SO: 2do Parcial

Galileo Cappella

2c2022

Disclaimer y Notas

En el exámen me saqué un 80 con [27, 15, 20, 18].
Las correcciones están marcadas con un bloque gris.
Cualquier respuesta es posible que diferentes profesores la corrijan diferente, recomiendo que te fijes cómo la explicarías vos si no te convence.
¡Mucha suerte!

Question 1: Sistemas de Archivos (30 puntos)

Se tiene un disco formateado con `FAT` para el cual se quiere poder devolver el contenido de un archivo a partir de su ruta absoluta. Para ello se debe implementar la función:

```
datos = cargar_archivo(directorios[])
```

donde `directorios` es la lista de los nombre de los directorios de la ruta (ordenados, incluyendo el nombre del archivo, y sin incluir a `root`). Es decir, si el archivo a abrir es `\Documentos\Users\foto.png` entonces `directorios = ['Documentos', 'Users', 'foto.png']`.

Para ello se cuenta con las siguientes funciones auxiliares:

- `FAT_entry(block_address)` que devuelve la entrada de la tabla `FAT` de la posición block_address.
- `raw_data = read_blocks(block_address1, block_address2, ...)` que lee del disco todos los bloques indicados por parámetro, en orden.
- `parse_directory_entries(raw_data)` que devuelve una lista de `struct_entrada_directorio` que representan las entradas de directorio del directorio pasado en `raw_data`.
- `raw_data = root_table()` que devuelve los datos de la tabla de directorios de root.

a) Enumere tres campos que debe tener según este tipo de filesystem, la estructura `struct_entrada_directorio`.

b) Escribir el pseudo-código de la función `cargar_archivo`.

Solution:

a) Debe tener:

-) Tipo. Para distinguir entre archivos, directorios, y otros.

-) Nombre. Único dentro del directorio (8+3 en FAT32, sin LFN).

-) Primer bloque del cluster. Para cargar los datos y seguir el cluster en la FAT.

b) Arranca leyendo el directorio root, y luego sigue el path hasta encontrar el archivo.

```
1:  procedure CARGAR_ARCHIVO(directorios[])
2:      raw_data ← root_table();                          ▷ A: Arranco con el root
3:      for name ∈ directorios do                         ▷ A: Por cada parte del path, en orden
4:          dir ← parse_directory_entries(raw_data);
5:          entry ← find_entry(dir, name);
6:          if entry = NULL then                          ▷ A: No existe
7:              return NULL;
8:          end if
9:          blocks ← get_blocks(entry);
10:         raw_data ← read_blocks(blocks);
11:     end for
12:     return raw_data;
```

Note:-

Acá devuelvo raw_data completo, pero el archivo capaz no cubre el último bloque entero.
Ejemplo: Si tenemos bloques de 512B y el archivo mide 513B, acá devuelvo 1024B.

```
13: end procedure

14: procedure FIND_ENTRY(dir, name)
15:     for entry ∈ dir do
16:         if entry.name = name then
17:             return entry;
18:         end if
19:     end for
20: end procedure

21: procedure GET_BLOCKS(entry)
22:     last_block ← entry.firstBlock;
23:     blocks = [last_block];
24:     while True do                                     ▷ A: Hasta el fin del cluster
25:         last_block ← FAT_entry(last_block);
26:         if last_block = 0 then                        ▷ A: Fin del cluster
27:             break;
28:         else
29:             blocks.append(last_block);
30:         end if
31:     end while
32:     return blocks;
33: end procedure
```

Question 2: Sistema de E/S - Drivers (25 puntos)

Nos piden desarrollar el driver para el nuevo *Palopalooza*. El mismo recibirá desde un servidor web las canciones que los clientes piden. Luego el sistema se encargará de buscar el mp3 de la canción y enviar los bytes de la canción hacia el sistema de parlantes. El driver se encargará de enviar dicha música hacia alguna de las pistas disponibles (es decir que no estén pasando música). Se cuenta con 3 pistas (y 3 sistemas de parlantes) y no se quiere que ninguna esté sin música si hay pedidos de usuarios.

Para esto, las canciones se procesan secuencialmente por orden de llegada (i.e. *FIFO*), pudiendo procesarse hasta 3 trabajos en simultáneo, uno por cada pista. Los trabajos consisten en una secuencia de comandos comenzando con la syscall `open` hasta el llamado a la syscall `close`. Si hay 3 trabajos abiertos y no cerrados, el próximo quedará bloqueado en la syscall `open`.

Mientras no se llame a la syscall `close`, un proceso puede enviar más bytes al dispositivo (nuevos llamados a la syscall `write`). Al hacer `close`, la canción continuará siendo reproducida hasta su finalización.

Se cuenta con la siguiente API para operar con el dispositivo de E/S, que deberá programar en un driver.

int open(int device_id) int close(int device_id)	Abre el dispositivo. Cierra el dispositivo.
int write(int device_id, char* data, int cantidad)	Escribe el valor en el dispositivo device_id.
int driver_init() int driver_remove()	Durante la carga del SO. Durante la descarga del SO.

Todas las operaciones retornan la constante `IO_OK` si fueron exitosas o la constante `IO_ERROR` si ocurrió algún error. Además en el kernel se cuenta con la estructura de datos `current`, a partir de la cual puede obtenerse el *PID* del proceso en ejecución llamando a la función `pid = task_pid_nr(current)`.

Para la programación de un driver, se dispone de las siguientes syscalls:

void OUT(int IO_address, char data)	Escribe data en el registro de E/S (1 único byte).
void IN(int IO_address)	Retorna el valor almacenado en el registro de E/S.
int request_irq(int irq, void* handler)	Permite asociar el procedimiento handler a la interrupción irq que es llamado por el equipo de música al terminar de escribir el byte enviado. Retorna `IRQ_ERROR` si ya está asociada a otro handler.
int free_irq(int irq)	Libera la interrupción irq del procedimiento asociado.

Se tienen a su vez las siguientes direcciones y valores:

- `EQUIPO_IEQ_X`, con X de 1 a 3, el valor del IRQ del equipo X.
- `ADDRESS_X`. Es la dirección a la que hay que escribir para enviar un byte al equipo.
- `CONTROL_X`. Leyendo este registro (con X, 1 a 3) se puede ver si el equipo X está libre (devuelve 0) u ocupado (devuelve 1).

Pueden suponer que los equipos no se utilizan antes de cargar el driver (pero podrían no estar listos al inicio). Además cuentan con el tipo de datos cola, con las funciones típicas (push, pop, peek, y length) y su crecimiento y uso de memoria es automático.

Tengan en cuenta que las llamadas a `write` pueden bloquear un tiempo (encolado de operaciones o alguna operación `OUT`) pero no durante la escritura (`OUT`) completa del buffer enviado a `write`.

Solution:

```
struct device_t {
  int pid;
  bool free;
  cola queue;
  mutex mtx;
```

```
} devices[3];

bool in_range(int device_id) {
  return 0 <= device_id && device_id < 3;
}

bool match_id(int pid, int device_id) { //U: El proceso controla al device pedido
  return in_range(device_id) && !devices[device_id].free && devices[device_id].pid == pid;
}

int driver_init() {
  //NOTA: Falta pedir major/minor y registrar. Pregunte y me dijeron que no es necesario
  bool done = false;
  while (!done) { //A: Estan todos iniciados. Busy waiting
    done |= IN(CONTROL_1) == 0 && IN(CONTROL_2) == 0 && IN(CONTROL_3) == 0;
    //NOTA: Pregunte y me dijeron que CONTROL_X es para esto
  }
  devices[X].free = true;
  if (request_irq(EQUIPO_IRQ_X, handler_X) == IRQ_ERROR) return IO_ERROR; //NOTA: handler_X
    definido abajo
  //NOTA: Repetir ambas lineas para X con 1, 2, y 3

  return IO_OK;
}

int driver_remove() {
  //NOTA: Falta liberar major/minor/etc.
  free_irq(EQUIPO_IRQ_X); //NOTA: Repetir para los 3

  return IO_OK;
}

int open(int device_id) {
  /* CORRECCION:
   * Durante el parcial me dijeron que device_id identificaba a uno de los 3 dispositivos, pero
    al corregir el parcial me pusieron "el device_id es el mismo para todos". El codigo esta
    armado sobre lo que entendi durante el examen.
   */

  if (!in_range(device_id)) return IO_ERROR;

  devices[device_id].mtx.lock(); //A: Espero a que este libre //TODO: CORRECCION
  devices[device_id].free = false;
  devices[device_id].pid = task_pid_nr(current);

  return IO_OK;
}

int close(int device_id) {
  if (!match_id(task_pid_nr(current), device_id)) return IO_ERROR; //A: No lo controla

  devices[device_id].free = true;
  devices[device_id].mtx.unlock(); //TODO: CORRECCION

  return IO_OK;
}

int write(int device_id, char *data, int cantidad) {
  if (!match_id(task_pid_nr(current), device_id)) return IO_ERROR; //A: No lo controla

  char *buffer = kmalloc(cantidad);
  copy_from_user(buffer, data, cantid);
  for (int i = 0; i < cantidad; i++)
    devices[device_id].cola.push_back(buffer[i]);

  if (IN(CONTROL_X) == 0) //A: No esta leyendo //NOTA: Reemplazar X para el device_id
    mandar_byte(device_id); //A: Lo arranco, hace un OUT //NOTA: Definido abajo

  return IO_OK;
}

```

```
74 void mandar_byte(int device_id) {
75   if (devices[device_id].cola.empty()) return; //A: Termino
76
77   char dato = devices[device_id].cola.peek(1);
78   devices[device_id].cola.pop_front();
79   OUT(ADDRESS_X, dato); //NOTA: Reemplazar X para cada device_id
80 }
81
82 void handler_X() { //NOTA: 3 copias, una por cada equipo
83   mandar_byte(X); //NOTA: X, el equipo correspondiente
84 }
```

Question 3: Sistemas Distribuidos (20 puntos)

En una variante descentralizada del protocolo Two Phase Commit, los participantes se comunican directamente uno con otro en vez de indirectamente con un coordinador

- Fase 1: El coordinador manda su voto a todos los participantes.
- Fase 2: Si el coordinador vota que no, los participantes abortan su transacción. Si vota que si, cada participante manda su voto al coordinador y al resto de participantes donde cada uno decide sobre el resultado acorde a el voto que le llega y lleva acabo el procedimiento.

a) ¿Qué ventajas y desventajas encuentra a esta variante con respecto a la variante centralizada? Hablar con respecto a la cantidad de mensajes, tolerancia a fallos, etc.

b) ¿En qué casos usaría cada versión del protocolo?

Solution:

a) Esta variante es más resistente a que se caigan nodos, en específico el coordinador, ya que una vez que este vote no se lo necesita para proceder.
Además es resistente a la pérdida de mensajes (si no recibí del líder pero sí de otro nodo, sé que hay que procesar algo).
Pero esto se paga teniendo que enviar más mensajes, pasa a ser de $O(n)$ a $O(n^2)$.
E igualmente depende del coordinador para recibir y responder pedidos.

b) Lo usaría si hay pocos nodos (la cantidad de mensajes no aumenta tanto) o si es común que se caigan nodos (en específico el coordinador).
En ambos casos los beneficios son más que los costos.

Question 4: Seguridad (25 puntos)

Dado el código a continuación:

```
1  void imprimir_habilitado(const char *nombre_usuario, const char *clave, const char *
       imprimir, int tam_imprimir) {
2    char *cmd = malloc(tam_imprimir + 5*sizeof(char));
3    if (cmd == NULL) exit(1);
4    if (usuario_habilitado("/etc/shadow", nombre_usuario, clave)) {
5      snprintf(cmd, tam_imprimir+4, "echo %s", imprimir);
6      system(cmd);
7    } else {
8      printf("El usuario o clave indicados son incorrectos.");
9      assert(-1);
10   }
11 }
```

El objetivo de la función es imprimir por pantalla el texto enviado como parámetro por el usuario, siempre y cuando el nombre de usuario y clave provistos por dicho usuario sean correctos.

Para esto se cuenta con la función `usuario_habilitado` que se sabe funciona correctamente y no cuenta con problemas de seguridad. La misma utiliza strings terminados en caracter nulo (`\0`) y lee el archivo provisto de contraseñas (encriptadas) de todos los usuarios del sistema, que puede ser sólo leído por `root`, devolviendo un booleano indicando si el usuario y la clave ingresados se encuentran en dicho archivo.

a) Indique si es necesario que el programa corra con algún nivel específico de permisos. Justifique en qué líneas y porqué.

b) Indique dos problemas de seguridad que podrían surgir (hint: tenga en cuenta el item anterior).

c) Indique alguna manera (valuación de los parámetros) de poder explotar cada una de las vulnerabilidades mencionadas.

d) Indique el impacto de las vulnerabilidades mencionadas, analizándolas según los tres requisitos fundamentales de la seguridad de la información.

e) Para cada vulnerabilidad, proponga una solución cómo modificaría el código en caso de corresponder.

Solution:

a) Como `/etc/shadow` sólo puede ser leído por el root. El programa tiene que correr con sus permisos.

Note:-
La consigna pedía "en qué líneas"

b y c)
-) Al correr como root, si mando "HACKED; /bin/bash" en imprimir (con el tamaño correspondiente) y mi usuario y clave (que yo conozco, por lo que son correctos). Puedo correr bash como root.

-) Como la llamada a echo no aclara el path, puedo agregar mi propio "echo" al principio del PATH y correr cualquier programa como root.

En ambos casos hay escalada de privilegios.

d) Las vulnerabilidades me dan control sobre el sistema entero. Puedo ver y modificar los datos de cualquier usuario y "fingir" ser cualquiera de ellos.

Note:-
Incompleto

e)
-) Para la primera vulnerabilidad, que es un format string, me fijaría que no mande caracteres peligrosos como ";" con una blocklist.

-) Para la segundaría, aclararía el path completo "/bin/echo".