

**Universidad Autónoma de Chihuahua**

**Facultad de Ingeniería**

**Sistemas Operativos**

**-- Examen Parcial --**

**Equipo “Helado Oscuro”**

**Integrantes:**

**Luis Julián Lozoya Navarrete**

**Efraín Gilberto Domínguez Banda**

**Ángel Torres Loya**

**Jesús Alan González Murillo**

**Procesos del sistema:**

El siguiente programa nos permite visualizar lo procesos que se llevan a cabo por el sistema, todo esto con ayuda de varios subsistemas con los que cuenta el kernel, que no son mas que componentes internos que gestionan funciones específicas del sistema operativo. Estos subsistemas son módulos de software dentro del kernel de Linux que se encargan de tareas especializadas, como la gestión de la memoria, el sistema de archivos, la red, el control de procesos, la gestión de dispositivos, entre otros.

Cada subsistema tiene su propia funcionalidad y está diseñado para trabajar en conjunto para proporcionar un entorno operativo completo y funcional en el sistema Linux.

En esta ocasión especificaremos que subsistemas influyen para el correcto funcionamiento del siguiente programa desarrollado en C.

**Explicación del programa:**

Empezamos incluyendo las bibliotecas necesarias para el correcto funcionamiento del código, y la declaración de dos variables globales:

**line**. Nos permitirá apuntar al contenido dentro del archivo cmdline para determinar el nombre de los procesos de usuario.

**tam**. Nos ayudará a medir la longitud de los archivos para determinar si el proceso es un proceso de kernel o un proceso de usuario.

Texto

Descripción generada automáticamente

Después esta nuestra función **get\_process\_cmd** que recibe como argumentos la dirección de un archivo y el inicio de una cadena de caracteres.

Texto

Descripción generada automáticamente

Ya en la función principal lo más importante con lo que contamos es la estructura **dirent** la cual nos permite manipular diversos directorios y leer sus contenidos, además de **pDir** la cual es una variable de tipo directorio.

Empezaremos abriendo el directorio “/proc” almacenado en **proc\_dir** con ayuda de la instrucción **opendir.**

Texto

Descripción generada automáticamente

Si no encuentra el directorio nos avisará con este mensaje y finalizará el programa.

Texto

Descripción generada automáticamente

Con ayuda de un while se recorrerán todos los subdirectorios dentro del directorio /proc, en los cuales almacenaremos su ruta en **proc\_id\_dir** para enviársela a nuestra función **get\_process\_cmd**

Texto

Descripción generada automáticamente

La función **get\_process\_cmd** funciona de la siguiente manera:

Recibe como argumentos **filename**, el cual tiene almacenado la ruta de **/proc/idProceso**, y **procname** que es donde almacenaremos el nombre del proceso.

Empezamos concatenando a la ruta almacenada en **filename** lo siguiente: **/cmdline**, teniendo ahora almacenado en filename **/proc/idProceso/cmdline** esto para abrir el archivo donde se encuentra el nombre del proceso de usuario

Texto

Descripción generada automáticamente

Después con ayuda del ciclo while obtenemos el contenido dentro del la ruta “**/proc/idProceso/cmdline**” y la almacenamos en **procname**.

**Nota:** Hay que tener en cuenta que no existirá contenido en la ruta especificada si es un proceso de kernel, por ello con ayuda de **ftell** determinamos si el archivo esta vacío o no.

Texto

Descripción generada automáticamente

Entonces en caso de que sea un proceso de kernel, la longitud del archivo será cero, por lo que cambiaremos la ruta, en lugar de finalizar con /cmdline ahora finalizará con **/stat**, que es donde podremos encontrar el nombre del proceso de kernel.

Texto

Descripción generada automáticamente

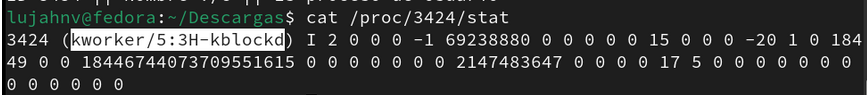
De igual forma abriremos el archivo, pero ahora con la finalización de la ruta en /stat y obtendremos con **fgets** lo que tiene contenido.

Texto

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Dentro de su contenido vendrán varios parámetros, pero lo que nos importa es lo que esta dentro de los paréntesis.



Entonces con **sscanf** le mandaremos la línea que acabamos de obtener en el fgets y filtraremos el contendido indicándole que no queremos el id que viene al inicio de la línea (con la instrucción %\*d) y que debe empezar después del paréntesis abierto y finalizar en el paréntesis cerrado. Todo esto será almacenado en **procname**.

Y por último cerraremos el archivo.

Texto

Descripción generada automáticamente

Ya devuelta en la función principal imprimiremos el **id del proceso**, el **nombre** del proceso obtenido en nuestra función anterior y determinamos si es proceso de kernel o no dependiendo del valor almacenado en tam.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Salida del programa.**

Imagen que contiene Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Como salida tenemos el ID del proceso, su nombre y si este es proceso de usuario o de kernel.

**Los subsistemas relacionados a nuestro programa son:**

**Filesystem (/proc)**

El sistema de archivos "/proc" es un sistema de archivos virtual creado en la memoria por el Kernel, y se utiliza para conocer información acerca de los procesos del sistema.

El programa interactúa con el subsistema al usar "opendir" para abrir el directorio /proc y "readdir" para leer su contenido, también al utilizar "fopen" para abrir el archivo cmdline de un proceso en específico de /proc, al mismo tiempo se está utilizando el subsistema de entrada/salida (I/O) ya que "opendir", "readdir", "fclose", "fopen", entro otras, son funciones que están accediendo y leyendo el contenido de los archivos.

**Gestión de Procesos (Process Management)**

Este subsistema del Kernel de Linux se encarga de administrar los procesos en ejecución en el sistema, asignando un identificador único (PID) a cada proceso.

El programa interactua con este subsistema al verificar si la entrada del directorio es un ID de proceso en "isdigit(entry->d\_name[0])", lo que permite saber cuáles son los procesos que están activos.

De igual manera en "sprintf(proc\_id\_dir, "/proc/%s/cmdline", entry->d\_name)" que se encarga de construir la ruta al archivo cmdline de cada proceso.

**Gestión de Memoria (Memory Management).**

Como su nombre lo indica, se encarga de administrar la memoria en el sistema, ya sea memoria física o virtual.

Su interacción con el programa está en "getline(&line, &len, fp)" que se encarga de leer las lineas del archivo usando la asiganción dinámica de memoria con el símbolo (&)