Sistemas Operativos

Departamento de Computación – FCEyN – UBA

Primer cuatrimestre de 2022

Nombre y apellido: _____

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

1	2	3	4	Nota

Recuperatorio Segundo parcial – 12 de julio - 1er cuatrimestre de 2022

- 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.
- **Cada código o pseudocódigo debe estar bien explicado y justificado en castellano. ¡Obligatorio!**
- 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. Sistemas de Archivos (25 puntos)

Considerando un sistema de archivos **Ext2**, se pide implementar una función *find_file_less_size(char * dir, int min_bytes, char * arch_nombre)*, que tome el *path* de un directorio, una cantidad de bytes, y un nombre, y devuelva una lista con todos los archivos que tengan un tamaño menor a *min_bytes*, cuyo nombre sea *arch_nombre*, a partir de *dir* (inclusive). La búsqueda debe también considerar los directorios internos al directorio dado. También se pide que diseñe la lista que retornará la función para que contenga la información de cada tipo de archivo encontrado (regular, binario, bloques, directorio, etc), su última fecha de modificación, su tamaño, y su propietario. Considere que cuenta con el tamaño del bloque en la variable *BLOCK_SIZE*. Se cuenta con las siguientes estructuras para representar inodos y entradas de directorio:

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type;
    char name[];
};
struct Ext2FSInode {
    unsigned short mode; // info sobre el tipo de archivo y los permisos
    unsigned short uid; // id de usuario
    unsigned int size; //  en bytes
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime; // fecha ultima modificacion
    unsigned int dtime;
    unsigned short gid; // id de grupo
    unsigned short links_count; // cantidad de enlaces al archivo
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
};
```

Se cuenta también con las siguientes funciones:

```
char * nombre_propietario(unsigned short id_usuario)
// dado el id de un usuario, devuelve el nombre del usuario.
struct Ext2FSInode * Ext2FS::inode_for_path(const char * path)
// dado un path, devuelve su inodo
void Ext2FS::read_block(unsigned int block_address, unsigned char * buffer)
// dada una direccion de bloque y un buffer, carga el bloque indicado en el // buffer
unsigned int Ext2FS::get_block_address(struct Ext2FSInode * inode, unsigned int block_number)
// dados un inodo y un numero de bloque, recorre el inodo buscando la
// direccion del bloque de datos indicado
struct Ext2FSInode * Ext2FS::load_inode(unsigned int inode_number)
// dado un numero de inodo, busca el inodo en el grupo y lo devuelve
```

Ejercicio 2. Drivers (25 puntos)

Se requiere diseñar un sistema de seguridad compuesto por una cámara, un sensor de movimiento, y un software de control. El requisito principal es que cuando el sensor detecta movimiento, el sistema responda con el encendido de la cámara por una cantidad de tiempo T, si detecta movimiento antes de que termine el tiempo T, la política a seguir es la de esperar un tiempo T desde esta última detección.

a) Proponga un diseño para este sistema de seguridad, donde debe indicar cuántos y qué tipo de registros tendría cada dispositivo, e indicando también para qué se utilizarían. También debe indicar y justificar el tipo de interacción: interrupciones, *polling, dma*, etc. con cada dispositivo. Puede usar uno o más tipos de interacción diferentes para cada dispositivo.

b) Una vez que tenga el diseño, debe escribir los dos *drivers* correspondientes (las operaciones que considere necesarias: *open, write, read*, etc), para poder cumplir el objetivo planteado. Tenga en cuenta que el software de control correrá a nivel de usuario, y podrá tener interacción con los drivers. No es necesario que escriba las especificaciones del software, pero sí se debe indicar cómo interactuará con los *drivers*. Cada operación usada debe estar justificada.

c) Explique cómo se genera la interacción a nivel del código entre los *drivers* que propuso.

Ejercicio 3. Seguridad (25 puntos)

El siguiente fragmento de código en C presenta al menos una vulnerabilidad grave. Se deberá hacer un análisis del mismo, se recomienda:

- Analice el mismo explicando qué hace el código, qué función cumple cada variable, y cuál sería el vector de ataque y el nombre de la vulnerabilidad. Justifique e indique cualquier detalle relevante.
- Explique las potenciales formas de impacto de los distintos ataques posibles, ejemplificando inputs y outputs.
- Dentro de los ataques posibles, elija el que considere **más grave** e indique con qué valores deberá ser ejecutado el programa para vulnerarlo (ver debajo ejemplo de una línea de ejecución). En caso de requerir un payload complejo, indicar la composición de cada parte del mismo. Indique cuáles serían las consecuencias del ataque elegido.
- En caso de requerir conocer algún dato o valor del programa para realizar el ataque, indique si es posible obtenerlo de algún modo.
- Explique si existen formas de mitigar la vulnerabilidad.

```
char MAX_SIZE = 127;
unsigned char buffer[128];
char *format = "%s %d %s\n";
char* algo_asi_si(char *cadena){
  scanf("%127s", buffer);
  printf(format, buffer, MAX_SIZE, cadena);
  return cadena;
}
char* algo_asi_no(char *cadena){
  if(strlen(cadena) > MAX_SIZE) exit(EXIT_FAILURE);
  sprintf(buffer, format, "echo" , atoi(cadena), "asi si?\n");
  system(buffer);
  return cadena;
}
int main(int argc, char **argv){
  setuid(0);
  printf(algo_asi_no(algo_asi_si(argv[1])));
}

$ echo "hola" | ./a.out "chau!"
```

Ejercicio 4. Sistemas Distribuidos (25 puntos)

Considere el siguiente algoritmo de consenso para sistemas distribuidos, donde todos los nodos están conectados entre sí:

Paso 1: Cuando un nodo se da cuenta de que el líder se ha bloqueado, envía un mensaje "ELECCIÓN" a todos los nodos que tienen id superior al suyo. En caso que no reciba respuestas de los nodos superiores, asume que es el líder y envía el mensaje "COORDINADOR", junto con su id a todos los nodos, incluidos los nodos con id inferiores.

Paso 2: Cuando un nodo con un id superior recibe un mensaje de "ELECCIÓN" de un nodo inferior, responde con su id de nodo junto con la respuesta "OK".

Paso 3: Cuando un nodo recibe la respuesta "OK" de muchos nodos (con sus respectivos ids), encuentra el id más alto, max(id), y envía mensajes "COORDINADOR" junto con el maximo id de todos los nodos. De esta forma se anuncia al nuevo líder.

Para asegurarse de que el líder recién elegido no se haya bloqueado, el nodo que envía el mensaje "COORDINADOR" junto con el id, espera un tiempo aleatorio para recibir la respuesta "ACEPTAR" del nuevo líder. Si no recibe respuesta de "ACEPTAR", el nodo que envía el mensaje "COORDINADOR" inicia de nuevo todos los pasos. Estos pasos continúan hasta que un nuevo líder responde con la respuesta "ACEPTAR" o el nodo sigue el **Paso 1** donde anuncia al nuevo líder.

Se pide:

a) Indicar si se logra elegir un líder al usar este algoritmo. Justifique su respuesta .

b) Indicar a cuál de los algoritmos vistos en clase se asemeja más, indicando sus ventajas y/o desventajas en comparación al algoritmo seleccionado. Justifique su respuesta.