10:8

Hoja 1	Ejercicio 1	Eric Brandwein LU 349/16

```
String nombre_dir(file_descriptor){
    int num_inodo_padre= numero_inodo(file_descriptor, 1);
    int inodo_propio=numero_inodo(file_descriptor, 0);
    Ext2FSInode inodo_padre=load_inode(num_inodo_padre);
    int datos[]=inodo_padre.indices_de_bloques_de_datos;
    //Tenemos que fijarnos si el directorio con el número
    //de inodo igual a inodo_propio está en algún bloque de
    //datos sin indirección, o con uno de los otros 3 grados de
    //indirección. Asumimos para este pseudocódigo que
    //se encuentra en el primer bloque sin indirección, pero
    //en el caso real nos deberíamos fijar en los siguientes 11
    //directos y en los 3 indirectos si es que no se encuentra
    //en el primero.
    int número_de_bloque = datos[0];
    bloque = obtener_bloque(número_de_bloque);
    For (entry : bloque){
        //cada "entry" es una entrada de archivo, con su nombre
        //y su inodo, en el DirEntry, y otras cosas más.
        if(entry.numero_inodo == inodo_propio){
            return entry.nombre;
            // el nombre puede ser muy largo, pero asumimos
            //que no se pasa del bloque.
        }
    }
}

int numero_inodo(file_descriptor, indice){
    Entries = read(file_descriptor);
    return entries[indice].numero_inodo;
}
```

sigue →

En el pseudocódigo asumimos que la función obtener_bloque está a nuestra disposición, que lee el contenido de un bloque en ~~el disco~~ la partición Ext2 que se está usando.

Lo que hacemos es obtener los números de inodo del directorio '.' y '..' del file_descriptor pasado, y usar los mismos para buscar el nombre en el directorio padre. '.' tendrá índice 0 en la lista de DirEntrys, y '..' tendrá índice 1.

Hoja 2	Ejercicio 2	Eric Brandwein LU 349/16

a) Bob está usando la función gets, que lee la entrada hasta un fin de línea y ~~guarda~~ escribe los caracteres en el buffer pasado como parámetro. Un atacante podría aprovecharse de esto y realizar un ataque de buffer overflow, escribiendo en la entrada más de 250 caracteres. Si escribe los suficientes, podría sobreescribir ~~el valor de~~ la dirección de retorno ~~en el~~ de la función en el stack, y así ejecutar código malicioso, escrito por él, si es que el stack es ejecutable, o ejecutar funciones que no se suponía que fuesen ejecutadas en ese momento.
Para lograrlo, sin embargo, debería sobreescribir todas las variables locales de la función, por ser passwordConfirmación la última declarada. Esto de todos modos no presenta un problema, ya que si sobreescribe la variable password con algo diferente a passwordConfirmación, el programa no entrará al último IF y retornará ~~sin error~~. inmediatamente.

b) Otro ~~método~~ problema que podría ocasionar el atacante es el de sobreescribir las contraseñas de otros usuarios. Al tener la posibilidad de cambiar el valor de la variable numeroDeUsuario, podría llamarse a la función actualizarPassword con un usuario diferente al actual. En este caso, el atacante debería conocer el número de usuario del ~~atacado~~. sobre el que quiere perjudicar.

c) Bob podría setear el bit de setuid ~~en~~ en los permisos del archivo, que indica que, cuando se ejecute, pase a tener los mismos permisos que el dueño del archivo, que es Bob.

d) Si el archivo de contraseñas estuviera en texto plano, Alice podría ingresar con la contraseña de root o de cualquier otro usuario y hacerle lo que quisiera al sistema, como ser cambiar todas las contraseñas. En cambio, si ~~estuviera en texto plano~~ lo almacenado fuese los funciones de hash o los de hash+salt, Alice debería

de alguna forma adivinar qué cadena de caracteres podría generar ese hash, cosa que, dependiendo del algoritmo de hashing, podría ser muy difícil.

N.º 8

Hoja 3	Ejercicio 3	Eric Brandwein LU 349/16

a) i) Lo ideal sería intercambiarlo lo más rápido posible, ya que la tardanza en reconocer entrada del usuario en un celular degrada mucho la experiencia.

ii) Para que el uso de la placa de red sea más esporádico, almacenar muchos datos para después enviarlos es una buena estrategia.

iv) Una GPU puede realizar muchos cálculos al mismo tiempo, por su cantidad incrementada de núcleos de ejecución. Esperar a tener suficientes cálculos encolados antes de hacer correr a la GPU podría ahorrar ciclos de vida y energía gastada.

v) Sería preferible intercambiarlo lo más rápido posible, otra vez por el tema de la respuesta a las entradas del usuario.

b) i) Depende de si se desea que, si la respuesta a la entrada del usuario no es inmediata, se pueda responder más tarde o no. Por ejemplo, si el sistema está ocupado procesando y el usuario mueve el mouse pero no se mueve el puntero, quizá sería extraño que el puntero se mueva y cliquee cosas más tarde.

ii) Se podría utilizar spooling para subir los archivos a la nube de a uno, mientras que ya se encuentran presentes en la carpeta local.

iv) Como lo que se espera de una GPU en este caso son los resultados, usar spooling no aportaría a mejorar el rendimiento.

c) i) No, porque cada toque del usuario debe normalmente ser procesado inmediatamente.

ii) Podría utilizarse DMA para comunicar los contenidos de dos archivos a la placa de red sin pasar por el procesador, y así liberarlo para que realice otros cálculos.

iv) Si es un cálculo con muchos pasos intermedios, usar DMA dentro de la GPU podría ayudar a la performance.

v) Quizá podría utilizarse ~~polling~~ si es que lo que se comunica es la posición del mouse en vez del movimiento del mismo. así, si se desea conocer la posición, solamente se debe consultar a la memoria, y no al dispositivo.

d) i) Interrupciones. La entrada del usuario es un evento que no ocurre seguido, y usar polling gastaría muchos ciclos de reloj.

ii) Para consultar si el servidor remoto contiene nuevos archivos, utilizaría polling. En cambio, cuando se agrega ~~un archivo~~ un archivo nuevo al directorio local, se comunicaría con el servidor lo más rapidamente posible.

iv) Interrupciones. El cálculo de la GPU es un proceso que podría demorar, así que hacer polling podría de nuevo gastar muchos ciclos de reloj.

v) Para los clicks, interrupciones. Para la posición actual del puntero, podría usarse polling si es que el valor está en memoria, ya que el movimiento del mouse es algo más continuo, y para no demorar al procesador con el cambio de contexto que implica una interrupción.

B/EN

N.O.: 8

Hoja 4	Ejercicio 4	Eric Brandwein LU 349/16

a) Si consideramos al que opere primero como el que primero cree el archivo de operte, el protocolo no ~~hace~~ cumple iii). Podría llegar a ocurrir que al nodo subastador le llegue el mensaje de creación del archivo más tarde que el del de creación de un archivo posterior, y que no tenga manera de saber cuál fue creado primero. Aunque los archivos tuvieran el tiempo de creación, cada nodo podría tener un tiempo mínimamente diferente a todos los otros, y entonces el tiempo de creación podría no ser el verdadero.

b) El protocolo sufre del problema del "acuerdo bizantino". Si, por ejemplo, el subastador dejase de funcionar justo después de crear el archivo ganadorES, y el adquiriente respondiera que sí quiere el lote, el mismo asumiría que lo adquirió, cuando en realidad el subastador nunca registró la adquisición. Si le agregásemos un mensaje de confirmación por parte del subastador al protocolo, seguiría ocurriendo una situación similar: podría caerse el adquiriente y el subastador asumir que ~~se~~ se concretó la compra. Si los paquetes se pudieran ~~perder, lo mismo~~ demorar o perder, la situación empeoraría: podría pasar que un nodo se quedase ~~esp~~ esperando una confirmación que nunca llegaría, o que llegaría muy tarde.
Si asumimos que estos problemas no podrían ocurrir, y que tampoco se cae la red, el protocolo podría llegar a funcionar.