Ignacio Grané Rojas - Carlos Andrés Mata Calderón
Escuela de Ingeniería en Computadores
Tecnológico de Costa Rica

September 3, 2024

TALLER #2 — Procesos en un entorno de Linux

1. Preguntas guía

1.1. Explique las etapas de creación de un proceso en Windows (CreateProcess)

R/· Para poder crear exitosamente un proceso en Windows es necesario que el `CreateProcess` pase por las siguientes etapas:

- **Validación de parámetros**: Se debe primeramente de verificar los parámetros antes de procedeer. Esto es, validar que los datos sean correctos y estén en el formato esperado para el nombre de la aplicación, los argumentos, las opciones de creación, los identificadores de seguridad, entre otros.
- **Resolución del archivo ejecutable**: Se busca determinar la ruta completa del archivo ejecutable que se va a cargar.
- **Creación de la estructura del proceso y del hilo principal:** Crea las estructuras de datos necesarias para el nuevo proceso y su primer hilo de ejecución. Esto es, establecen las referencias en la tabla de procesos e hilos del sistema operativo.
- **Asignación del espacio de direcciones**: Se asigna el espacio de direcciones virtuales para el nuevo proceso.
- **Carga del archivo ejecutable**: Se mapea el archivo ejecutable en la memoria del nuevo proceso, lo que implica cargar la sección del código, inicializar las secciones de datos, y configurar los punteros necesarios para la ejecución.
- **Inicialización del contexto del hilo principal**: El sistema operativo configura el contexto inicial del hilo principal, que incluye los registros del CPU, el PC, y el *stack pointer*, para que el hilo pueda comenzar a ejecutarse desde el programa.
- **Notificación a los módulos involucrados**: Informar a diversos subsistemas y componentes de que se ha creado un nuevo proceso.
- **Ejecución del hilo principal**: Una vez configurado todo, el sistema operativo transfiere el control al hilo principal del nuevo proceso, iniciando su ejecución. Esto implica que el hilo comienza a ejecutar las instrucciones desde el punto de entrada definido en el archivo ejecutable.
- **Retorno de la Función `CreateProcess`**: Finalmente, la función `CreateProcess` devuelve el control al proceso padre que la invocó, proporcionando los identificadores del proceso e hilo creados junto con cualquier código de error que pueda haber ocurrido durante el proceso de creación.

1.2. ¿Cuáles son las variables necesarias que se deben de guardar cuando se quiere implementar un cambio de contexto?

R/· A la hora de realizar un cambio de contexto es fundamental guardar el estado completo del proceso o hilo que está siendo desactivado para poder reanudarlo correctamente más adelante. Este guardado es realizado mediante una estructura de datos específica, a menudo llamada *bloque de control de proceso* (PCB), y o TCB para el caso de los hilos. Cuando se reactiva el proceso o hilo, este contexto debe ser restaurado para que continúe su ejecución desde el punto exacto donde fue interrumpido. Las variables necesarias que se deben de guardar son, principalmente, las siguientes:

- **Registros del CPU**: Los registros contienen datos temporales y variables utilizadas durante la ejecución del proceso.

  Estos registros incluyen:

  - **Puntero de Pila (Stack Pointer, SP)**: Se encarga de apuntar a la cima de la pila del proceso.
  - **Puntero Base (Base Pointer, BP)**: Utilizado para referencia a datos locales en la pila.
  - **Contador de Programa (PC)**: Indica la dirección de la próxima instrucción a ejecutar.
  - **Banderas**: Contiene las banderas de condición y control, como las banderas de acarreo o cero.

- **Estado de la pila:** Es indispensable guardar todos los datos almacenados en la pila actual del proceso en ejecución.

- **Registros de control**: Incluyen registros específicos del sistema asociados al control, por ejemplo el control de la memoria o el modo de operación del CPU.

- **Registros de la tabla de páginas**: En un sistema con paginación, se debe guardar el estado de los registros que apuntan a la tabla de páginas del proceso, para así restaurar el espacio de direcciones virtuales correcto.

- **Registros de punto flotante**: Se deben de almacenar los registros que se encuentran trabajando con operaciones de punto flotante.

- **Registros de interrupciones**: Indican qué interrupciones están habilitadas o deshabilitadas en el momento del cambio de contexto.

- **Estado del procesador**: En arquitecturas de mayor complejidad, el estado del procesador también puede incluir información sobre el modo de operación (si se encuentra en modo usuario o modo núcleo), el estado de ejecución (por ejemplo si está en una llamada al sistema), y otras configuraciones específicas de la arquitectura.

- **Información de Control del Proceso**: El proceso en cuestión que se desea pasar a "segundo plano" tiene información asociada, que típicamente suele ser:

  - **Identificadores de proceso o hilo**: Identificadores únicos del proceso o hilo en ejecución, que se deben de guardar para mantener el contexto.
  - **Prioridad del proceso o hilo**: La prioridad actual con la que el proceso o hilo está ejecutándose.
  - **Tiempo de ejecución**: Estadísticas del tiempo que el proceso ha estado ejecutándose.

1.3. ¿Cómo se podría implementar un cambio de contexto por hardware y no por software? Realice un esquema de arquitectura con su propuesta

R/· Para dicha implementación, sería primordialmente necesario definir los componentes de hardware necesarios, siendo estos:

1. **Unidad de guardado de contexto**: Se encargaría de guardar automáticamente el contexto del proceso saliente cuando se detecta un cambio de contexto. Este buscaría guardar, principalmente, algunos atributos como los siguientes:

   - **Registros A y B**: Dos conjuntos de registros donde uno puede almacenar el contexto actual mientras que el otro está listo para cargarse con el nuevo contexto.
   - **Registros de control**: Guarda el estado de los registros de control del CPU.
   - **Memoria**: Guarda y restaura la configuración de la unidad de administración de memoria.

2. **Unidad de carga de contexto**: Una vez guardado el contexto anterior, se deberá de cargar el nuevo.

3. **Manejador de cola de tareas**: Gestiona la cola de tareas listas para ejecución, por lo que determina el próximo proceso a ejecutar basándose en prioridades o en un esquema de planificación.

4. **Unidad de detección de interrupciones**: Gestiona las interrupciones y notifica a la unidad de guardado de contexto y la unidad de carga de contexto para que realicen el cambio de contexto pertinente.

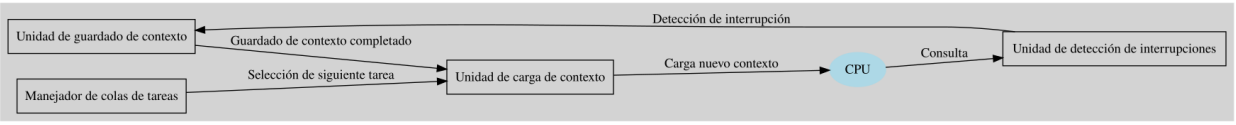


Figura 1: Diagrama de idea de interrupciones por componentes de hardware

1.4. Para que sirve el comando ps y top en un entorno de Linux

R/· Ambos corresponden a comandos utilizados para monitorear los procesos del sistema en tiempos de ejecución, aunque cada uno cumple con un propósito específico distinto.

El comando `ps` es mayormente utilizado cuando se desea obtener información acerca de los procesos actuales que está ejecutando el sistema con una vista estática. `top`, por su parte, actualiza continuamente la información, por lo que es considerada como una herramienta interactiva que proporciona una vista dinámica en tiempo real de los procesos en ejecución.

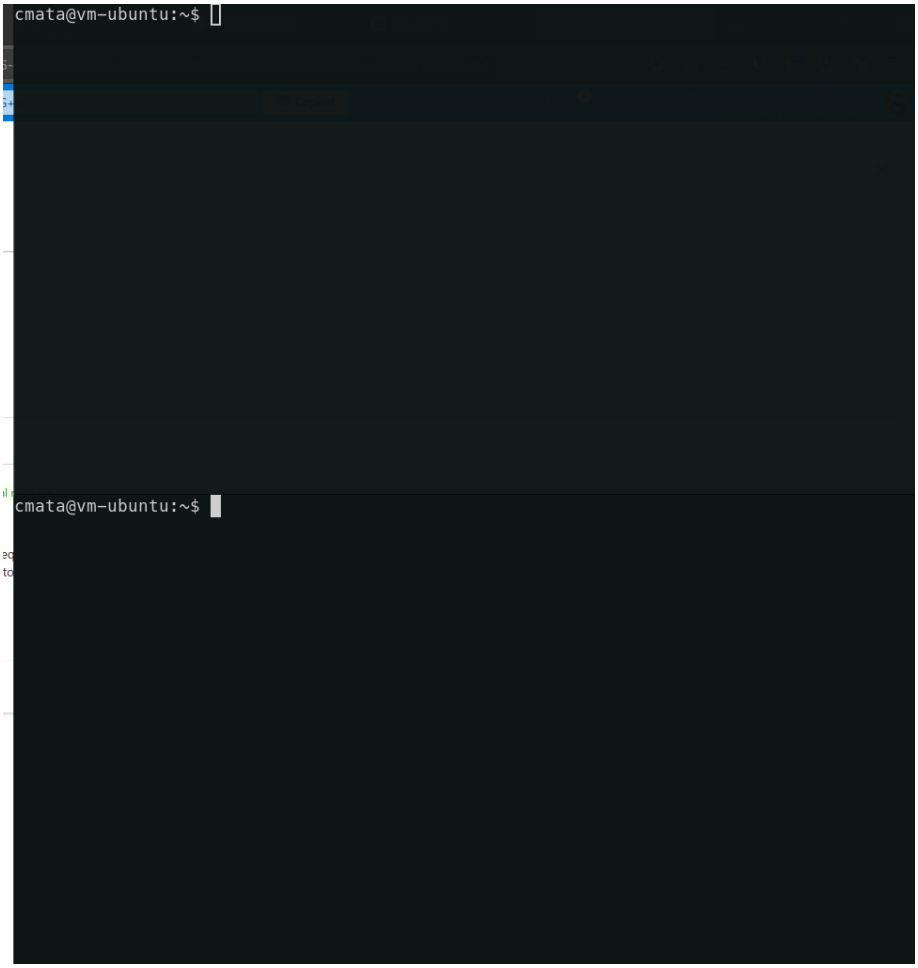
1.5. Investigue los posibles estados de un proceso en un entorno de Linux y cómo se representan

R/· Existen ocho posibles estados:

- **R: En ejecución o listo para ejecutarse**: El proceso está en ejecución o está listo para ejecutarse. Si el proceso no está actualmente en ejecución, está esperando su turno para ser ejecutado por el calendarizador del CPU.
- **S: En espera interrumpible**: : El proceso está esperando que ocurra un evento específico, como la llegada de datos desde un dispositivo de entrada/salida.
- **D: En espera no interrumpible**: El proceso está en espera de una operación de hardware que no puede ser interrumpida. Este estado suele estar relacionado con operaciones de E/S en las que el proceso no puede ser interrumpido hasta que la operación se complete.
- **T: Detenido**: El proceso está detenido, ya que habría recibido una señal de detenerse. Un proceso en este estado no ejecuta ninguna instrucción hasta que se reanude.
- **Z: Proceso zombi**: Se usa para indicar que un proceso ha terminado, pero su entrada en la tabla de procesos aún no ha sido limpiada por su proceso padre. Esto ocurre cuando el proceso padre aún no ha leído el estado de salida del proceso hijo. Un proceso en este estado ya no consume recursos del sistema, salvo la entrada en la tabla de procesos.
- **X: Proceso muerto**: Este es un estado transitorio en el que el proceso ha terminado de ejecutarse y está en proceso de ser eliminado del sistema.
- **I: Inactivo (para tareas del kernel)**: Se utiliza en algunos casos para describir tareas en el kernel que están inactivas.
- **P: Parado (para tareas del kernel)**: Es utilizado para tareas del kernel que están esperando para ser reutilizadas.

2. Procesos en Linux

2.1. Conéctese a su máquina virtual por medio de SSH (También lo puede hacer local). Acceda mediante dos conexiones, es decir, dos consolas.

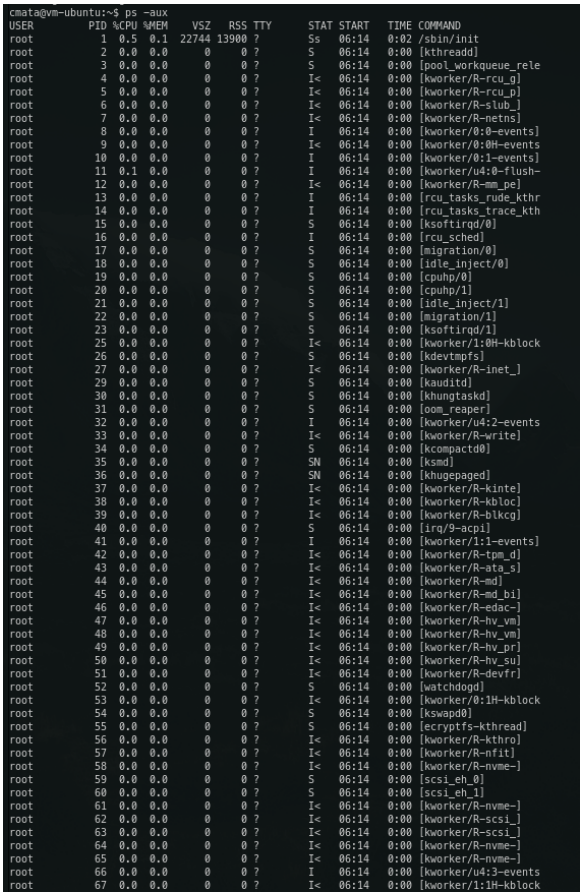

cmata@vm-ubuntu:~$
cmata@vm-ubuntu:~$

2.2. Ejecute el comando: explique cual es el significado de aux.

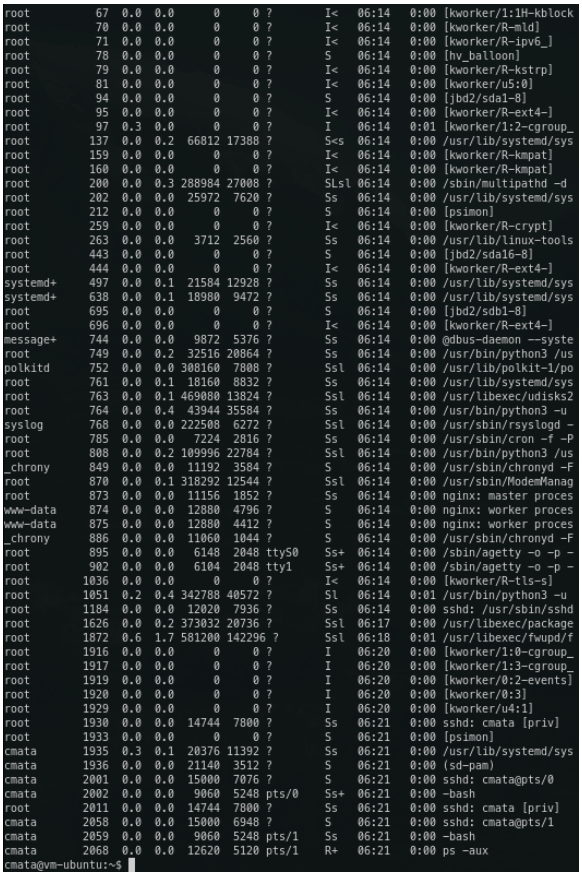
ps −aux

Permite a los usuarios ver, administrar y monitorear los procesos en ejecución [1].
Donde los argumentos signfica:

- a: Muestra procesos de todos los usuarios.
- u: Muestra una salida detallada que incluye el usuario que posee el proceso, el uso de CPU, el uso de memoria, entre otros.
- x: Muestra procesos que no están asociados a una terminal (TTY). Esto incluye muchos procesos del sistema y servicios que corren en el fondo.



```
cmata@vm-ubuntu:~$ ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.5  0.1  22744 13900 ?        Ss   06:14   0:02 /sbin/init
root           2  0.0  0.0      0     0 ?        S    06:14   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    06:14   0:00 [pool_workqueue_rele
root           4  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-rcu_g]
root           5  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-rcu_p]
root           6  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-slub_]
root           7  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-netns]
root           8  0.0  0.0      0     0 ?        I    06:14   0:00 [kworker/0:0-events]
root           9  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/0:0H-events
root          10  0.0  0.0      0     0 ?        I    06:14   0:00 [kworker/0:1-events]
root          11  0.1  0.0      0     0 ?        I    06:14   0:00 [kworker/u4:0-flush-
root          12  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-mm_pe]
root          13  0.0  0.0      0     0 ?        I    06:14   0:00 [rcu_tasks_rude_kthr
root          14  0.0  0.0      0     0 ?        I    06:14   0:00 [rcu_tasks_trace_kth
root          15  0.0  0.0      0     0 ?        S    06:14   0:00 [ksoftirqd/0]
root          16  0.0  0.0      0     0 ?        I    06:14   0:00 [rcu_sched]
root          17  0.0  0.0      0     0 ?        S    06:14   0:00 [migration/0]
root          18  0.0  0.0      0     0 ?        S    06:14   0:00 [idle_inject/0]
root          19  0.0  0.0      0     0 ?        S    06:14   0:00 [cpuhp/0]
root          20  0.0  0.0      0     0 ?        S    06:14   0:00 [cpuhp/1]
root          21  0.0  0.0      0     0 ?        S    06:14   0:00 [idle_inject/1]
root          22  0.0  0.0      0     0 ?        S    06:14   0:00 [migration/1]
root          23  0.0  0.0      0     0 ?        S    06:14   0:00 [ksoftirqd/1]
root          25  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/1:0H-kblock
root          26  0.0  0.0      0     0 ?        S    06:14   0:00 [kdevtmpfs]
root          27  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-inet_]
root          29  0.0  0.0      0     0 ?        S    06:14   0:00 [kauditd]
root          30  0.0  0.0      0     0 ?        S    06:14   0:00 [khungtaskd]
root          31  0.0  0.0      0     0 ?        S    06:14   0:00 [oom_reaper]
root          32  0.0  0.0      0     0 ?        I    06:14   0:00 [kworker/u4:2-events
root          33  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-write]
root          34  0.0  0.0      0     0 ?        S    06:14   0:00 [kcompactd0]
root          35  0.0  0.0      0     0 ?        SN   06:14   0:00 [ksmd]
root          36  0.0  0.0      0     0 ?        SN   06:14   0:00 [khugepaged]
root          37  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-kinte]
root          38  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-kbloc]
root          39  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-blkcg]
root          40  0.0  0.0      0     0 ?        S    06:14   0:00 [irq/9-acpi]
root          41  0.0  0.0      0     0 ?        I    06:14   0:00 [kworker/1:1-events]
root          42  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-tpm_d]
root          43  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-ata_s]
root          44  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-md]
root          45  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-md_bi]
root          46  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-edac-]
root          47  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-hv_vm]
root          48  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-hv_vm]
root          49  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-hv_pr]
root          50  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-hv_su]
root          51  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-devfr]
root          52  0.0  0.0      0     0 ?        S    06:14   0:00 [watchdogd]
root          53  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/0:1H-kblock
root          54  0.0  0.0      0     0 ?        S    06:14   0:00 [kswapd0]
root          55  0.0  0.0      0     0 ?        S    06:14   0:00 [ecryptfs-kthread]
root          56  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-kthro]
root          57  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-nfit]
root          58  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-nvme-]
root          59  0.0  0.0      0     0 ?        S    06:14   0:00 [scsi_eh_0]
root          60  0.0  0.0      0     0 ?        S    06:14   0:00 [scsi_eh_1]
root          61  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-nvme-]
root          62  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-scsi_]
root          63  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-scsi_]
root          64  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-nvme-]
root          65  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-nvme-]
root          66  0.0  0.0      0     0 ?        I    06:14   0:00 [kworker/u4:3-events
root          67  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/1:1H-kblock
```



```
root          67  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/1:1H-kblock
root          70  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-mld]
root          71  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-ipv6_]
root          78  0.0  0.0      0     0 ?        S    06:14   0:00 [hv_balloon]
root          79  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-kstrp]
root          81  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/u5:0]
root          94  0.0  0.0      0     0 ?        S    06:14   0:00 [jbd2/sda1-8]
root          95  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-ext4-]
root          97  0.3  0.0      0     0 ?        I    06:14   0:01 [kworker/1:2-cgroup_
root         137  0.0  0.2  66812 17388 ?        S<s  06:14   0:00 /usr/lib/systemd/sys
root         159  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-kmpat]
root         160  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-kmpat]
root         200  0.0  0.3 288984 27008 ?        SLsl 06:14   0:00 /sbin/multipathd -d
root         202  0.0  0.0  25972  7620 ?        Ss   06:14   0:00 /usr/lib/systemd/sys
root         212  0.0  0.0      0     0 ?        S    06:14   0:00 [psimon]
root         259  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-crypt]
root         263  0.0  0.0   3712  2560 ?        Ss   06:14   0:00 /usr/lib/linux-tools
root         443  0.0  0.0      0     0 ?        S    06:14   0:00 [jbd2/sda16-8]
root         444  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-ext4-]
systemd+     497  0.0  0.1  21584 12928 ?        Ss   06:14   0:00 /usr/lib/systemd/sys
systemd+     638  0.0  0.1  18980  9472 ?        Ss   06:14   0:00 /usr/lib/systemd/sys
root         695  0.0  0.0      0     0 ?        S    06:14   0:00 [jbd2/sdb1-8]
root         696  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-ext4-]
message+     744  0.0  0.0   9872  5376 ?        Ss   06:14   0:00 @dbus-daemon --syste
root         749  0.0  0.2  32516 20864 ?        Ss   06:14   0:00 /usr/bin/python3 /us
polkitd      752  0.0  0.0 308160  7808 ?        Ssl  06:14   0:00 /usr/lib/polkit-1/po
root         761  0.0  0.1  18160  8832 ?        Ss   06:14   0:00 /usr/lib/systemd/sys
root         763  0.0  0.1 469080 13824 ?        Ssl  06:14   0:00 /usr/libexec/udisks2
root         764  0.0  0.4  43944 35584 ?        Ss   06:14   0:00 /usr/bin/python3 -u
syslog       768  0.0  0.0 222508  6272 ?        Ssl  06:14   0:00 /usr/sbin/rsyslogd -
root         785  0.0  0.0   7224  2816 ?        Ss   06:14   0:00 /usr/sbin/cron -f -P
root         808  0.0  0.2 109996 22784 ?        Ssl  06:14   0:00 /usr/bin/python3 /us
_chrony      849  0.0  0.0  11192  3584 ?        S    06:14   0:00 /usr/sbin/chronyd -F
root         870  0.0  0.1 318292 12544 ?        Ssl  06:14   0:00 /usr/sbin/ModemManag
root         873  0.0  0.0  11156  1852 ?        Ss   06:14   0:00 nginx: master proces
www-data     874  0.0  0.0  12880  4796 ?        S    06:14   0:00 nginx: worker proces
www-data     875  0.0  0.0  12880  4412 ?        S    06:14   0:00 nginx: worker proces
_chrony      886  0.0  0.0  11060  1044 ?        S    06:14   0:00 /usr/sbin/chronyd -F
root         895  0.0  0.0   6148  2048 ttyS0    Ss+  06:14   0:00 /sbin/agetty -o -p -
root         902  0.0  0.0   6104  2048 tty1     Ss+  06:14   0:00 /sbin/agetty -o -p -
root        1036  0.0  0.0      0     0 ?        I<   06:14   0:00 [kworker/R-tls-s]
root        1051  0.2  0.4 342788 40572 ?        Sl   06:14   0:01 /usr/bin/python3 -u
root        1184  0.0  0.0  12020  7936 ?        Ss   06:14   0:00 sshd: /usr/sbin/sshd
root        1626  0.0  0.2 373032 20736 ?        Ssl  06:17   0:00 /usr/libexec/package
root        1872  0.6  1.7 581200 142296 ?       Ssl  06:18   0:01 /usr/libexec/fwupd/f
root        1916  0.0  0.0      0     0 ?        I    06:20   0:00 [kworker/1:0-cgroup_
root        1917  0.0  0.0      0     0 ?        I    06:20   0:00 [kworker/1:3-cgroup_
root        1919  0.0  0.0      0     0 ?        I    06:20   0:00 [kworker/0:2-events]
root        1920  0.0  0.0      0     0 ?        I    06:20   0:00 [kworker/0:3]
root        1929  0.0  0.0      0     0 ?        I    06:20   0:00 [kworker/u4:1]
root        1930  0.0  0.0  14744  7800 ?        Ss   06:21   0:00 sshd: cmata [priv]
root        1933  0.0  0.0      0     0 ?        S    06:21   0:00 [psimon]
cmata       1935  0.3  0.1  20376 11392 ?        Ss   06:21   0:00 /usr/lib/systemd/sys
cmata       1936  0.0  0.0  21140  3512 ?        S    06:21   0:00 (sd-pam)
cmata       2001  0.0  0.0  15000  7076 ?        S    06:21   0:00 sshd: cmata@pts/0
cmata       2002  0.0  0.0   9060  5248 pts/0    Ss+  06:21   0:00 -bash
root        2011  0.0  0.0  14744  7800 ?        Ss   06:21   0:00 sshd: cmata [priv]
cmata       2058  0.0  0.0  15000  6948 ?        S    06:21   0:00 sshd: cmata@pts/1
cmata       2059  0.0  0.0   9060  5248 pts/1    Ss   06:21   0:00 -bash
cmata       2068  0.0  0.0  12620  5120 pts/1    R+   06:21   0:00 ps -aux
cmata@vm-ubuntu:~$
```

2.3. Investigue cada uno de los datos de los procesos del punto anterior (PID, VSZ ...).

Observe el ultimo proceso es;

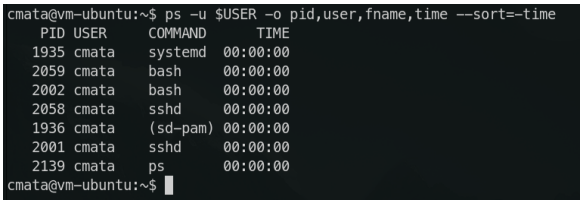
```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
cmata       2068  0.0  0.0  12620  5120 pts/1    R+   06:21   0:00 ps -aux
```

Donde:

- USER: Usuario que ejecuta el proceso, en este caso cmata.
- PID: Identificador de proceso, aquí es 2068.
- %CPU: Porcentaje del CPU utilizado por el proceso, 0.0 indica que no está usando prácticamente CPU en el momento de la captura.
- %MEM: Porcentaje de la memoria RAM utilizada por el proceso, también 0.0 aquí.
- VSZ: Tamaño virtual del proceso en KB.
- RSS: Tamaño residente (memoria física usada) en KB.
- TTY: Terminal asociada al proceso, pts/1 indica una pseudo-terminal slave.
- STAT: Estado del proceso (R+: en ejecución en una terminal).
- START: Hora de inicio del proceso.
- TIME: Tiempo total de CPU usado desde el inicio.

2.4. Busque el comando que retorna los procesos propios de un usuario y tome la captura de pantalla del que posee mayor tiempo en el procesador.

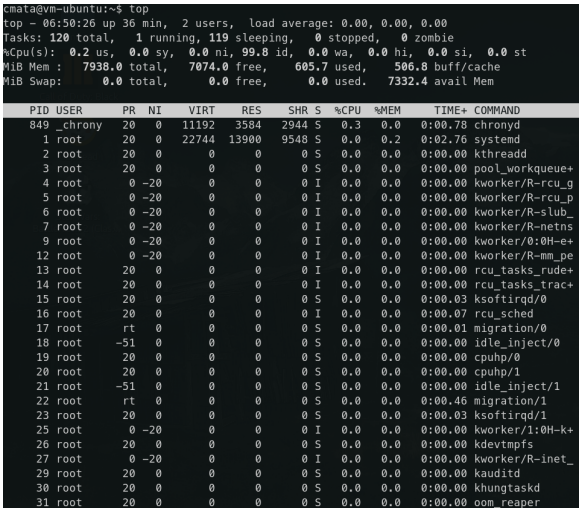
```
ps -u $USER -o pid,user,fname,time --sort=-time
```



2.5. Ejecute el comando: top en la primera consola. ¿Para que sirve?

```
top
```

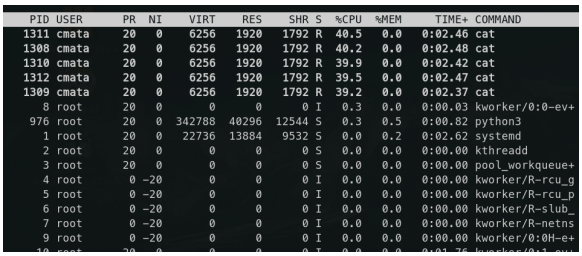
El programa top proporciona una vista dinámica y en tiempo real del funcionamiento de un sistema. Permite visualizar un resumen del sistema, además de una lista de los procesos o hilos que están siendo manejados por el kernel de Linux en ese momento [2].



2.6. Ejecute el comando (5 veces en la segunda consola)

```
cat /dev/zero > /dev/null &
```

2.7. Ejecute el comando top nuevamente en la primera consola. ¿Qué observa con respecto al top anterior?



Cuando se ejecutó el comando cat /dev/zero > /dev/null & cinco veces en una segunda consola y después revisé el monitor de procesos con top en la primera consola, se observa que estos comandos estaban utilizando una cantidad considerable de CPU, cada uno con un uso aproximado entre 39.9 % y 40.5 %. Estos procesos aparecen en estado R", que significa que están activos y corriendo. A pesar de que cada proceso consume una pequeña cantidad de memoria, el impacto en el uso total de la memoria es mínimo.

El comando en sí está redirigiendo un flujo constante de ceros, generado por /dev/zero, hacia /dev/null, que es esencialmente como enviar datos a un agujero negro donde se descartan. Este proceso se traduce en una carga de trabajo continua para el CPU porque siempre tiene datos para procesar, lo que explica el alto porcentaje de uso del CPU observado para estos procesos.

2.8. ¿Qué significa los valores de cada uno de los parámetros de los procesos creados (PR, NI, VIRT ...)?

- PR (Priority): Muestra la prioridad del kernel para el proceso. Un número más bajo significa mayor prioridad.
- NI (Nice value): Indica el valor de "nice"del proceso, que es un ajuste manual para la prioridad del proceso. Un valor más alto en este campo da a entender que el proceso es menos prioritario, permitiendo que otros procesos más urgentes reciban más tiempo de CPU.
- VIRT (Virtual Memory Size): Refleja la cantidad total de memoria virtual utilizada por el proceso.
- RES (Resident Set Size): Representa la porción no intercambiada de la memoria que el proceso ha ocupado en la RAM física.
- SHR (Shared Memory): Muestra la cantidad de memoria compartida utilizada por el proceso.
- S (Status): Indica el estado actual del proceso: 'R' para ejecutándose, 'S' para dormido, 'T' para detenido por rastreo de software, 'Z' para zombie, entre otros.
- %CPU: Muestra el porcentaje del tiempo de CPU utilizado por el proceso desde la última actualización.
- %MEM: Refleja el porcentaje de la memoria física total utilizada por el proceso.

- TIME+: Indica el tiempo total de CPU que el proceso ha utilizado desde que se inició.
- COMMAND: Muestra el comando que inició el proceso.

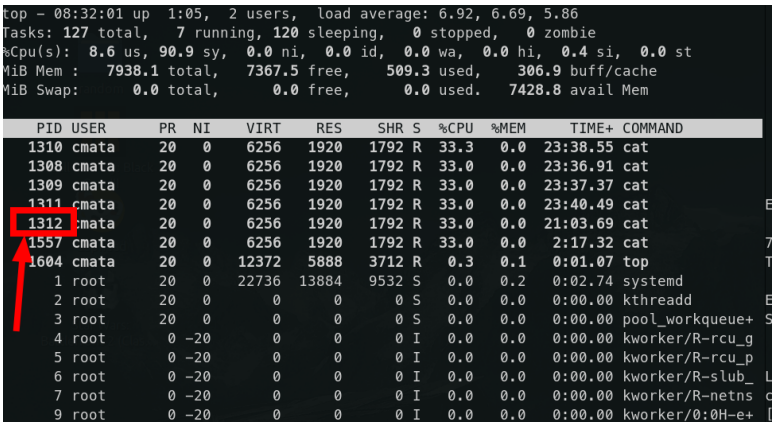
2.9. Note que todos los procesos creados tienen una prioridad similar ¿Por qué sucede esto?

Se observa que todos los procesos creados poseen una prioridad similar porque han sido iniciados bajo las mismas condiciones y configuraciones de sistema. Al ejecutar múltiples instancias del comando cat /dev/zero > /dev/null &, cada proceso se lanza con el mismo valor de prioridad y nice por defecto que el sistema asigna a este tipo de comando.

2.10. ¿Por qué el parámetro "Time" aumenta paulatinamente?

El parámetro "Time.[en] el monitor de procesos top muestra el tiempo total de CPU consumido por el proceso desde su inicio. Este valor aumenta paulatinamente porque mide la cantidad de tiempo de procesador que el proceso ha utilizado.

2.11. Obtenga un identificador de alguno de los procesos creados anteriormente.



2.12. Aumente la prioridad de dicho proceso con el comando

renice –n 10 PID.

```
cmata@vm-ubuntu:~$ renice -n 10 1312
1312 (process ID) old priority 0, new priority 10
```

```
top - 08:33:13 up  1:06,  2 users,  load average: 6.26, 6.54, 5.88
Tasks: 127 total,   7 running, 120 sleeping,   0 stopped,   0 zombie
%Cpu(s):  9.0 us, 90.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
MiB Mem :   7938.1 total,   7367.5 free,    509.2 used,    306.9 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   7428.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1308 cmata     20   0    6256   1920   1792 R  34.9   0.0  24:01.08 cat
   1309 cmata     20   0    6256   1920   1792 R  34.9   0.0  24:01.41 cat
   1310 cmata     20   0    6256   1920   1792 R  34.6   0.0  24:02.64 cat
   1311 cmata     20   0    6256   1920   1792 R  34.6   0.0  24:04.54 cat
   1557 cmata     20   0    6256   1920   1792 R  33.9   0.0   2:41.44 cat
   1312 cmata     30  10    6256   1920   1792 R  26.6   0.0  21:27.48 cat
   1604 cmata     20   0   12372   5888   3712 R   0.3   0.1   0:01.25 top
      1 root      20   0   22736  13884   9532 S   0.0   0.2   0:02.74 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workque
```

2.13. Inicie un proceso con prioridad alta con el siguiente comando: nice -n -10 cat /dev/zero >/dev/null &

```
cmata@vm-ubuntu:~$ sudo nice -n -10 cat /dev/zero > /dev/null &
[7] 1563
```

2.14. Ejecute el comando top nuevamente.

```
top - 08:24:25 up 57 min,  2 users,  load average: 6.56, 5.59, 5.12
Tasks: 130 total,   8 running, 122 sleeping,   0 stopped,   0 zombie
%Cpu(s):  8.9 us, 90.3 sy,  0.4 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
MiB Mem :   7938.1 total,   7369.5 free,    507.5 used,    306.7 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   7430.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1565 root     -10 -10    6256   1920   1792 R  99.7   0.0   1:02.27 cat
   1310 cmata     20   0    6256   1920   1792 R  19.9   0.0  22:09.83 cat
   1311 cmata     20   0    6256   1920   1792 R  19.9   0.0  22:11.73 cat
   1308 cmata     20   0    6256   1920   1792 R  19.5   0.0  22:08.05 cat
   1309 cmata     20   0    6256   1920   1792 R  19.5   0.0  22:08.56 cat
   1557 cmata     20   0    6256   1920   1792 R  19.5   0.0   0:48.33 cat
   1312 cmata     30  10    6256   1920   1792 R   2.0   0.0  20:48.40 cat
   1604 cmata     20   0   12372   5888   3712 R   0.3   0.1   0:00.02 top
      1 root      20   0   22736  13884   9532 S   0.0   0.2   0:02.72 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workqueue+
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_g
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_p
```

3. Creación de procesos con parámetros establecidos por el usuario

3.1. Para los siguientes programas pruebe su funcionamiento con valores que tomen algún tiempo considerable en terminar su ejecución, por ejemplo, valores mayores a 100

3.2. Realice un programa recursivo en C y Python que sea capaz calcular el factorial de cualquier número entero.

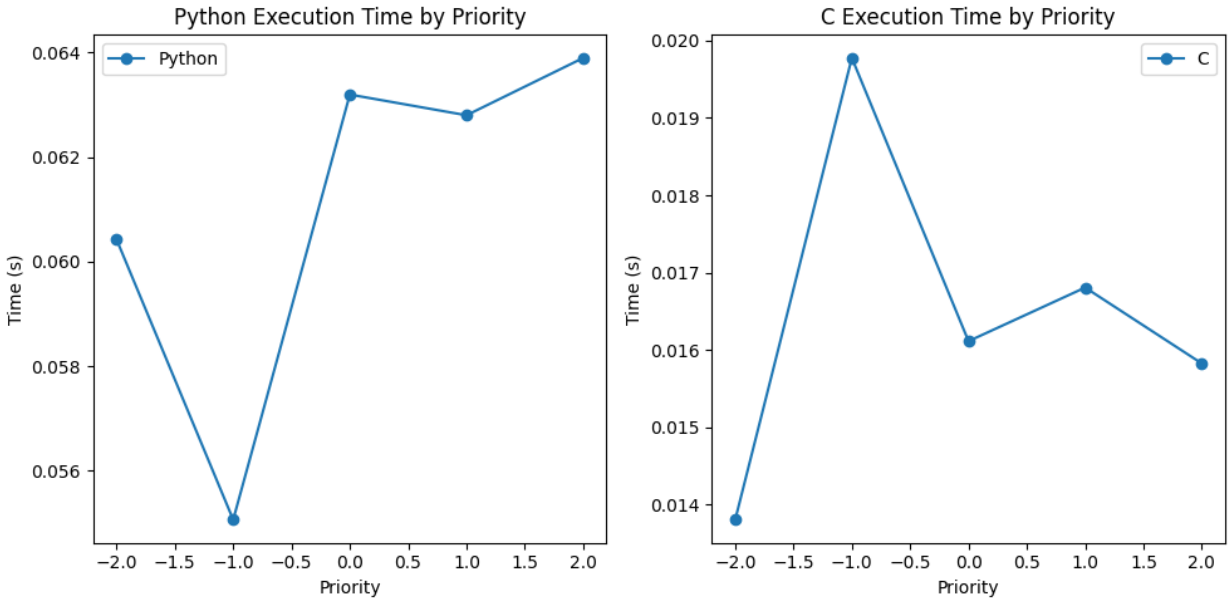
```
❯ bat ./factorial.c -pp
#include <gmp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void factorial(unsigned int n, mpz_t result) {
  if (n == 0 || n == 1) {
    mpz_set_ui(result, 1);  // Factorial of 0 or 1 is 1
  } else {
    mpz_t temp_result;
    mpz_init(temp_result);
    factorial(n - 1, temp_result);       // Recursive call for n-1
    mpz_mul_ui(result, temp_result, n);  // result = temp_result * n
    mpz_clear(temp_result);
  }
}

int main(int argc, char *argv[]) {
  int print_result = 0;  // Flag to determine whether to print the result

  // Check if the correct number of arguments is provided and if '-p' is one of
  // them
  if (argc < 2) {
    printf("Usage: %s <number> [-p]\n", argv[0]);
    return 1;
  }

  // Check for the '-p' option in the arguments
  for (int i = 2; i < argc; i++) {
    if (strcmp(argv[i], "-p") == 0) {
      print_result = 1;  // Set the print flag if '-p' is found
      break;
    }
  }

  unsigned int number = atoi(argv[1]);

  mpz_t result;
  mpz_init(result);
  factorial(number, result);  // Calculate factorial

  if (print_result) {
    printf("The factorial of %u is:\n", number);
    mpz_out_str(stdout, 10, result);  // Print result in base 10
    printf("\n");
  }

  mpz_clear(result);
  return 0;
}
```

```
> bat ./factorial.py -pp
import sys
import argparse

# Set the recursion limit
sys.setrecursionlimit(200000)
sys.set_int_max_str_digits(50000)

def factorial(n):
    """
    Recursively calculates the factorial of a given number.

    Args:
        n (int): The number to calculate the factorial for.

    Returns:
        int: The factorial of the given number.
    """
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
            description='Calculate the factorial of a given number.')
    parser.add_argument('number', type=int,
                        help='The number to calculate the factorial for.')
    parser.add_argument('-p', '--print', action='store_true',
                        help='Print the factorial result to the console.')
    args = parser.parse_args()

    result = factorial(args.number)
    if args.print:
        print(f"The factorial of {args.number} is:")
        print(result)%
```

3.3. **Ejecute dichos programas en su máquina virtual y tome el tiempo de ejecución, así como los parámetros de ambos procesos con el comando top (En caso de que sea necesario coloque prints entre cada recursión).**

3.4. **Ejecute el programa realizado en Python 5 veces con 5 diferentes prioridades de manera ascendente y muestre una gráfica del comportamiento del mismo (Prioridad-Tiempo).**

3.5. **Ejecute el programa realizado en C 5 veces con 5 diferentes prioridades de manera descendente y muestre una gráfica del comportamiento de mismo (Prioridad-Tiempo).**



3.6. Investigue el comando para eliminar un proceso. Posteriormente ejecute el programa en C y elimínelo antes de que termine su ejecución.

En sistemas operativos GNU/Linux, se utiliza el comando `kill` para eliminar procesos en ejecución. Este comando requiere el número de identificación del proceso (PID) que se desea terminar. Si el proceso no responde a la señal de terminación estándar, se puede forzar su cierre utilizando la señal `-9`, como en:

```
kill -9 PID
```

Para eliminar un programa en C antes de que termine su ejecución, primero se compila y ejecuta el programa en segundo plano, obteniendo su PID. Luego, se utiliza el comando `kill` para enviar la señal de terminación, finalizando el proceso antes de que complete su tarea.

```
gcc -o programa programa.c  # Compilar el programa en C
./programa &                # Ejecutar el programa en segundo plano
ps aux | grep rograma       # Obtener el PID del programa
kill -9 PID                 # Eliminar el programa utilizando su PID
```

3.7. Discuta el comportamiento de la gráfica. ¿Es el comportamiento que esperaba?

Las gráficas que presentas muestran que los tiempos de ejecución para los programas en Python y C varían con cambios en la prioridad del proceso, pero no muestran una tendencia consistente que relacione directamente mayor prioridad con menor tiempo de ejecución. Este comportamiento puede parecer inesperado, pero en realidad refleja la naturaleza compleja de la gestión de procesos en sistemas operativos modernos.

La prioridad de un proceso no garantiza necesariamente un mayor tiempo de CPU, especialmente si el proceso no está compitiendo activamente por recursos o si está limitado por otros factores como operaciones de entrada/salida. Además, los sistemas operativos están diseñados para equilibrar la carga y optimizar la respuesta general del sistema, lo que puede minimizar el impacto visible de cambios en la prioridad.

Por esta razón los resultados ilustran que la prioridad por sí sola no es una solución mágica para mejorar el rendimiento y debe ser considerada junto con otros factores del sistema.

4. Hilos en Linux

4.1. Realice un programa en C que tome un archivo de texto (.txt) y cuente la cantidad de apariciones de una determinada palabra. Tome el tiempo de ejecución del mismo.

4.2. Implemente dicho programa con 2,3,4,5 hilos y grafique el comportamiento (Cantidad de hilos- Tiempo) puede utilizar un software como excel o su equivalente.

4.3. ¿La mejora es lineal? Justifique dicha monotonía

Referencias

[1] man7.org, "ps(1) - Linux man page," 2024. Accessed: 2024-08-28.

[2] man7.org, "top(1) - Linux man page," 2024. Accessed: 2024-08-28.

[3] K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*. Online Publisher, 2012.

Submitted by Ignacio Grané Rojas - Carlos Andrés Mata Calderón on 3 de septiembre de 2024.