*Proyecto #1*

Demonios en el lenguaje de programación C

*Héctor Eduardo Arellano Negrin*

Escuela de Computación

UCV

Caracas, Venezuela

arellanonegrin2@hotmail.com

*Alberto Jose Suarez Bravo*

Escuela de Computación

UCV

Caracas, Venezuela

rakios25@hotmail.com

*Abstract*—En el presente paper se relatará sobre la experiencia que se tuvo, durante la realización del proyecto uno de la materia obligatoria Sistemas Operativos, sobre procesos inherentes al propio SO, los demonios, dichos procesos serán implementados en sistemas UNIX/LINUX, específicamente en Linux Debian, para verificar continuamente los archivos de la carpeta var/log del sistema; para ello se usarán los llamados Fork() y Exec() del sistema y se utilizarán Pipes para la comunicación entre los procesos. Aparte de hacer uso del makefile y shellscript para agilizar la compilación y ejecución del programa.

Keywords—Lenguaje C, demonio(daemon), sistemas operativos, makefile, shellscript, LINUX, procesos, Fork, Exec, Pipes.

# Introducción

Desde hace mucho tiempo el hombre ha buscado la manera de automatizar las inmensas tareas a diario, lo cual dio paso a muchos descubrimientos como la invención del teléfono, el automóvil, entre muchos otros. Pero se resalta más la invención del computador que sería una generalización de todos los inventos que requieren de un computo como tal, de este se desprenden muchas investigaciones y nuevos descubrimientos en este campo como: la programación, la invención de memorias para estas máquinas, descubrimiento de métodos para ir optimizando los sistemas de cómputo, y la necesidad de crear uno de los escaños mas importantes de la informática, el sistema operativo.

El sistema operativo es, lo que uno coloquialmente llamaría el administrador de la computadora, encargado de manejar los recursos de manera eficaz, segura y eficiente. Siendo intermediario entre dos roles: el usuario y el nucleo(kernel). El usuario es el nivel de abstracción donde se encuentran las aplicaciones para las personas, lenguaje de alto nivel, etc. Mientras que el núcleo sería la capa de abstracción para el hardware y procesos más internos al CPU, se habla de programación a muy bajo nivel, llegando a binario puro, etc.

Ahora bien el sistema operativo se encarga como mencionamos antes de toda la administración del uso de recursos entre partes (el usuario y el núcleo), para ello se requieren de diversas técnicas como: asignación de recursos, la sincronización de procesos e hilos, entre muchos temas relacionados, entonces lo que se viene a dar es una de las herramientas más poderosas conocidas como daemons o por su traducción al español demonios.

# Marco teórico

Antes que todo se debe tener en cuenta que todo el proyecto fue realizado en y para sistemas LINUX, especialmente para la distribución Debian, por ende, describiremos cada uno de los conceptos asociados tomando como referencia estos sistemas, la investigacion y explicacion de cada uno de ellos en este paper se realiza para la optima compresion de la solucion desarrollada para atacar el problema planteado.

Para empezar hablaremos de unos de los procesos bases cuando se habla de las funcionalidades del sistema operativo, y los cuales su comprensión es fundamental al momento de realizar este proyecto, los daemons. En sistemas LINUX se conoce como demonio o daemon (*Disk And Execution Monitor*) a un proceso que se ejecuta en segundo plano del sistema operativo, se ejecuta en todo momento y no posee interacción directa con el usuario, también se le conoce genéricamente como servicio o proceso, del cual no percibimos su ejecución. Un demonio realiza una operación especifica en tiempos predefinidos o en respuesta a ciertos eventos del sistema. El origen del nombre se refiere a su función de "vigilante" y no a una connotación negativa como se podría pensar en un principio.

Técnicamente, un proceso en un sistema LINUX es un demonio, cuando su proceso padre termina de ejecutarse y se le asigna como padre al proceso INIT, además no debe estar asociado a ninguna tty/terminal, sin embargo también se reconoce como demonio a cualquier proceso que corra en segundo plano sea hijo de INIT o no.

Ejemplos de demonio, son los servidores de las impresoras, servidores de correo, sistemas de conexión con redes, etc.

¿Cómo funcionan los mismos? La mayoría de los demonios son inicializados junto con el arranque del sistema, teniendo como padre al proceso INIT, normalmente estos procesos están en estado waiting hasta la ocurrencia de un evento en la que ejecutan su tarea, demonios conocidos son por ejemplo: Apache, httpd y Cron.

## El objetivo de este proyecto en resumen eran la implementación de un daemon que resolviera el problema planteado, pero aparte de este proceso en la solución se involucran muchos otros temas importantes a tomar en cuenta, estos influyen en gran *medida en el tipo de trato y planteamiento que se tuvo que tomar para la correcta implementacion de una solucion.*

*2.1) ¿Qué es un makefile? Make es una herramienta de gestión de dependencias, típicamente, las que existen entre los archivos que componen el código fuente de un programa, para dirigir su recompilación o "generación" automáticamente. Si bien es cierto que su función básica consiste en determinar automáticamente qué partes de un programa requieren ser recompiladas y ejecutar los comandos necesarios para hacerlo, también lo es que Make puede usarse en cualquier escenario en el que se requiera, de alguna forma, actualizar automáticamente un conjunto de archivos a partir de otro, cada vez que éste cambie. Make es muy usada en los sistemas operativos tipo Unix/Linux. Por defecto lee las instrucciones para generar el programa u otra acción del fichero makefile. Las instrucciones escritas en este fichero se llaman dependencias.*

*La herramienta make se usa para las labores de creación de fichero ejecutable o programa, para su instalación, la limpieza de los archivos temporales en la creación del fichero, todo ello especificando unos parámetros iniciales (que deben estar en el makefile) al ejecutarlo. Además de ser éste su objetivo principal, es utilizado para automatización de otras tareas como la creación de documentos del formato docbook, mantenimiento del sistema, simplemente usando o creando makefiles que hagan estas tareas.*

*.*

*2.2) ¿Qué es un syslog? syslog es una aplicación o biblioteca que envía los mensajes de registro. Un mensaje de registro suele tener información sobre la seguridad del sistema, aunque puede contener cualquier información. Junto con cada mensaje se incluye la fecha y hora del envío.*

*2.3) ¿Qué significa comprimir los datos? La compresión es un caso particular de la codificación, cuya característica principal es que el código resultante tiene menor tamaño que el original. Se define como la reducción del volumen de datos tratables para representar una determinada información empleando una menor cantidad de espacio.*

*2.4) ¿Que es un Shellscript? Un shell script es un fichero donde hemos introducido cualquier comando que se puede ejecutar en la shell. Se pueden incluir comandos UNIX, utilidades, emplear pipes, redireccionar entradas y salidas, definir y utilizar variables, además del uso de una serie de comandos UNIX creados para ser usados fundamentalmente en los shell scripts, como son condicionales y comandos para controlar el flujo del programa.*

*Normalmente, un daemon tiene asociado un shell script, que permite iniciar, parar y ver el estado del proceso [5], este shell script se encuentra en la carpeta /etc/ dentro de un subdirectorio que varía según la distribución del sistema operativo que se este usando, siempre que sea derivado del sistema UNIX.*

*Para convertir el proceso de la aplicación en un demonio se hizo un llamado a fork() del proceso initd, el encargado de invocar los shell scripts necesarios asociados a los procesos demonio del sistema operativo (Debian en este caso) para luego inhabilitar las descripciones básicas de interacción con el usuario ya previamente mencionadas (entrada/salida/error).*

*2.5) ¿Que es un Fork()? es un llamado al sistema que hace referencia a la creación de una copia de sí mismo por parte de un programa, este hace un duplicado del proceso que lo invocó, así este pasa a ser conocido como el proceso padre y el proceso generado es referido como proceso hijo. El hijo al ser un exacto duplicado del padre, puede ser diferenciado del mismo mediante el Process ID (PID), una identificación unica para cada proceso y se puede obtener después de realizar el llamado.*

*Ademas cuando se habla de Fork() es conviente tambien aclarar lo que es Exec(), este llamado reemplaza la imagen del proceso hijo creado por la llamada Fork() por una nueva, esto hace que dicho proceso hijo ejecute un lote de instrucciones que no están en el código del proceso padre que lo creó.*

*En este proyecto se tuvo la necesidad de indagar también sobre la comunicación entre procesos, esta se define como un conjunto de herramientas que ofrece el sistema operativo para que se comuniquen procesos que no comparten un mismo espacio de memoria. Como por ejemplo los pipes, los cuales seran el siguiente y ultimo punto que tuvimos que enfocarnos al realizar este proyecto.*

*2.5)¿Qué son los pipes? Los pipes o tuberias son mecanismos de comunicación entre procesos que permiten que dos o más procesos se envíen información entre ellos mismos. Son usados comúnmente por el shell para conectar la salida estándar de un utility o comando a la entrada estándar de otro.*

*Implementación*

*A la hora de hacer el demonio no se tenía ni mediana idea de como empezar, pues el primer paso era resolver el problema de ¿ como convierto un archivo C en un demonio?, después de un tiempo de investigación, se encontró que la estructura que un programa .C tenia que seguir para convertirse en un demonio estos son, realización de fork(), cambio de privilegios de los archivos abiertos por el demonio, cierre de lso canales básicos de comunicación por terminal y en ves de estos usar el log del sistema y utilizar un ciclo infinito debido a que el demonio siempre esta en ejecucion, dentro de este ciclo ira el codigo que ejecutara el demonio .*

*Iniciamos nuestro código creando los procesos para la tarea con fork(), verificamos los valores del pid para ver si es válido o no, luego se canbian los permisos de los futuros arhivos abierto por el demonio usando* umask(0)*.*

*Cabe acotar que se usará para compilar este proyecto un makefile debido a la facilidad y automatización de la compilación, debido a la comodidad de no tener que escribir línea por línea los diferentes comandos necesarios en la terminal ( donde un error de escritura significaría volver a empezar) y a la ventaja de instalar la compilación del código como si fuera un servicio del sistema operativo, gracias a esto se puede controlar el inicio y detención del proceso usando un shellcript previamente proporcionado, init.sh.*

*Antes de que el programa lleve acabo sus tareas se verifica que exista el directorio del archivo .ini , esta verificación es realizada por la función* Comprobar\_Carpeta(), en la cual se utilizan funciones del header dirent.h, para ignorar carpetas y archivos que no interesan, hasta que se encuentre el archivo .ini solicitado en el directorio, si no se encuentra o la carpeta no existe en el directorio solicitado, el programa generara un error.*.*

*Para la impresión de errores se utiliza el llamados de escritura en syslog(), debido a que, primero es para escribir como se nos indica en el enunciado y segundo porque un deamon no se comunica o informa errores por la impresión normal en la terminal, si no que genera mensajes directos al syslog del sistema operativo, para lograr esto cerrarnos los descriptores básicos (STDIN, STDOUT, STDEER).*

*Después de las primeras verificaciones, se procede con la inicialización del demonio se abre el archivo .ini que contiene y realiza la lectura del archivo, específicamente se toma el LOG\_TAG, que será el identificador de los mensajes que debe generar el demonio en el syslog y se tomara el INTERVALO, es cual es un valor que determinara cuando tiempo el demonio realizara Sleep() ( el demonio se detiene durante un tiempo y vuelve a iniciarse).*

*Con el demonio ya inicializado el siguiente paso es verificar los archivos de la carpeta var/log y sus respectivos md5, para esto se implemento una estructura lista con algunas de sus funciones básicas adaptadas al problema, en la cual cada Nodo de la lista representaba el nombre completo de un archivo ( concatenado, su direccion en el directorio con su nombre) y su actual md5, de tal forma que usando una función comparar() declarada en la implementación de la lista fuera sencillo determinar si el md5 habia cambiado, si ha cambio se empaqueta ( comprime), en el caso que ni siquiera exista el archivo en la lista, se inserta() por primera vez en la lista y su md5 es 0.*

*Para poder hacer estas verificación de los md5 correctamente se necesitaba que cada proceso hijo ejecutándose revisara el md5 de un archivo y estos lo retornaran al proceso padre para poder hacer la comparación con la lista, esto se realiza en la función* retornar\_md5(), para poder comunicarse entre el proceso hijo y el padre implementamos un pipe no nominal el cual es un enlace de comunicación unidireccional que puede crearse usando la llamada al sistema pipe. Al mismo tiempo en la función retornar\_md5() es en la cual se obtiene el md5 al pasarse por parámetro la ruta concatenado con el nombre del archivo y ejecutando "md5sum" ( programa que devuelve el md5 de un archivo) utilizando execlp ( una variación de la llamada al sistema exec, que permite utilizar programa ejecutables como si fueran llamadas de funciones). este md5, es enviado usando pipe al padre para su verificación.

*Cuando se halla verificado todos los archivos, llega el momento de empaquetar, se realiza siguiendo los estándares que se requieren en el proyecto, se abren todos los archivos a empaquetar, toda su información se empieza a volcar en un único archivo .pak, para lograr que este archivo tuviera el nombre con el estándar requerido se utilizo las librería time.h, para poder obtener la hora y fecha del sistema y así poder concatenarlo con el nombre.*

*Finalmente con toda la información de los log que su md5 se modifico en un archivo .pak que fue escrito siguiendo los estándares requeridos en el planteamiento del problema, se se realiza finalmente el empaquetado del archivo, utilizando para esto la función comprimir, la cual vuelve a dar uso a la llamada del sistema execlp para ejecutar el programa de compresión gzip.*

*Conclusión*

*Después de tanto trabajo elaborando este proyecto se percató que la herramienta llamada demonio puede ser muy útil para el sistema operativo a la hora de hacer análisis a diestra y siniestra durante el tiempo de vida que dure encendida la máquina, e incluso después de haber sido apagada, su función se puede retomar una vez se vuelva a iniciar sin necesidad de volver a iniciar su estado, siendo de mucha utilidad en infinidades de tareas que un sistema operativo necesita cumplir para garantizar el mayor rendimiento y seguridad para el usuario del computador que este utilizando.*

*Y el uso de cada herramienta proporcionada a la hora de hacer un demonio, resulta de mucha utilidad cuando se desea “echar código”, facilitando la implementación del mismo. Herramientas de mucha importancia para el trabajo de hoy en día a la hora de buscar innovación en el campo de los sistemas operativos actuales.*

##### Agradecimientos

Queremos agradecer a: K.G, J.M, J.A y J.K por su gran apoyo y colaboración, al grupo docente de SO por la ayuda proporcionada y el material dado. A nuestros padres, que nos dieron muchos ánimos durante el transcurso del proyecto.

##### Referencias

[1] W. Stallings, Operating Systems: Internals and Design Principles, 7th ed., Prentice Hall, 1 Lake Street, Upper Saddle River, New Jersey, 07458, 2012

[2] A. Silberschatz, P. B. Galvin, G.Garne, Operating System Concepts, 8th ed., Wiley, 2008

[3] <https://pasky.wordpress.com/2009/08/05/funciones-opendir-readdir-y-closedir-en-c/>

[4]<http://pubs.opengroup.org/onlinepubs/7908799/xsh/readdir.html>

[http://linux.die.net/man/3/closedirUnmask](http://h4ttp://linux.die.net/man/3/closedirUnmask)

[6]<http://pubs.opengroup.org/onlinepubs/009695399/functions/umask.html>

[7]<http://pubs.opengroup.org/onlinepubs/007908775/xsh/dirent.h.html>

[8]<http://pubs.opengroup.org/onlinepubs/009695399/functions/chdir.html>

[9]<http://www.thegeekstuff.com/2012/02/c-daemon-process/>

[https://es.wikipedia.org/wiki/Compresi%C3%B3n\_de\_datos](https://es.wikipedia.org/wiki/Compresión_de_datos)

<http://www.iac.es/sieinvens/SINFIN/CursoUnix/cap8.php>

<https://es.wikipedia.org/wiki/Syslog>