Sistemas Operativos

Departamento de Computación – FCEyN – UBA
Segundo cuatrimestre de 2019

1	2	3	4	Nota
16	20	25	25	86

Segundo parcial – 05/11 - Segundo cuatrimestre de 2019

- 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)
Considere que se tiene un file system Ext2, y éste contiene el archivo test.txt de 200 MB, en la ruta /home/rb. El FS maneja bloques de tamaño 8 KB. Se usan 32 bits para direccionar bloques, y la memoria usa palabras de 32 bits. Si se requieren leer los primeros 100000 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?
b) ¿En qué orden y en qué forma (leer, escribir) deben ser utilizadas esas estructuras?
c) ¿Aproximadamente cuántos accesos de escritura y de lectura a disco se deben hacer? Justifique su respuesta, y mencione qué consideraciones tomó en cuenta.
d) ¿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)

Escriba el código de un módulo `/dev/invertir` el cual funcione de la siguiente manera: cada vez que el usuario escribe una cadena en el dispositivo, se almacena en un buffer. Luego el contenido del buffer se invierte y se almacena el resultado en la memoria del usuario. Por ejemplo, si la cadena inicial es "sistemas", el resultado almacenado es "sametsis". Cada vez que el usuario lee del dispositivo, se devuelve la cadena invertida, y su número de caracteres.

Funciones útiles:

```
copy_to_user // para copiar a la memoria del usuario.
copy_from_user // para copiar desde la memoria del usuario.
kmalloc //pedir memoria en modo kernel.
kfree // liberar memoria en modo kernel.
strlen // contar caracteres en un string.
invstr // invierte una cadena.
```

Debe incluir en su código las estructuras y la inicialización/finalización requeridas para el módulo.

Ejercicio 3.(25 puntos)
En el contexto de los sistemas distribuidos, particularmente en la parte vinculada al consenso entre participantes, escriba el pseudocódigo del protocolo *commit* de dos fases (*two phase commit protocol*), tanto para el coordinador como para los participantes. Describa un ejemplo de aplicación, y muestre su funcionamiento usando diagramas (de eventos, de estados, etc).

Ejercicio 4.(25 puntos)
Hay una vulnerabilidad en el código fuente que figura a continuación, y que permite que se ejecuten comandos arbitrarios. El programa permite cambiar un parámetro de red del kernel de linux.

```
int main(int argc, char **argv, char **envp)
{
  char *buffer;
  char c;
  gid_t gid;
  uid_t uid;

  gid = getegid();
  uid = geteuid();
  setresgid(gid, gid, gid);
  setresuid(uid, uid, uid);

  printf("Elija el valor a setear el parámetro de tcp_syncookies (0/1)");
  c = getchar();

  asprintf(&buffer, "sysctl -w net.ipv4.tcp_syncookies=%c", c);
  printf("a punto de invocar system(\"%s\")\n", buffer);
  system(buffer);
}
```

- ¿Qué condiciones se deben cumplir para que el ataque puede realizarse y obtener mayores privilegios que los del usuario que lo invoca?
- Describa los pasos necesarios para poder llegar a perpetrar un ataque al binario que se genera compilando este código.

Ayuda:

- El comando sysctl se usa para configurar o ver los parámetros del kernel en tiempo de ejecución.
- net.ipv4.tcp_syncookies es un parámetro del kernel, que activa un modo de protección contra ataques DDoS.

22

2|4 B-

(2)

(b declaré ~~char* buffer = NULL;~~ en init)

```
size_t invertir_write (file* filp, char* data, size_t size){
    buffer = kmalloc (size+1, FGP_KERNEL);   → memory leak
    copy_from_user (buffer, data, size);
    buffer [size] = 0;   //no está de más agregar fin de línea
    invstr (buffer);
    size_t escrito = copy_to_user(data, buffer, size);   NO devuelve error, sino read
    return escrito;
}

size_t invertir_read(file* filp, char* data){
    if (buffer != NULL){
        int len = strlen(buffer);   //tiene fin de línea así que ojo.
        char* temp;
        //o el formato que corresponda...
        asprintf (&temp, "la cadena invertida es: %c y su longitud, %d", buffer, len);
        size_t leído = copy_to_user (data, buffer, strlen(temp));
        return leído;
    }
    return -1;
}

static struct file_operations mifops {
    .OWNER = THIS_MODULE;
    .READ = &invertir_read;
    .write = &invertir_write;
}
```

```
// los tipos y unidades de esta parte (y de todo el ejercicio pueden no ser del todo correctos, pero me concentré en que conceptualmente los pasos sean correctos.
static struct micdev {};
static int midevno ;
static struct miclass {};
//inicializo cdev, devno y class

invertir_init(void){
    cdev_init(&micdev, &mifops);
    chrdev_region_alloc(&devno, "naubre"); // salteo los parámetros que no eran tan importantes :D
    add_device(&micdev, devno)
    class = create_class(&class)
    // y me falta montar el dispositivo, pero no recuerdo el comando.
    char* buffer = NULL;
    return 0;
}

invertir_exit(void){
    cdev_del(&micdev);
    unregister_chrdev_region(devno);
    class_destroy(&class);
    kfree(buffer);
    return 0;
}

MODULE_INIT(invertir_init);
MODULE_EXIT(invertir_exit);
```

// en clase lo pusimos al principio del código pero me pareció más prolijo en init. Creo que está bien

No, No es global así. Si la declaras acá vive en init.

③ Un protocolo de commit de dos fases podría usarse, por ejemplo, para coordinar ~~transacciones~~ compras de divisas ~~al~~ a un banco por parte de sus usuarios, como el ejemplo visto en la práctica. El coordinador sería una entidad que recibe los pedidos de compra, y este se comunica con una ~~otra~~ que maneje el stock de divisas y otra que revise los balances de los usuarios.

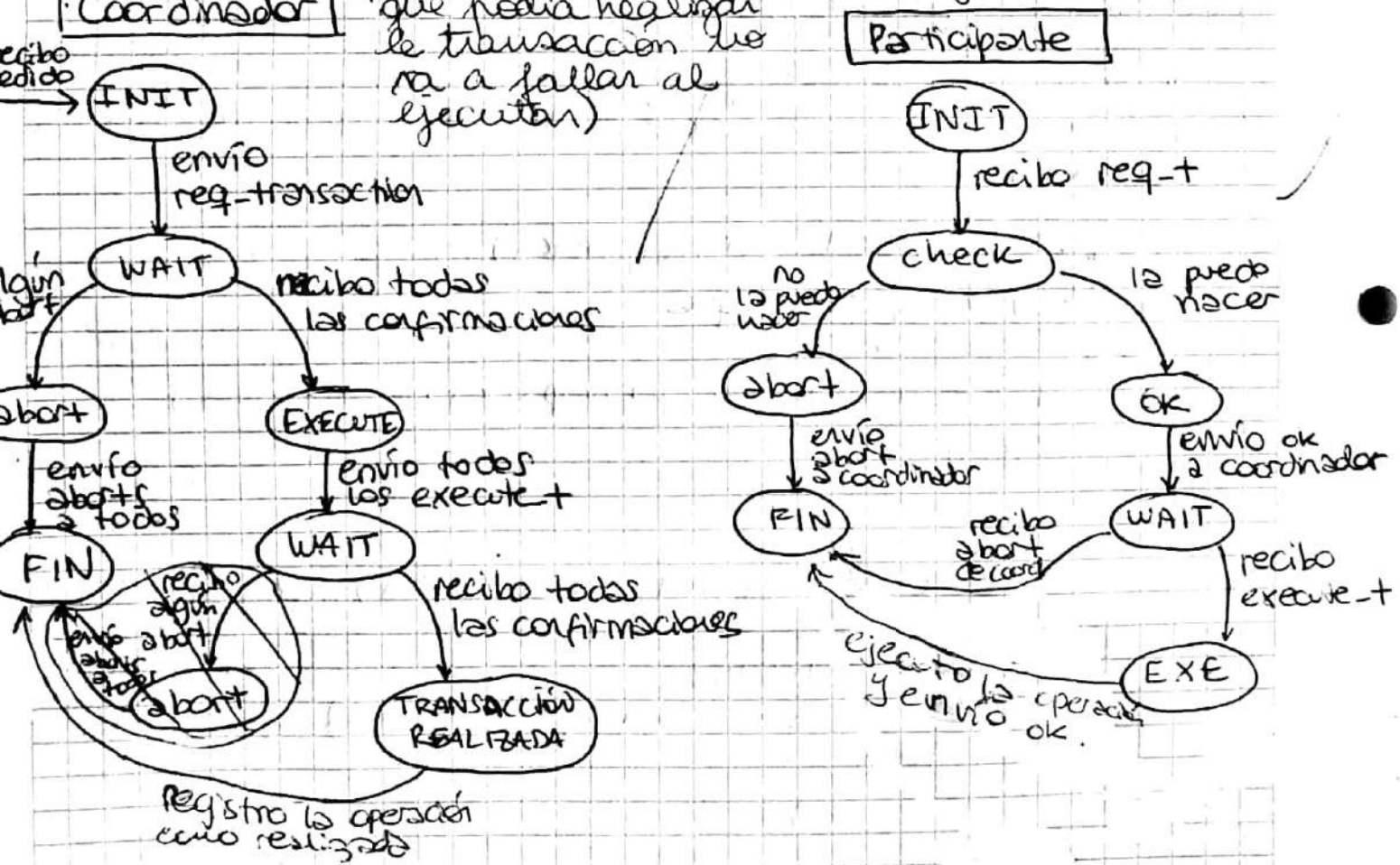
Funciona del siguiente modo; asumiendo que no hay pérdida de mensajes (timeouts):

```
coordinador (pedido){
    // pregunta a los participantes si pueden
    // hacer la transacción (NO dice que lo hagan
    aún)
    req_transaction (participantes, info_pedido)
    wait()
    if (algún abort){
        abort_transaction (participantes, info_pedido)
        return -1;
    }
    if (todos ok){
        execute_transaction (participantes, info_pedido)
        wait()
        if (algún abort){
            abort_transaction (participantes, info_pedido)
            return -1;
        }
        if (todos ok){
            return 0;
        }
    }
}
```

→ la info. que envía el coordinador

```
participante (info_pedido){
 if(req_transaction) {
    if(check_transaction (info_pedido) ) {
        req_ok (coordinador)
    }else { abort (coordinador)}
 }
 if( execute_transaction) {                    → "mi función de ejecución
    if ( execute_transaction (info_pedido)){
        exe_ok (coordinador)
    }else { abort (coordinador)}
 }
 if(abort){ abort (pedido)}
}
```

Diagrama de estados: (sin timeouts / pérdida de mensajes)
(Asumiendo que si un participante confirma que podía hacerlo la transacción no va a fallar al ejecutar)



~~Observación: habría que definir cuánto tiempo espera un participante por un posible abort después~~

(4) Como sysctl no se llama aclarando su ruta, se podría crear un nuevo ejecutable con el mismo nombre, ~~al~~ que pueda ejecutar ~~acceder~~ aquel usuario cuyos permisos nos gustaría obtener. Si en PATH colocamos primero la ruta de nuestro nuevo archivo sysctl, el programa lo encontrará antes que al código real de sysctl y lo ejecutará en su lugar.

Hay que tener en cuenta que este ataque nos dará los privilegios que tenga quien cree el binario: si el flag setuid del binario está prendido, podremos ejecutarlo nosotros y obtener los permisos de su creador, que podrían ser mayores a los nuestros.