buscar y manipular texto.

echo "error 404" | grep -Eo "[0-9]+"

**Manejo de argumentos**: Scripts que aceptan entrada del usuario.

#!/bin/bash

echo "Ejecutando script con el argumento: $1"

**Automatización y monitoreo**: Scripts para automatizar tareas y monitorear sistemas. #!/bin/bash

if ps aux | grep -q "[a]pache2"; then

echo "Apache está corriendo."

else

echo "Apache no está corriendo."

fi

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

**Procesamiento Paralelo con GNU Parallel**: Ejecutar tareas en paralelo para optimizar el tiempo de procesamiento.

cat lista\_urls.txt | parallel wget

**Validación de entradas**: Prevenir la ejecución de comandos maliciosos. read -p "Introduce tu nombre: " nombre

echo "Hola, $nombre" # Asegúrate de validar o escapar $nombre si se usa en comandos más complejos.

**Optimización de scripts**: Utilizar herramientas y técnicas para reducir el tiempo de ejecución.

find . -name "\*.txt" | xargs grep "patrón"

**Ejercicios**

Indica las actividades que realizan cada uno de los scripts (recuerda son archivos Bash y por tanto terminan en .sh.

#!/bin/bash

# Configuración

UMBRAL\_CPU=70.0 # Uso máximo de CPU permitido (%)

UMBRAL\_MEM=500 # Uso máximo de memoria permitido (MB)

LOG\_FILE="/var/log/monitoreo\_procesos.log"

EMAIL\_ADMIN="admin@ejemplo.com"

PROCESOS\_PARALELOS=("proceso1" "proceso2" "proceso3") # Nombres de los procesos a monitorear

# Función para convertir memoria de KB a MB

convertir\_kb\_a\_mb() {

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

echo "$(( $1 / 1024 ))"

}

# Función para obtener y verificar el uso de recursos de los procesos

verificar\_procesos() {

for PROC in "${PROCESOS\_PARALELOS[@]}"; do

ps -C $PROC -o pid=,%cpu=,%mem=,vsz=,comm= --sort=-%cpu | while read PID CPU MEM VSZ COMM; do

MEM\_MB=$(convertir\_kb\_a\_mb $VSZ)

if (( $(echo "$CPU > $UMBRAL\_CPU" | bc -l) )) || [ "$MEM\_MB" -gt "$UMBRAL\_MEM" ]; then

echo "$(date +"%Y-%m-%d %H:%M:%S") - Proceso $COMM (PID $PID) excede los umbrales con CPU: $CPU%, MEM: ${MEM\_MB}MB" >> $LOG\_FILE

kill -9 $PID && echo "$(date +"%Y-%m-%d %H:%M:%S") - Proceso $PID terminado." >> $LOG\_FILE

echo "Proceso $PID ($COMM) terminado por alto uso de recursos" | mail -s "Alerta de Proceso Terminado" $EMAIL\_ADMIN

fi

done

done

}

# Loop principal para el monitoreo continuo

while true; do

verificar\_procesos

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

sleep 60 # Espera 60 segundos antes de la próxima verificación done

.................................................................................................................................. #!/bin/bash

DIRECTORIOS=("dir1" "dir2" "dir3")

DESTINO\_BACKUP="/mnt/backup"

backup\_dir() {

dir=$1

fecha=$(date +%Y%m%d)

tar -czf "${DESTINO\_BACKUP}/${dir##\*/}\_$fecha.tar.gz" "$dir" echo "Backup completado para $dir"

}

export -f backup\_dir

export DESTINO\_BACKUP

parallel backup\_dir ::: "${DIRECTORIOS[@]}"

.................................................................................................................................. #!/bin/bash

NODOS=("nodo1" "nodo2" "nodo3")

TAREAS=("tarea1.sh" "tarea2.sh" "tarea3.sh")

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

distribuir\_tareas() {

for i in "${!TAREAS[@]}"; do

nodo=${NODOS[$((i % ${#NODOS[@]}))]}

tarea=${TAREAS[$i]}

echo "Asignando $tarea a $nodo"

scp "$tarea" "${nodo}:/tmp"

ssh "$nodo" "bash /tmp/$tarea" &

done

wait

}

distribuir\_tareas

.................................................................................................................................. #!/bin/bash

LOCK\_FILE="/var/lock/mi\_recurso.lock"

RECURSO="/path/to/recurso\_compartido"

adquirir\_lock() {

while ! (set -o noclobber; > "$LOCK\_FILE") 2> /dev/null; do echo "Esperando por el recurso..."

sleep 1

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

done

}

liberar\_lock() {

rm -f "$LOCK\_FILE"

}

adquirir\_lock

# Trabajar con el recurso

echo "Accediendo al recurso"

sleep 5 # Simular trabajo

liberar\_lock

.................................................................................................................................. #!/bin/bash

NODOS=("nodo1" "nodo2" "nodo3")

ARCHIVO\_METRICAS="/tmp/metricas\_$(date +%Y%m%d).csv"

recolectar\_metricas() {

echo "Nodo,CPU(%),Memoria(%),Disco(%)" > "$ARCHIVO\_METRICAS" for nodo in "${NODOS[@]}"; do

ssh "$nodo" "

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

cpu=\$(top -bn1 | grep 'Cpu(s)' | sed 's/.\*, \*\([0-9.]\*\)%\* id.\*/\1/' | awk '{print 100 - \$1}');

memoria=\$(free | awk '/Mem:/ {print \$3/\$2 \* 100.0}');

disco=\$(df / | awk 'END{print $(NF-1)}');

echo \"\$HOSTNAME,\$cpu,\$memoria,\$disco\";

" >> "$ARCHIVO\_METRICAS"

done

}

recolectar\_metricas

**awk**

Awk es una herramienta de scripting extremadamente poderosa y versátil para procesar y analizar datos en Unix/Linux. Es especialmente útil para manipular datos textuales y produce resultados formatados. **awk** funciona leyendo archivos o flujos de entrada línea por línea, dividiendo cada línea en campos, procesándola con acciones definidas por el usuario y luego imprimiendo la salida.

En el contexto de la computación paralela y concurrente, **awk** puede ser utilizado para analizar y procesar datos generados por procesos, monitorizar el rendimiento del sistema, y preparar datos para ser procesados en paralelo. Cuando se combina con pipes de Linux y expresiones regulares, **awk** se convierte en una herramienta aún más potente, permitiendo a los usuarios filtrar, procesar y redirigir la salida de comandos en secuencias complejas de operaciones.

**awk** puede ser usado para extraer información específica de la lista de procesos generada por el comando **ps**.

ps aux | awk '{print $1, $2, $3, $4, $11}' | head -n 10

Pregunta: ¿Qué hace y cual es el resultado del código anterior?

**awk** puede ser utilizado para preparar y filtrar datos que necesiten ser procesados en paralelo. Por ejemplo, puedes dividir un archivo grande en múltiples archivos más pequeños basados en algún criterio, que luego pueden ser procesados en paralelo:

awk '{print > ("output" int((NR-1)/1000) ".txt")}' input.txt

Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

Pregunta: Comprueba con este archivo de texto el anterior script:

https://babel.upm.es/~angel/teaching/pps/quijote.txt

La combinación de **awk** con pipes y expresiones regulares expande significativamente sus capacidades de procesamiento de texto. Por ejemplo, para monitorizar archivos de log en busca de errores y filtrar mensajes relevantes:

tail -f /var/log/app.log | grep "ERROR" | awk '{print $1, $2, $NF}'

Pregunta: ¿puedes comprobar cual es el resultado si aplicas el script anterior en tus archivos logs?

Pregunta: ¿cuál es el resultado de utilizar este script (usa apache)?

ps -eo user,pid,pcpu,pmem,cmd | grep apache2 | awk '$3 > 50.0 || $4 > 50.0 {print "Alto recurso: ", $0}'

**Ejercicios:**

¿Cuál es la salida de los siguientes scripts (recuerda que son archivos de texto en bash)

1. ps -eo pid,pcpu,pmem,cmd | awk '$2 > 10.0 || $3 > 10.0'

2. awk '{print $0 >> ("output-" $4 ".log")}' /var/log/syslog

3. grep "Failed password" /var/log/auth.log | awk '{print $(NF-3)}' | sort | uniq -c | sort -nr 4. inotifywait -m /path/to/dir -e create | awk '{print "Nuevo archivo creado:", $3}' 5. find . -type f -name "\*.py" -exec ls -l {} + | awk '{sum += $5} END {print "Espacio total usado por archivos .py: ", sum}'

6. awk '{sum+=$NF} END {print "Tiempo promedio de respuesta:", sum/NR}' access.log 7. ps -eo state | awk '/D/ {d++} /R/ {r++} END {print "Espera (D):", d, "- Ejecución (R):", r}'

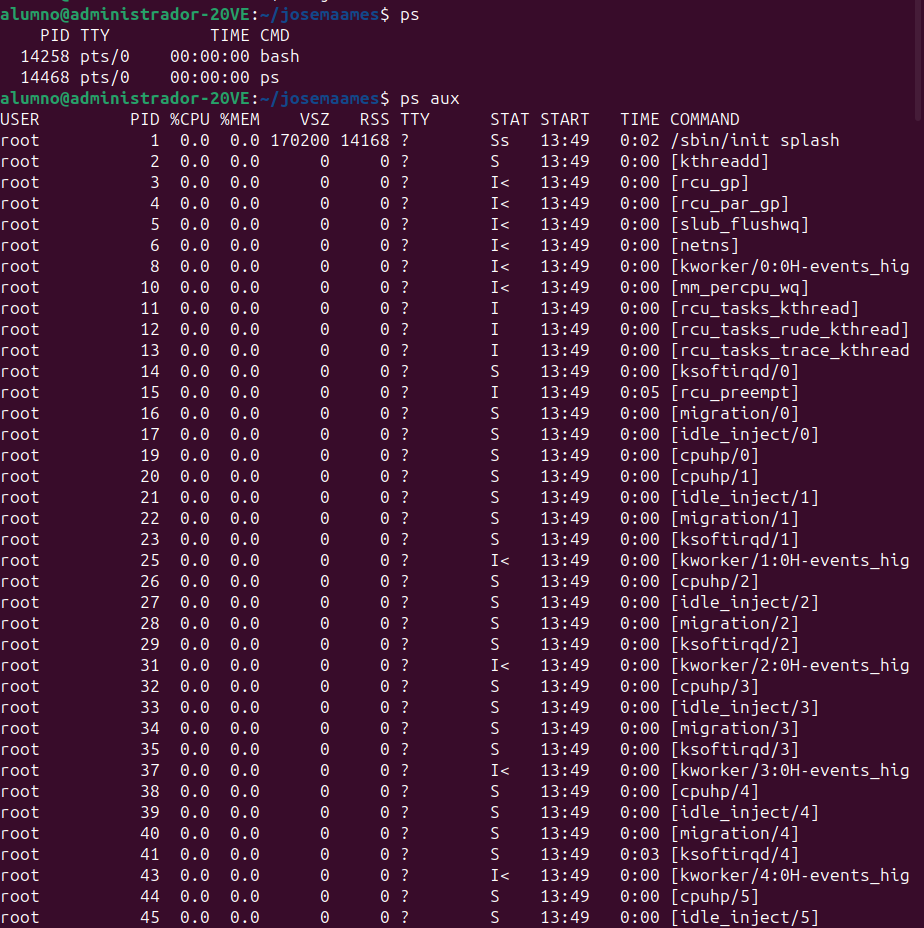
8. ps -eo state | awk '/D/ {d++} /R/ {r++} END {print "Espera (D):", d, "- Ejecución (R):", r}'

9. awk '/SwapTotal/ {total=$2} /SwapFree/ {free=$2} END {if ((total-free)/total\*100 > 20.0) print "Alerta: Uso excesivo de swap"}' /proc/meminfo

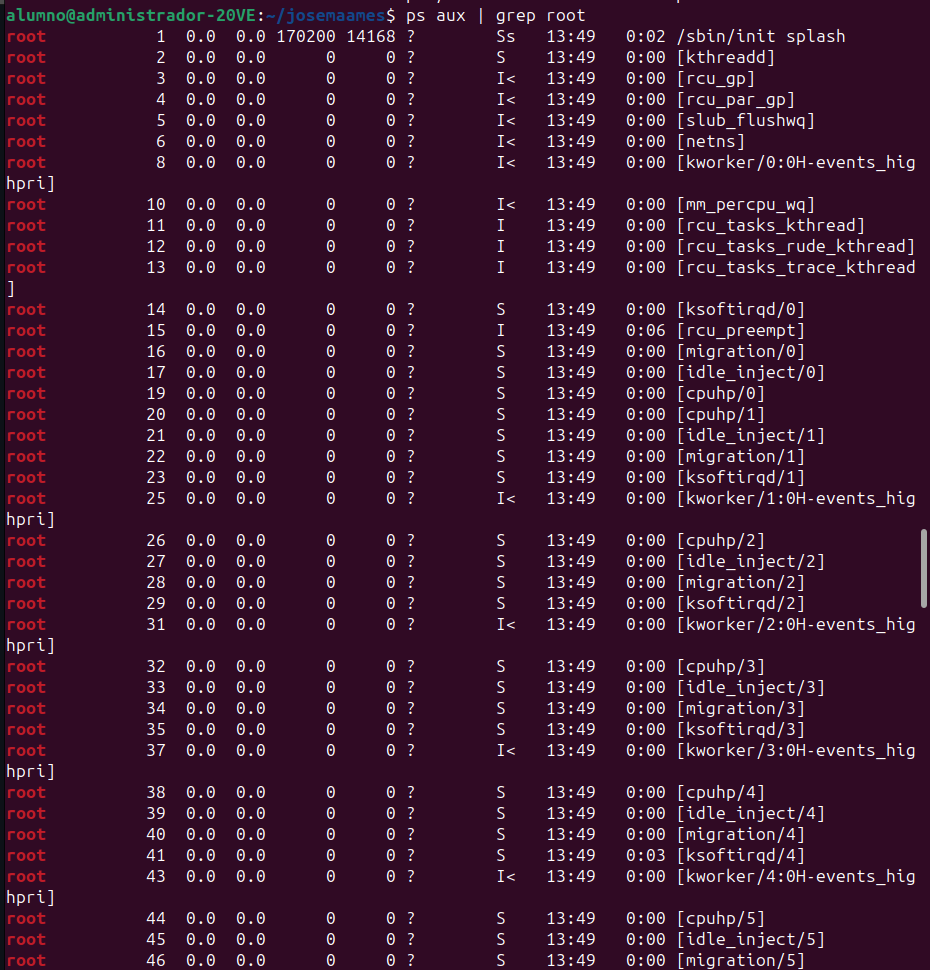
10. ls -l | awk '!/^total/ && !/^d/ {sum += $5} END {print "Uso total de disco (sin subdirectorios):", sum}'

DESARROLLO:

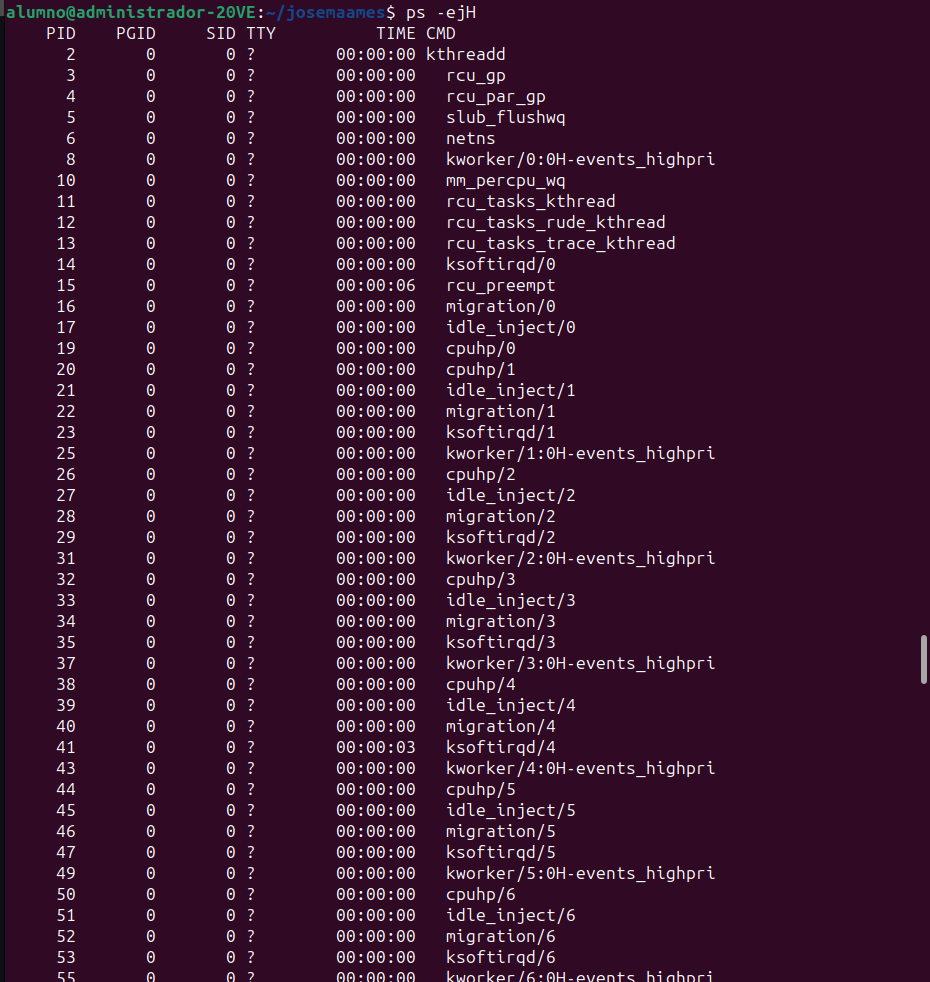
Listar todos los procesos con detalles completos:



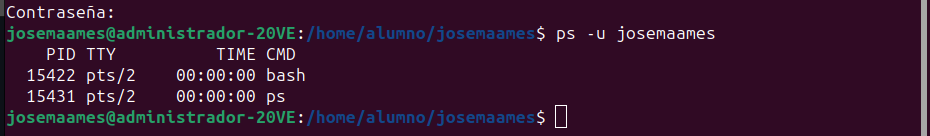
Buscar procesos específicos por nombre:



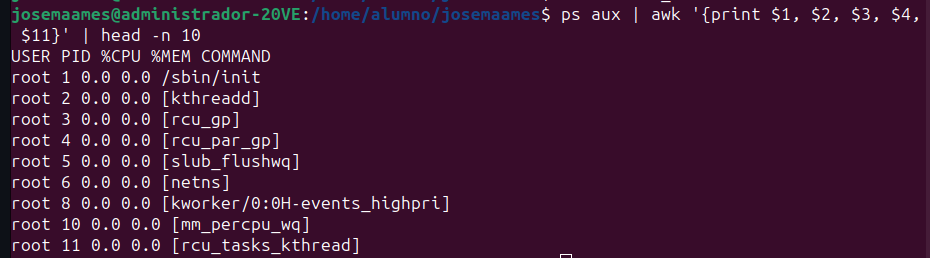
Enseñar procesos en un árbol jerárquico (útil para ver relaciones padre-hijo en procesos concurrentes):



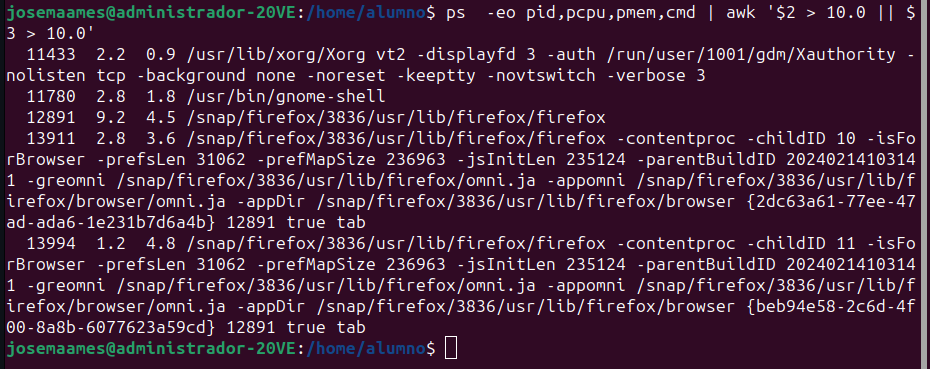
Los procesos más de un usuario específico:



Ejercicios AWK







Computación Paralela y Distribuida

**Departamento Académico de Ingeniería C8286-Computación Paralela y Distribuida** 

***scriba us r***

***espuestas aquí.*** *Fin del documento*

Computación Paralela y Distribuida