

Tarea de Programación Laboratorio 1: Desarrollo del programa `psinfo`

1. Introducción

En una empresa de desarrollo de software de sistema se está realizando un proyecto de personalización de un sistema operativo Linux Ubuntu. Los diseñadores del sistema piensan incluir nuevos comandos en el intérprete de línea de comandos incluido en la distribución. A usted se le ha asignado la tarea de programar un nuevo comando el cual permite obtener información importante acerca de un determinado **proceso** del sistema operativo. El comando se llamará **psinfo**.

2. Contextualización: Información de procesos en Linux

2.1. Sobre los procesos

Existe un dicho dado por los genio de Free Electrons y es “Todo en Unix es un archivo y todo lo que no es archivo en Unix es un proceso”¹. Recordemos que un proceso es una instancia de un programa en ejecución la cual tiene una serie de datos asociados como: Archivos abiertos, memoria asignada, identificador de proceso, padre, prioridad, estado, entre otros. Tanto en Ubuntu como en Linux se puede obtener algo de información relacionada con los procesos en ejecución al ejecutar el administrador de tareas. Para llamarlo en Ubuntu, en el menú se puede digitar **System Monitor** tal y como se muestra en la figura 1.

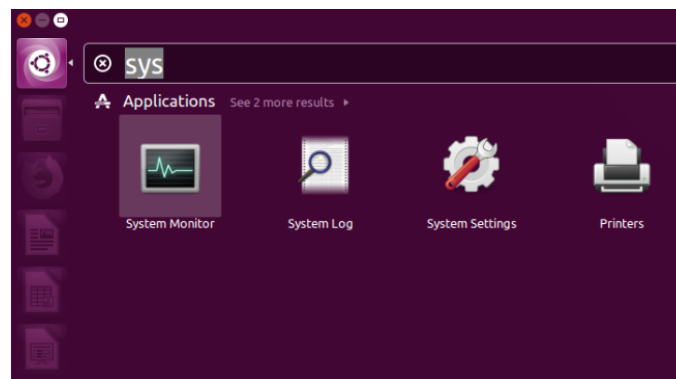
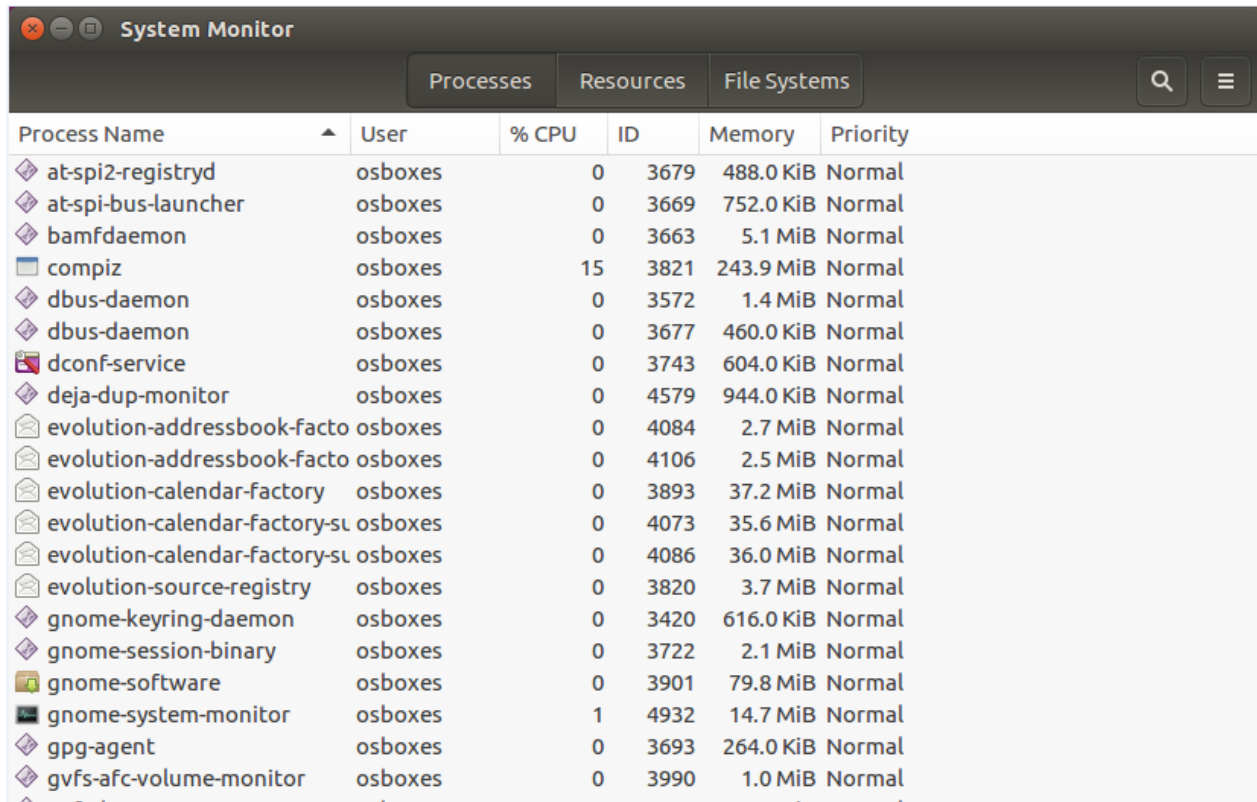


Figura 1. Inicio del Monitor del Sistema (System Monitor)

¹ Ver diapositiva 65 del documento [The Unix and GNU/Linux command line](#)

Una vez que se abre el monitor del sistema, se pueden ver algunos de los procesos en ejecución e información relacionada (Figura 2), por ejemplo su identificador de proceso (PID).



Process Name	User	% CPU	ID	Memory	Priority
at-spi2-registrd	osboxes	0	3679	488.0 KiB	Normal
at-spi-bus-launcher	osboxes	0	3669	752.0 KiB	Normal
bamfd daemon	osboxes	0	3663	5.1 MiB	Normal
compiz	osboxes	15	3821	243.9 MiB	Normal
dbus-daemon	osboxes	0	3572	1.4 MiB	Normal
dbus-daemon	osboxes	0	3677	460.0 KiB	Normal
dconf-service	osboxes	0	3743	604.0 KiB	Normal
deja-dup-monitor	osboxes	0	4579	944.0 KiB	Normal
evolution-addressbook-facto	osboxes	0	4084	2.7 MiB	Normal
evolution-addressbook-facto	osboxes	0	4106	2.5 MiB	Normal
evolution-calendar-factory	osboxes	0	3893	37.2 MiB	Normal
evolution-calendar-factory-sl	osboxes	0	4073	35.6 MiB	Normal
evolution-calendar-factory-sl	osboxes	0	4086	36.0 MiB	Normal
evolution-source-registry	osboxes	0	3820	3.7 MiB	Normal
gnome-keyring-daemon	osboxes	0	3420	616.0 KiB	Normal
gnome-session-binary	osboxes	0	3722	2.1 MiB	Normal
gnome-software	osboxes	0	3901	79.8 MiB	Normal
gnome-system-monitor	osboxes	1	4932	14.7 MiB	Normal
gpg-agent	osboxes	0	3693	264.0 KiB	Normal
gvfs-afc-volume-monitor	osboxes	0	3990	1.0 MiB	Normal

Figura 2. System Monitor

En la siguiente sección se verán algunos comandos con el fin de analizar cómo se obtiene esta misma información desde consola.

2.2. Procesos y consola

Desde la consola de linux es posible obtener información relacionada con los procesos. A continuación se presenta una lista de los comandos más populares utilizado para obtener información de procesos. Si quiere conocer más a fondo el uso de los comandos para obtener información de procesos se sugiere el texto guía del curso (www.google.com) y las referencias al final del documento. En la siguiente tabla se da una breve descripción del comando ps:

- **Comando ps**

```
ps <opciones>
```

Lista los procesos que se están ejecutando en el sistema, si no se agrega ningún parámetro opcional ps mostrará los procesos del usuario que se encuentra logueado. Las opciones

adicionales de este comando se pueden consultar en el manual o en las referencias al final del documento. Por ejemplo el comando:

```
ps ux
```

Lista todos los procesos correspondientes al usuario actual. Mientras el comando:

```
ps aux
```

Lista todos los procesos que se están ejecutando en el sistema.

- **Comando top**

```
top
```

Despliega los procesos más importantes ordenados de acuerdo al uso de porcentaje de CPU. Debido a que en este comando la información se despliega (más o menos) en tiempo real, la consola se bloquea de modo que para salir de este se presiona la tecla **q**.

- **Comando jobs**

```
jobs
```

Retorna la lista de procesos en background en el mismo shell. Lanzar procesos en background es de utilidad si no se quiere bloquear la consola. Para hacer que un proceso se ejecute en background, se debe agregar **&** al final de la línea. Así:

```
comando &
```

Por ejemplo, si hubiésemos deseado correr gedit en background, el comando sería:

```
gedit &
```

2.3. Practicando los comandos anteriores

A continuación se muestra paso a paso la ejecución de los comandos anteriormente mencionados en una máquina determinada con el fin de facilitar su comprensión. Pese a que se muestran capturas y parte de las salidas se recomienda que los ejecute en su máquina y compare los resultados, su comprensión puede ser de utilidad para el desarrollo de la práctica.

1. Abra una terminal de linux y ejecute el comando ps. ¿Qué observa?

```
$ ps
  PID TTY          TIME CMD
 5151 pts/4    00:00:00 bash
 5204 pts/4    00:00:00 ps
```

2. Ejecute el comando jobs. ¿Qué observa y por qué?
3. Ahora llame el editor de texto de su preferencia (en este ejemplo se llamó gedit) en background. Si usted observa la salida verá un número ¿Con qué se relaciona este?

```
$ gedit &
[1] 5205
```

4. Ejecute nuevamente el comando jobs. ¿Cambia la salida?

```
$ jobs
[1]+  Running                  gedit &
```

5. Ahora ejecute el comando ps con la opción l

```
$ ps l
 F  UID    PID  PPID  PRI  NI   VSZ   RSS WCHAN    STAT TTY          TIME COMMAND
 0  1000    5151   5144   20    0  22636  5240 wait     Ss   pts/4        0:00 bash
 0  1000    5205   5151   20    0 655856 36848 poll_s  Sl   pts/4        0:07 gedit
 0  1000    6339   5151   20    0  28916  1620 -        R+   pts/4        0:00 ps l
```

A continuación se describen algunos de los ítems (los cuales se muestran en la salida anterior) asociados con el proceso :

- **PID:** Process id
- **VSZ:** Virtual process size (code + data + stack)
- **RSS:** Process resident
- **size:** number of KB currently in RAM
- **TTY:** Terminal STAT:
- **Status:** R (Runnable), S (Sleep), W (paging), Z (Zombie)...

¿Que significado tiene dentro de lo que se ha visto en la teoría cada una de estas salidas? Llene una tabla con la información con estos datos y que le fue arrojada en su caso particular.

6. Desde el monitor del sistema observe la información asociada al proceso del gedit (ver Figura 3). Compare los resultados con la salida del comando ejecutado en el paso 5

gedit (PID 5205)	
Process Name	gedit
User	osboxes (1000)
Status	Sleeping
Memory	7.6 MiB
Virtual Memory	640.5 MiB
Resident Memory	36.0 MiB
Writable Memory	N/A
Shared Memory	28.4 MiB
X Server Memory	12.8 KiB
CPU	0%
CPU Time	0:02.60
Started	Today 10:27 AM
Nice	0
Priority	Normal
ID	5205
Security Context	unconfined
Command Line	gedit
Waiting Channel	poll_schedule_timeout
Control Group	/user.slice/user-1000.slice (pids), /user.slice (blkio/devices/me

Figura 3. Información del proceso gedit

7. Cierre el programa gedit y ejecute nuevamente el comando ps. ¿Por qué la salida?

```
$ ps
  PID TTY          TIME CMD
 5151 pts/4    00:00:00 bash
 6559 pts/4    00:00:00 ps
[1]+  Killed                  gedit
```

2.4. Sobre el sistema de archivos procfs:

El sistema de archivos `/proc` se utiliza para ofrecer información relacionada con el sistema. [En este enlace](#) pueden consultar la información detallada contenida en esta carpeta². Con el fin de facilitar la implementación de lo que se pide en el laboratorio puede ser de utilidad usar algunos comandos de consola, para ello trate de seguir las siguientes actividades orientado por el tutor:

Actividad 1:

Suponga que usted tiene una consola abierta y ejecute el comando `ps`. Luego ejecute el comando `gedit` para llamar el editor de textos:

```
$ gedit &
[1] 4858
```

² También puede consultar el manual de proc usando el comando `man proc` en la terminal

Actividad 2:

Ahora explore el contenido del directorio `/proc`, esto lo puede hacer con el comando `ls`, tal y como se muestra a continuación en pantalla. Consulte y describa brevemente qué información aparece allí.

```
$ ls /proc
1      173    2493  2740  2962  3283  84          cmdline      misc
10     176    25     2741  299   33 85    consoles   modules
102    18     250    2749  2991  333  856      cpuinfo      mounts
1073   183    2507   2755  2993  3334  86          ...
```

Preguntas:

1. ¿Qué significan todos esos números enteros? (Consulte el enlace de arriba).
2. ¿Estos números son archivos o directorios?
3. El número entero 1 ¿con qué está asociado?

Actividad 3:

Ahora explore el contenido del número asociado al `gedit`.

```
$ ls /proc/4858
attr      cwd      map_files  oom_adj      sessionid  timers
autogroup environ  maps       oom_score    setgroups  timerslack_ns
auxv      exe      mem        oom_score_adj smaps      uid_map
cgroup    fd        mountinfo  pagemap      stack      wchan
clear_refs fdinfo    mounts     personality   stat
cmdline   gid_map  mountstats projid_map    statm
comm      io       net        root          status
coredump_filter limits   ns          sched         syscall
cpuset    loginuid numa_maps  schedstat     task
```

Preguntas:

1. ¿Qué información se obtiene de los archivos `stat` y `status` anteriormente resaltados?
2. ¿La información en dichos archivos es la misma?, ¿De cual archivo es mejor leerla y por que?
3. Teniendo en cuenta lo anterior llene la siguiente tabla para el proceso asociado al editor de textos. Llene la siguiente tabla para su caso particular.

Información	Valor
Name	
State	
FDSize	

VmPeak	
VmSize	
VmLck	
voluntary_ctxt_switches	
nonvoluntary_ctxt_switches	

3. Descripción del programa a desarrollar **psinfo**

Para el desarrollo de este programa se ha definido una metodología por etapas donde cada etapa debe proporcionar un producto funcional completo. **psinfo** es una aplicación hecha en C cuyo objetivo es implementar de manera más simple comandos como los que se ejecutaron anteriormente mediante la lectura de archivos del sistema, en este caso los contenidos en el directorio `/proc`.

Se inicia con el desarrollo de una implementación muy simple de la función aumentando su funcionalidad etapa por etapa.

3.1. Etapa 1

Se requiere implementar el programa básico **psinfo**, el cual recibe como argumento de entrada un identificador de proceso (pid) y retorna en pantalla una información de estado del proceso. La información de estado solicitada es la siguiente:

1. Nombre del proceso (Name).
2. Estado del proceso (State).
3. Tamaño total de la imagen de memoria.
 - a. Tamaño de la sección de memoria TEXT (VmExe).
 - b. Tamaño de la sección de memoria DATA (VmData).
 - c. Tamaño de la sección de memoria STACK (VmStk).
4. Número de cambios de contexto realizados:
 - a. Voluntarios (voluntary_ctxt_switches).
 - b. No voluntarios (nonvoluntary_ctxt_switches).

La información debe tomarse del pseudo sistema de archivos de procesos del sistema operativo linux *procfs* (ubicado en la ruta `/proc`). [En este enlace](#) pueden consultar la información detallada contenida en esta carpeta. Se sugiere revisar especialmente el archivo **`/proc/[pid]/status`**, pues es el que contiene la información solicitada.

Ejemplo de uso:

```
$ ./psinfo 10898
```

```
Nombre del proceso: gedit
Estado: S (sleeping)
Tamaño total de la imagen de memoria: 715000 KB
    Tamaño de la memoria en la región TEXT: 10000KB
    Tamaño de la memoria en la región DATA: 24200KB
    Tamaño de la memoria en la región STACK: 21000KB
Número de cambios de contexto realizados (voluntarios - no voluntarios): 17536 - 189
```

3.2. Etapa 2

Implementar en el comando la opción **lista** (-l) en la cual el usuario pueda pasar una lista de identificadores de procesos. Para cada uno de los procesos en la lista, el programa debe obtener la información de estado, almacenarla en una estructura de memoria temporal (usando estructuras de C y memoria dinámica) y reportar los valores almacenados en pantalla.

Ejemplo de uso:

```
$ psinfo -l 10898 1342 2341
-- Información recolectada!!!
Pid: 10898
Nombre del proceso: gedit
...
Pid: 1342
Nombre del proceso: chrome
...
Pid: 2341
Nombre del proceso: bash
...
```

3.3. Etapa 3

Implementar una opción de reporte (-r) en la cual el proceso realice la funcionalidad solicitada en la etapa 2, pero generando un archivo de salida con la información de estado de los procesos solicitados. El archivo de salida debe llamarse `psinfo-report-[pids].info`.

Ejemplo de uso:

```
$ psinfo -r 10898 1342
Archivo de salida generado: psinfo-report-10898-1342.info
```

3.4. Etapa 4

Implemente las validaciones necesarias para identificar un uso incorrecto del programa. Por ejemplo, detectando parámetros no válidos o solicitud de información de procesos inexistentes. Si el

programa detecta un mal uso debe imprimir un mensaje informando acerca del error detectado, luego debe imprimir un mensaje mostrando un resumen de los parámetros válidos y debe finalizar el programa reportando un código de ejecución errónea al *bash*. El código de error puede ser definido por usted mismo y debe ser diferente al valor correspondiente a la ejecución normal.

4. Consideraciones generales

- Desarrolle el programa por etapas conservando siempre la última versión funcional de su implementación (se sugiere hacer uso de repositorios para este fin: git o svn).
- Realice su implementación teniendo en cuenta todas las buenas prácticas de programación explicadas en clase de laboratorio.
- Realice una implementación orientada a funciones. Tenga en cuenta que el código que usted desarrolle debe ser funcional y, en lo posible, eficiente.
- Documente bien el código. Siga las buenas prácticas de programación que ha ido adquiriendo a lo largo de la carrera.
- El programa deberá ser multiarchivo.
- Ante cualquier duda consulte con el equipo de profesores del laboratorio en los medios dispuestos para tal fin (foros piazza o moodle).

5. Enlaces

1. 20 comandos para administrar y gestionar facilmente los procesos en Linux (Accedido el 6 de marzo de 2018:
<https://openwebinars.net/blog/20-comandos-para-administrar-y-gestionar--facilmente-los-procesos-linux/>)
2. Obtener información básica sobre procesos del sistema Linux en C y C++ (parte 3) (Accedido el 6 de marzo de 2018:
<https://poesiabinaria.net/2017/07/obtener-informacion-basica-procesos-del-sistema-linux-c-c-parte-3/>)
3. Linux Sysadmin Basics -- 6.3 The /proc Filesystem (Accedido el 15 de septiembre de 2018:
<https://www.youtube.com/watch?v=0XdjODvsRN8>)
4. MAN PAGES (Accedido el 15 de septiembre de 2018:
<https://www.cs.mcgill.ca/~guide/help/man.html>)
5. Check actually memory usage of process (Accedido el 15 de septiembre de 2018:
https://github.com/anthonywei/ps_mem.py)