Lápiz

Sistemas Operativos

Departamento de Computación – FCEyN – UBA
Segundo cuatrimestre de 2019

Nombre y apellido: _____

Nº orden: _____ L.U.: _____ Cant. hojas: 6

1	2	3	4	Nota
21	25	22	25	93

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:

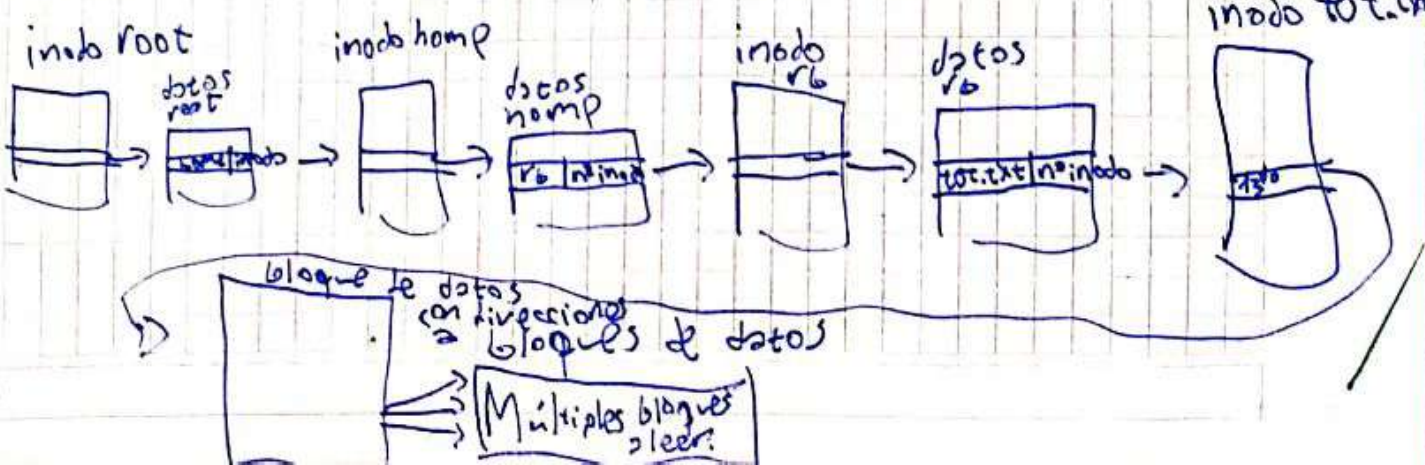
- El comando sysctl se usa para configurar o ver los parámetros del kernel en tiempo de ejecución.
- net.ipv4.tcp_syncookies es un parámetro del kernel, que activa un modo de protección contra ataques DDoS.

1)

a) Para hacer dicha lectura, se deberá utilizar el superbloque, ya que en él tengo información crítica del file system, como por ejemplo el tamaño de cada bloque. Puedo encontrar una copia del mismo en cada block group. También debo utilizar múltiples inodos y bloques de datos, en particular, el inodo especial correspondiente al directorio root, que utilizaré para encontrar el bloque de datos que contenga la info del directorio home. También deberé acceder a los inodos y bloques de datos del resto de los directorios y del archivo test.txt.

b) Primero debo acceder al inodo especial correspondiente al directorio root. En él, encontraré las direcciones de los bloques de datos (asumiendo que la info del directorio home se encuentre entre los primeros 12 bloques de datos, caso contrario tendría indirecciones simples, dobles o triples). Luego (asumiendo que se encuentra en el primer bloque) leeré desde el bloque de datos, el número de inodo que se corresponda con la entrada del directorio home. Después, en el inodo del directorio home (lo traeré del disco si no estaba en memoria), leeré

la dirección (o indirección) del bloque de datos donde se encuentre la entrada correspondiente al directorio rb (Si no está en el primer bloque de datos, debo explorar los siguientes bloques, eventualmente con indirecciones simples, dobles y/o triples) hasta encontrarlo). Una vez encuentre el nº de inodo correspondiente al directorio rb, lo traeré a memoria (si no estaba), y en el buscaré el bloque de datos donde se encuentre la entrada correspondiente al archivo test.txt (con las mismas consideraciones que para los inodos anteriores. Luego, una vez que tenga el nº de inodo correspondiente al archivo test.txt, lo traeré a memoria (si no estaba). Una vez lo tenga en memoria, iré a la 13ra entrada (con los primeros 12, de direcciones directas, puedo acceder a los primeros 98304 bytes) que se corresponde con una indirección simple. Traeré el bloque de la 13ra entrada a memoria (si no estaba), y en él encontraré 2,048 direcciones de bloques (bytes 98304 a 16,777,216-1). Luego traeré los bloques que quiera leer.

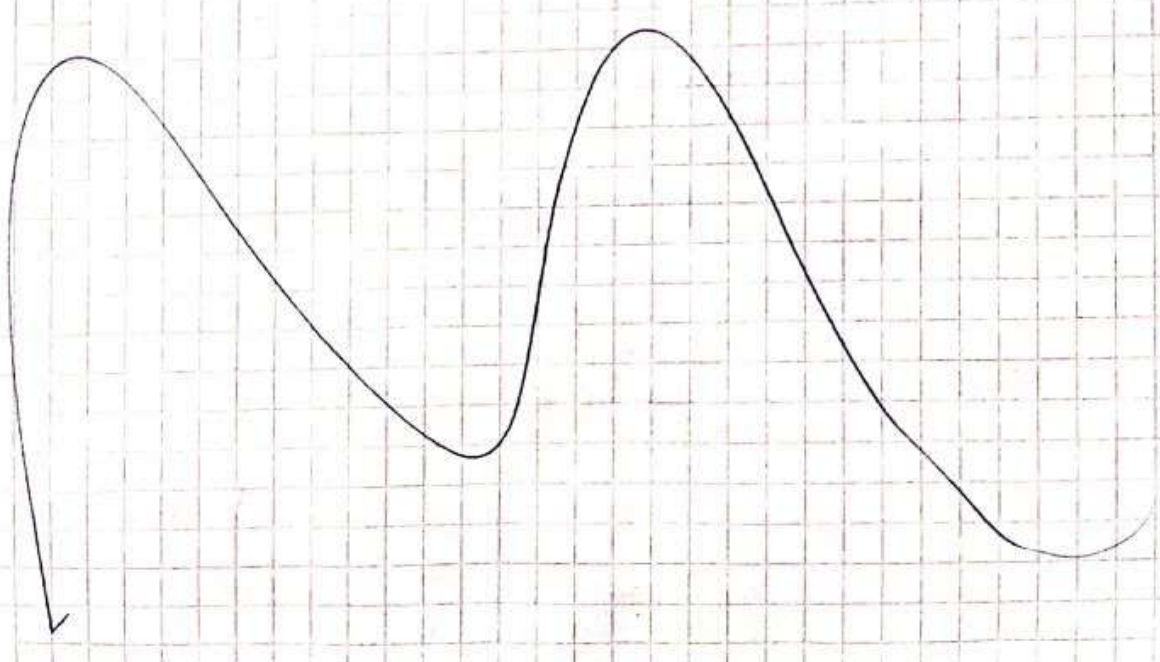


c-d)	disco	mem
• Primero cargo el superbloque a memoria	+1 lectura	+1 escritura
• Cargo el inodo del dir root	+1 l	+1 e
• Leo la entrada del inodo donde se encuentra la dir del bloque de datos que contiene la entrada correspondiente al directorio home (sumo que se encuentra en el primer bloque de datos)		+1 l
Traigo el bloque de datos que tiene dicha entrada. Y leo la entrada	+1 l	+1 e +1 l
• Traigo el inodo correspondiente al directorio home a memoria	+1 l	+1 e
• Leo la entrada del inodo correspondiente al bloque de datos que contiene la entrada del dir v6 (sumo que es el primero) (A)	+1 l	+1 l
• Traigo el inodo del dir v6	+1 l	+1 e
• Leo la entrada del inodo correspondiente al bloque de datos donde se encuentra la entrada del archivo test.txt (sumo que es el primer bloque) (B)	+1 l	+1 l +1 e +1 l
Traigo el inodo del archivo test.txt	+1 l	+1 e
Leo la 13ra entrada (indirección simple (96 KB = ~16 MB)		+1 l
Traigo el bloque con direcciones de bloques a memoria	+1 l	+1 e

	disco	mem
Leo 13 entradas del bloque de direcciones $\lceil (100000/8192) \rceil$		13 l
Traigo las 13 bloques a mem	13 l	13 e
	22 l	20 l
	0 e	22 e

En total tengo 22 lecturas desde disco y 0 escrituras. Y, 20 lecturas desde memoria y 22 escrituras a memoria.

⊗ medir los accesos a memoria en palabras, no en bloques!



2)

```
char * buffer, =NULL
mutex lock;
struct file_operations = invertir_operators{
  .owner = THIS_MODULE;
  .read = invertir_read;
  .write = invertir_write}
```

(*) (Atrás de esta pag.
module_init(invertir_init)
module_exit(invertir_exit)

```
// Asumo que solo se escribe si no había nada escrito
//antes.
int invertir_write(struct file* filep, char* data, size_t
      size, offset_t offset){
  mutex_lock(&lock);
  size_t size_form, size_eff
  if(buffer!=NULL){
    buffer = kmalloc(size, GFP_KERNEL)
    copy_from_user(buffer, data, size)
    size_form = strlen(buffer)
    size_eff = min(size_form, size)
    invstr(buffer)
    mutex_unlock(&lock)
    return size_eff.
  }
  mutex_unlock(&lock)
  return -EPERM
```

```
int invertir_read (struct File *filep, Charte data,
size_t size, offset_t offset){

    size_t size_form;
     mutex_lock (& lak)
    if(buffer != NULL){

        size_form = strlen (buffer)
        copy_to_user ( data, buffer, size_form)
        kfree ( buffer)
        buffer = NULL
         mutex_unlock (&lock)
        return  size_form;

    }
    mutex_unlock (&lock)
    return -EPERM

}

dev_t  invertir_devno;

invertir_init ( ) {
    allocate_chardev ( invertir_devno , invertir_operations
    buffer = NULL
     mutex_init(&lock,1)
     ;
     ;

}

invertir_exit ( ) {
    destroy_chardev (invertir_devno)
     mutex_destroy (&lock)
     ;
}
```

3)

Pseudocódigo Cominit de dos fases:

Coordinador:

while (true){

:espero operaciones.
// me llega una operación
- envío el mensaje prep_ok con la info de la operación a las participantes involucradas.
- Espero mensajes.
- Si me llega algún abort por parte de un participante, envío la señal abort al resto de los participantes, y vuelo al principio del ciclo.
- Una vez me llegan mensajes de todos los participantes diciendo prep_ok, envío mensajes a todos diciendo commit_ok.
- Espero mensajes.

- Una vez que recibo mensajes de todos los participantes diciendo commit_ack, se hizo efectiva la operación, y vuelvo a esperar operaciones

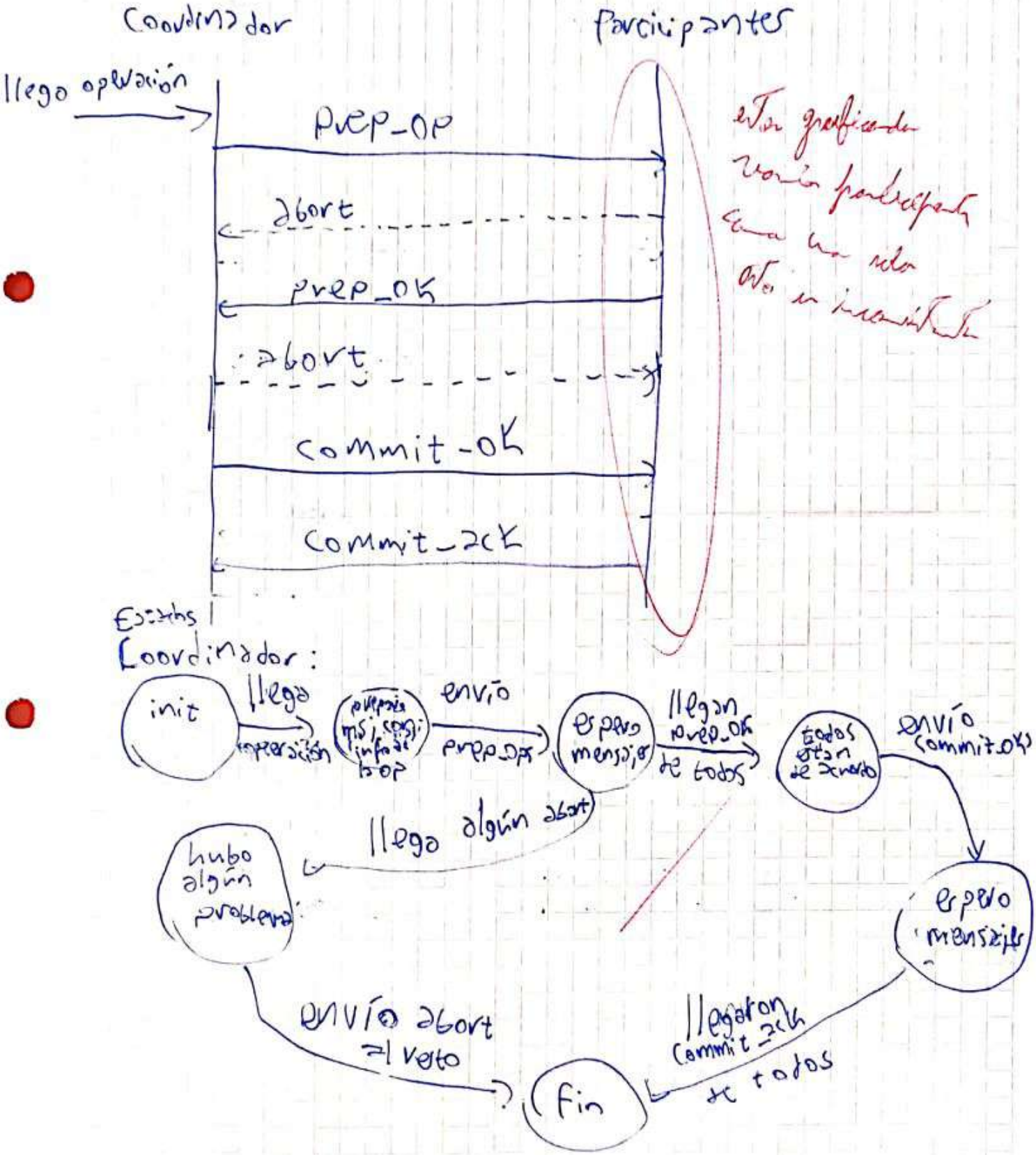
}

NOTA

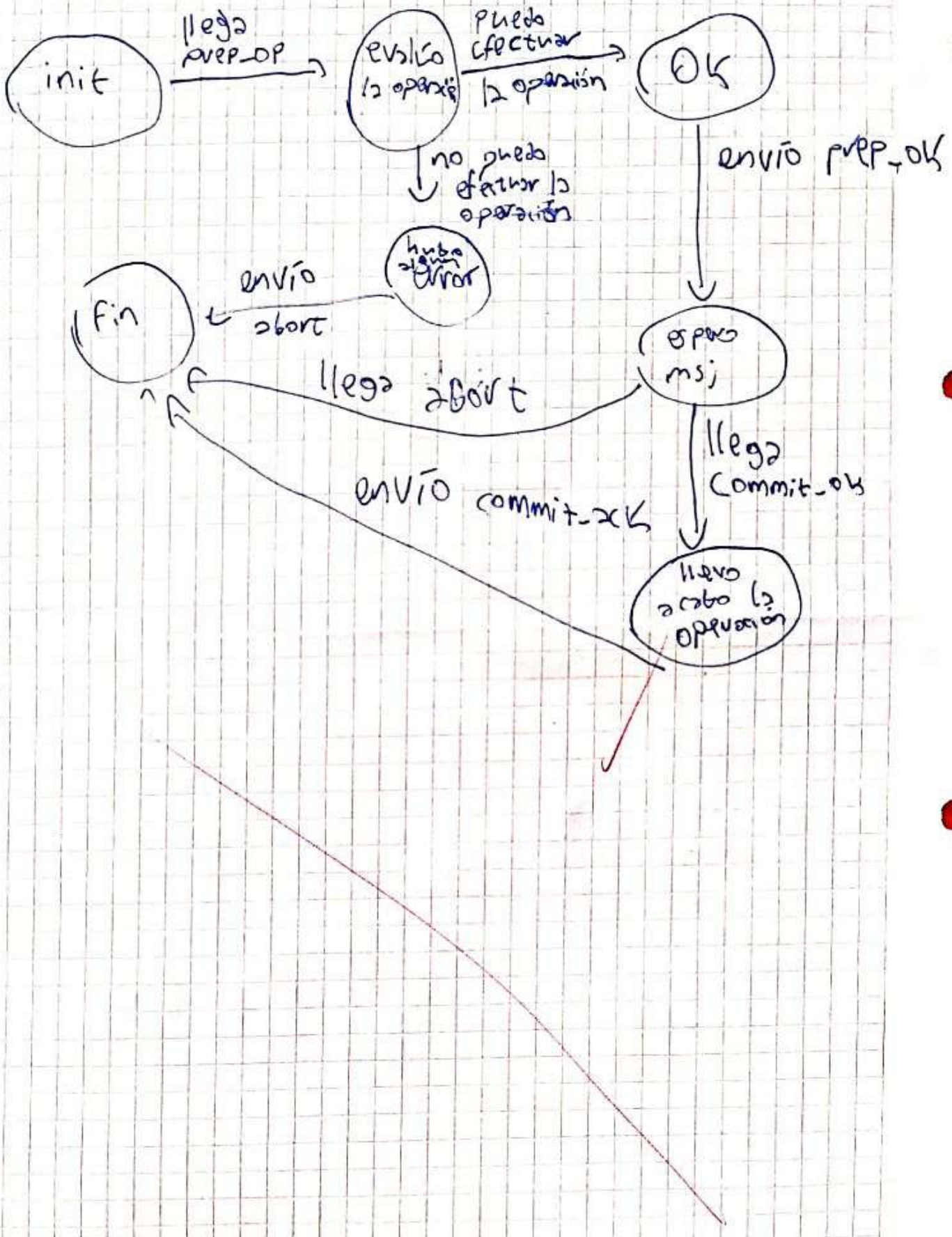
Participantes:

```
while (true) {
    . Espero mensaje prep_op del coordinador.
    . if ( puedo hacer la operación ) {
        envío prep_ok al coordinador.
      } else {
        envío abort al coordinador.
      }
    . Espero mensajes.
    . if (me llega abort) {
        . vuelo al principio del ciclo)
      } else if (me llegó commit_ok) {
        . efectúo la operación
        . envío el mensaje commit_ack al
          coordinador
      }
}
```

Mensajes


Coordinador
Participantes
llego operación
prep_op
abort
prep_ok
abort
commit_ok
commit_ack
Estados
Coordinador:
init
llega operación
envío prep_op
espero mensajes
llegan prep_ok de todos
todos están de acuerdo
envío commit_ok
llega algún abort
hubo algún problema
envío abort = veto
fin
llegaron commit_ack de todos

Participante



NOTA

4)

Un ataque posible sería modificar la variable de ambiente $PATH$, agregandole al principio un directorio que contenga un binario bajo el nombre systctl que haga lo que yo desee, por ejemplo system("rm -rf -/").

Para que esto se pueda lograr, el usuario que lo invoca debe tener permisos de ejecución para el binario, y además, para obtener un mayor privilegio, el creador del mismo debe tener un mayor privilegio que el que lo invoca, y haber seteado el permiso setUid en el binario en cuestión (así el invocante puede correr el binario con los privilegios del creador).

Este ataque se podría prevenir detallando explícitamente la dirección del binario systctl, dentro de la llamada a sprintf.

Para modificar la variable $PATH$ al cantar con ejecutar el comando export $PATH$ = /tmp : $PATH$ (siendo tmp el directorio donde se encuentra el binario antes mencionado) antes de invocar al procedimiento