



**Universidad  
de Concepción**



**Facultad  
de Ingeniería**  
Universidad de Concepción

## **Proyecto 1: Sistemas Operativos** **Shell con implementación de daemon**

### **Integrantes:**

Oliver Alarcón Brito

Sofía Bravo Saavedra

Esteban Chandía Cifuentes

## Introducción

Una shell es una interfaz de usuario que facilita la comunicación entre el usuario y el núcleo del sistema. Funciona como una capa interactiva que permite a los usuarios interactuar con el sistema operativo emitiendo comandos. La shell interpreta estos comandos y los traduce para que el sistema operativo los pueda ejecutar.

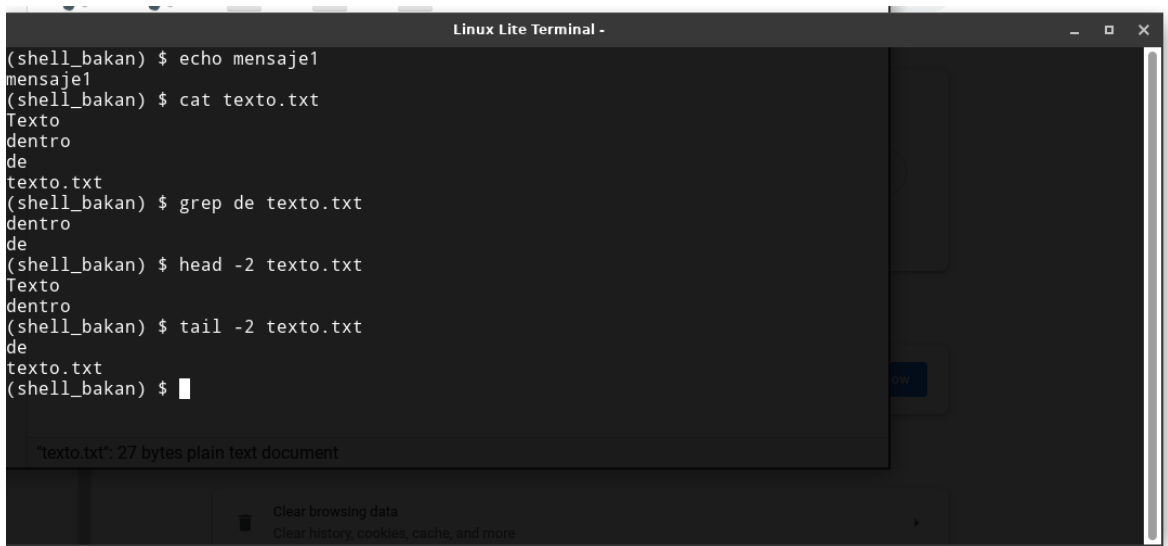
Un daemon es un tipo especial de programa o proceso informático que se ejecuta en segundo plano, sin intervención directa del usuario, y generalmente se inicia automáticamente cuando se inicia el sistema operativo. Los daemons son esenciales para la gestión y el funcionamiento de un sistema informático, ya que realizan diversas tareas en segundo plano, como proporcionar servicios, administrar recursos o realizar trabajos de mantenimiento.

Utilizando los conceptos anteriores, se ha desarrollado una shell simple que puede ejecutar algunos comandos comunes, y que también permite la creación de un daemon que mide y registra una línea con la información requerida del sistema en el log del sistema, todo con el objetivo de introducirse en el manejo de procesos concurrentes en unix, creación, ejecución y terminación usando llamadas a sistemas `fork()`, `exec()` y `wait()`. Además el uso de otras llamadas a sistema como `signals` y comunicación entre procesos usando `pipes`.

## Desarrollo

Shell:

Para compilar la shell se debe ejecutar la línea: `gcc *.c -o shell_bakan` en la carpeta con todos los archivos. Después se ejecuta: `./shell_bakan`, que inicia la shell, esperando que le ingresen los comandos o pipes. Cuando la shell se encuentra en este estado, si se hace *enter* sin ingresar ningún comando, el prompt vuelve a aparecer. Los comandos que soporta esta shell son: `ls`, `grep`, `echo`, `cat`, `head`, `tail`, etc (comandos básicos de una shell según lo requerido).



Al momento de implementar la shell hay dos situaciones; la primera es cuando el input viene sin pipes. Para esto, se hacen tokens de la entrada (`tokenize_input`) para diferenciar entre el

comando y sus argumentos, luego se ejecuta el comando en un proceso hijo (que se hace con fork) y con execvp (execmd). La segunda es cuando el input viene con pipes, para esto se hace parecido. Primero se separa el input con strtok para dividir los comandos separados por “|” y después se pasa a la función (execute\_pipeline) que los ejecuta como pipeline. Esta función hace que, si es el primer proceso, recibe entrada estándar y manda su salida al siguiente proceso, si es un proceso intermedio, entonces recibe el input del anterior y manda su output al siguiente y ,si es el final, entonces recibe el input de otro proceso y el output es estándar..

```
Linux Lite Terminal -
linux ~ Documents ./.a.out
(shell_bakan) $ ps aux | sort -nr -k 4 | head -10
linux      7139  0.5  9.6 1185846148 193804 ?        Ssl   17:24   0:18 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=31
49 --enable-crash-reporter=, --extension-process --change-stack-guard-on-fork=enable --disable-gpu-compositing --lang=en-US --num-rast
er-threads=1 --renderer-client-id=46 --time-ticks-at-unix-epoch=-1696626783784840 --launch-time-ticks=4207588652 --shared-files=v8_con
text_snapshot_data:100 --field-trial-handle=0,i,12573278723667355661,1653861854779327953,262144
linux      3142  1.9  7.1 34239716 144116 ?        Ssl   16:19   2:24 /opt/google/chrome/chrome
root      2429  0.9  4.8 396452 97560 tty7      Ssl+  16:15   1:12 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/root
/:0 -nolisten tcp vt7 -novtswitch
linux      7283  0.2  3.4 1185808872 70360 ?        Ssl   17:30   0:08 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=31
49 --enable-crash-reporter=, --change-stack-guard-on-fork=enable --disable-gpu-compositing --lang=en-US --num-raster-threads=1 --rende
rer-client-id=49 --time-ticks-at-unix-epoch=-1696626783784840 --launch-time-ticks=4597303372 --shared-files=v8_context_snapshot_data:1
00 --field-trial-handle=0,i,12573278723667355661,1653861854779327953,262144
linux      3184  0.4  2.8 34009012 57016 ?        Ssl   16:19   0:31 /opt/google/chrome/chrome --type=gpu-process --crashpad-handler-pid
=3149 --enable-crash-reporter=, --change-stack-guard-on-fork=enable --gpu-preferences=WAAAAAAAAAAgAAEAAAAAAAAAAAAAAAAABgAAAAAAAA4AAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAgAAAAAAAAACAAAAAAAAAIAAAAAAAAAA== --shared-files --field-trial-
handle=0,i,12573278723667355661,1653861854779327953,262144
linux      7151  0.0  2.7 1185813424 56248 ?        Ssl   17:24   0:02 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=31
49 --enable-crash-reporter=, --change-stack-guard-on-fork=enable --disable-gpu-compositing --lang=en-US --num-raster-threads=1 --rende
rer-client-id=47 --time-ticks-at-unix-epoch=-1696626783784840 --launch-time-ticks=4218821504 --shared-files=v8_context_snapshot_data:1
00 --field-trial-handle=0,i,12573278723667355661,1653861854779327953,262144
linux      7292  0.0  2.5 1185815332 51244 ?        Ssl   17:30   0:01 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=31
49 --enable-crash-reporter=, --change-stack-guard-on-fork=enable --disable-gpu-compositing --lang=en-US --num-raster-threads=1 --rende
rer-client-id=48 --time-ticks-at-unix-epoch=-1696626783784840 --launch-time-ticks=4597689016 --shared-files=v8_context_snapshot_data:1
00 --field-trial-handle=0,i,12573278723667355661,1653861854779327953,262144
linux      7327  0.0  2.2 1185802984 44300 ?        Ssl   17:30   0:00 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=31
49 --enable-crash-reporter=, --change-stack-guard-on-fork=enable --disable-gpu-compositing --lang=en-US --num-raster-threads=1 --rende
rer-client-id=50 --time-ticks-at-unix-epoch=-1696626783784840 --launch-time-ticks=4610853357 --shared-files=v8_context_snapshot_data:1
00 --field-trial-handle=0,i,12573278723667355661,1653861854779327953,262144
linux      7303  0.0  2.1 1185802980 44000 ?        Ssl   17:30   0:00 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=31
49 --enable-crash-reporter=, --change-stack-guard-on-fork=enable --disable-gpu-compositing --lang=en-US --num-raster-threads=1 --rende
rer-client-id=51 --time-ticks-at-unix-epoch=-1696626783784840 --launch-time-ticks=4602960531 --shared-files=v8_context_snapshot_data:1
00 --field-trial-handle=0,i,12573278723667355661,1653861854779327953,262144
root      1932  0.0  2.0 445892 40396 ?        Ssl   16:15   0:04 /usr/libexec/packagekitd
(shell_bakan) $
```

**Daemon:**

Al momento de ejecutar el programa se puede ejecutar también el daemon, para esto se debe ejecutar: “./shell\_bakan daemon t p”, siendo ‘t’, el intervalo de tiempo cada cuanto se ejecuta el daemon, y ‘p’, el tiempo total en el que se ejecuta el daemon . Si no se pone nada además de daemon, se tomarán por defecto los valores de t y p como 5 y 60 respectivamente. El daemon, al ser un proceso que queda ejecutándose en segundo plano, se puede verificar en la shell escribiendo el comando: “grep diablito\_bakan /var/log/syslog”.

El daemon crea un proceso hijo utilizando fork y el proceso padre sale, dejando solo al proceso hijo en ejecución, de este modo queda ejecutándose en segundo plano. Luego, gracias a ‘umask’, se cambia de directorio de trabajo a la raíz del sistem y crea una nueva sesión utilizando ‘setsit()’. Luego, en syslog se abre el sistema de registro (openlog) y el daemon entra en un bucle durante ‘p’ segundos ejecutándose cada ‘t’ segundos, donde recopila y analiza la información relevante de ‘/proc/stat’ y la registra utilizando ‘syslog’. Este proceso dura hasta que haya transcurrido el tiempo total ‘p’ o se puede matar antes utilizando killall shell\_bakan.

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
linux ~ > Documents gcc *.c -o shell_bakan
linux ~ > Documents ./shell_bakan daemon 5 100
argc: 4
argv[0]: ./shell_bakan
argv[1]: daemon
argv[2]: 5
argv[3]: 100
linux ~ > Documents grep diablito_bakan /var/log/syslog
Oct 6 18:35:18 linux diablito_bakan[8215]: Processes: 8216
Oct 6 18:35:18 linux diablito_bakan[8215]: Procs Running: 4
Oct 6 18:35:18 linux diablito_bakan[8215]: Procs Blocked: 0
Oct 6 18:35:23 linux diablito_bakan[8215]: Processes: 8219
Oct 6 18:35:23 linux diablito_bakan[8215]: Procs Running: 1
Oct 6 18:35:23 linux diablito_bakan[8215]: Procs Blocked: 0
Oct 6 18:35:28 linux diablito_bakan[8215]: Processes: 8219
Oct 6 18:35:28 linux diablito_bakan[8215]: Procs Running: 1
Oct 6 18:35:28 linux diablito_bakan[8215]: Procs Blocked: 0
Oct 6 18:35:33 linux diablito_bakan[8215]: Processes: 8219
Oct 6 18:35:33 linux diablito_bakan[8215]: Procs Running: 1
Oct 6 18:35:33 linux diablito_bakan[8215]: Procs Blocked: 0
Oct 6 18:35:38 linux diablito_bakan[8215]: Processes: 8219
Oct 6 18:35:38 linux diablito_bakan[8215]: Procs Running: 1
Oct 6 18:35:38 linux diablito_bakan[8215]: Procs Blocked: 0
linux ~ > Documents ps aux | grep shell
linux 8215 0.0 0.0 2776 100 ? Ss 18:35 0:00 ./shell_bakan daemon
```

Conclusión

La exitosa implementación de la shell y el daemon representa un logro significativo en este proyecto. Hemos desarrollado una shell funcional que admite todos los comandos mencionados en este informe y, al mismo tiempo, hemos logrado crear un daemon capaz de medir y registrar la información requerida de manera efectiva.

Este proyecto ha proporcionado una valiosa oportunidad para profundizar en conceptos cruciales que hemos estudiado previamente en nuestras clases. A través de la aplicación de llamadas al sistema como fork(), exec(), wait(), así como el manejo de señales y la comunicación entre procesos mediante pipes, hemos adquirido un conocimiento sólido respecto al funcionamiento de, en este caso, sistemas operativos basados en Linux, que puede ser aplicable a otros sistemas operativos.