- 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, LU y número de orden.
- **Cada código o pseudocódigo debe estar bien explicado y justificado en castellano. ¡Obligatorio!**
- Toda suposición o decisión que tome deberá justificarla adecuadamente. Si la misma no es correcta o no se condice con el enunciado no será tomada como válida y será corregida acorde.
- 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)

Suponiendo un sistema de archivos `Ext2`, se solicita escribir en pseudocódigo la función `void my_grep(char* palabra, char* path)`. Esta función debe recibir como parámetros una palabra y una ruta a un directorio, y debe imprimir por pantalla las líneas de los archivos regulares (solamente) que contienen esa palabra. La búsqueda debe realizarse también en cada subdirectorio. La salida debe tener el siguiente formato:

```
<Nombre_archivo>, línea de texto
```

Considerar al caracter `'\0'` como fin de línea, y al caracter `EOF` como fin de archivo. Se cuenta con las siguientes estructuras de `Ext2FS`:

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type; // 0x1: regular file, 0x2: directory
    char name[];
};

struct Ext2FSInode {
    unsigned short mode;
    unsigned short uid;
    unsigned int size; // tamaño en bytes
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime;
    unsigned int dtime;
    unsigned short gid;
    unsigned short links_count;
    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 la constante `BLOCK_SIZE` para el tamaño de bloque, y las siguientes funciones:

- `struct Ext2FSInode * Ext2FS::inode_for_path(const char * path)`: que dado un path, devuelve su inodo

- `void Ext2FS::read_block(unsigned int block_address, unsigned char * buffer)`: que lee de disco el bloque de dirección `block_address` y lo coloca en buffer.
- `unsiged int Ext2FS::get_block_address(struct Ext2FSInode * inode, unsigned int block_number)`: que devuelve la dirección del bloque de datos (`block_number`) del inodo (`inode`).
- `struct Ext2FSInode * Ext2FS::load_inode(unsigned int inode_number)`: que devuelve el inodo número `inode_number`.
- `unsigned char * get_line(unsigned char * text)`: que dado un texto, devuelve su contenido desde el inicio hasta el primer caracter de salto de línea inclusive.
- `bool find(unsigned char * line, unsigned char * word)`: que devuelve true si la palabra `word` está en `line`, false en caso contrario.
- `size_t strlen ( const char * str )`: que devuelve la longitud del string `str`.

Ejercicio 2. Sistema de E/S - Drivers (25 puntos)

Considere un sistema de refrigeración de una sala de reuniones compuesto por dos dispositivos conectados a una computadora que ejecuta un sistema operativo Linux: (i) un ventilador y (ii) un sensor de temperatura ambiente con cronómetro incorporado. Cada dispositivo debe ser manejado por un driver independiente.

Se pide:

a) Proponer un diseño, en donde debe indicar cuántos y qué tipo de registros tendría cada dispositivo, e indicando también para qué se utilizarían. Indicar y justificar el tipo de interacción con cada dispositivo (interrupciones, polling, dma, etc.).

b) Una vez que tenga el diseño, escribir los drivers correspondientes a ventilador y temperatura, de modo tal que cumplan los siguientes objetivos:

   - Al iniciarse, la aplicación de usuario deberá recibir tres parámetros de configuración por entrada estándar: un umbral de temperatura mínima, un umbral de temperatura máxima, y un tiempo T (todos enteros).
   - El sensor de temperatura deberá poder informar con un número entero el promedio de la temperatura de los últimos T segundos.
   - Si la temperatura promedio de los últimos T segundos se encuentra por debajo del valor mínimo, el ventilador deberá apagarse.
   - Si la temperatura promedio de los últimos T segundos se encuentra por arriba del valor máximo, el ventilador deberá encenderse.
   - Cualquier temperatura que se encuentre entre la mínima y la máxima se deberá considerar dentro del rango normal de refrigeración, y no deberá tener ningún impacto en el estado del ventilador.
   - Para reducir el consumo energético del sistema, el ventilador solamente deberá cambiar de estado cuando la temperatura promedio se encuentre fuera del rango normal (menor a la temperatura mínima o mayor a la máxima).
   - No está permitido realizar sleep u otras operaciones similares.
   - Para cada driver se deberá implementar en código C las funciones mínimas necesarias para poder cumplir el objetivo planteado. El código deberá ser sintácticamente válido y respetar las buenas prácticas mencionadas durante las clases. Por simplicidad, siempre que esto no impacte en la solución, se permitirá omitir el chequeo de errores. Todas las decisiones implementativas deberán estar debidamente justificadas.

c) Además, se solicita explicar el funcionamiento de la aplicación de usuario, y su interacción con los drivers utilizando pseudocódigo lo más similar a C posible. Tenga en cuenta que la aplicación de control correrá a nivel de usuario. Indicar con código C cómo interactuará el software de control con los drivers. Cada operación usada debe estar justificada.

Implementar cualquier función o estructura adicional que considere necesaria (tener en cuenta que en el kernel no existe la libc). Se podrán utilizar además las siguientes funciones vistas en la práctica:

```
unsigned long copy_from_user(char *to, char *from, uint size)
unsigned long copy_to_user(char *to, char *from, uint size)
int IN (int regnum)
void OUT(int regnum, int value)
void *kmalloc(uint size)
void kfree(void *buf)
void request_irq(int irqnum, void *handler)
void free_irq(int irqnum)
void sema_init(semaphore *sem, int value)
void sema_wait(semaphore *sem)
void sema_signal(semaphore *sem)
void mem_map(void *source, void *dest, int size)
void mem_unmap(void *source)
```

Ejercicio 3. Sistemas distribuidos: (25 puntos)

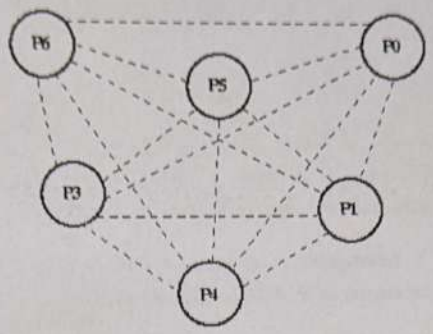

Figura 1: Sistemas distribuidos

Comenzamos con 7 procesos, todos conectados directamente entre sí, con ids del 0 al 6. El Proceso 6 es el líder, ya que tiene el número más alto. Si el Proceso 6 falla y el Proceso 3 se da cuenta de que el Proceso 6 no responde:

a) Explique cómo se utilizará el **algoritmo de Bully** para elegir un nuevo líder. Mencione la cantidad total de mensajes que se envían.

b) Explique brevemente cómo sería la elección de líder usando **Flood Max**. Mencione la cantidad total de mensajes que se envían.

Ejercicio 4. Seguridad: (25 puntos)

Se cuenta con un código el cual verifica si una clave es válida, permitiendo acceder al sistema con privilegios de administrador en caso afirmativo. Para el siguiente código se pide:

a) Explicar qué vulnerabilidad tiene, y cómo serían los valores exactos de cada byte del input de un *exploit* que permita realizar una ejecución de código arbitrario con escalamiento de privilegios. Aclarar las funciones utilizadas y justificar con un diagrama de las posiciones de memoria afectadas que incluya sus significados y sus valores antes y después de aplicar el *exploit*. ¿Cómo generaría dicho input, y cómo ejecutaría el programa en cuestión? (18p)

b) Proponer una corrección para el código en C que solucione la vulnerabilidad, detallando qué líneas modificaría, y por qué. Proponga además dos formas adicionales de mitigar la vulnerabilidad que no requieran modificar el código, explicando detalladamente su funcionamiento, y mostrando cómo dificultarían el ataque. (7p)

```
extern bool clave_es_valida(const char* clave);
extern void acceder_al_sistema();

void convertir_a_minuscula(char* buffer) {
  do {
    *buffer = tolower(*buffer);
  } while (*(buffer++) != '\0');
}

char* copiar_en_minuscula(const char* str) {
  char buffer[16];
  strcpy(buffer, str);
  convertir_a_minuscula(buffer);
  return strdup(buffer);
}

int main(int argc, const char* argv[]) {
  char* clave = copiar_en_minuscula(argv[1]);
  if (clave_es_valida(clave)) {
    acceder_al_sistema();
  }
  return 0;
}
```

Ayuda 1: Asuma que `clave_es_valida()` y `acceder_al_sistema()` son funciones seguras.

Ayuda 2:

```
Dump of assembler code for function main:
   0x0000555555555080: sub    rsp,0x8
   0x0000555555555084: mov    rdi,QWORD PTR [rsi+0x8]
   0x0000555555555088: call   0x5555555551d0 <copiar_en_minuscula>
   0x000055555555508d: mov    rdi,rax
   0x0000555555555090: call   0x555555555210 <clave_es_valida>
   0x0000555555555095: test   al,al
   0x0000555555555097: jne    0x5555555550a0 <main+32>
   0x0000555555555099: xor    eax,eax
   0x000055555555509b: add    rsp,0x8
   0x000055555555509f: ret
   0x00005555555550a0: xor    eax,eax
   0x00005555555550a2: call   0x555555555220 <acceder_al_sistema>
   0x00005555555550a7: jmp    0x555555555099 <main+25>
```