Sistemas Operativos

Departamento de Computación – FCEyN – UBA
Segundo cuatrimestre de 2023

Nombre y apellido: [redacted]
[redacted] L.U.: [redacted] Cant. hojas: 8

1	2	3	4	Nota
24	24	25	24	97

A

Segundo parcial – 28/11 - Segundo cuatrimestre de 2023

- 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)

En un típico sistema tipo UNIX contamos con el comando `ls` que nos permite listar el contenido de un directorio. Si se utiliza la opción `-R` o `--recursive`, el comando lista recursivamente el contenido de los subdirectorios. Por ejemplo, si se tiene la siguiente estructura de directorio:

```
/ruta/a/un/directorio/
| archivo1
| archivo2
| un_subdirectorio/
|    | archivo5
|    | archivo6
| archivo3
| otro_subdirectorio/
|    | tambien_subdirectorio/
|         | archivo8
|         | archivo9
|    | archivo7
| archivo4
```

Obtendríamos un resultado parecido al siguiente, al ejecutar el comando `ls -R /ruta/a/un/directorio/`:

```
archivo1
archivo2
un_subdirectorio/
archivo5
archivo6
archivo3
otro_subdirectorio/
tambien_subdirectorio/
archivo8
archivo9
archivo7
archivo4
```

a) Se solicita escribir el pseudocódigo de la función `mi_ls_r(char * path)` que deberá tener el mismo comportamiento que `ls -R path`. La función provista debe funcionar en un sistema de archivos tipo FAT.

Asumir que se tiene la constante `BLOCK_SIZE` con el tamaño de bloque, la constante `FAT_DIR_ENTRY_SIZE` con el tamaño de entrada de directorio y las siguientes funciones y estructuras ya implementadas:

```
// Ver la FAT como un arreglo
// indice i corresponde al bloque de disco i
// FAT[i] indica el bloque siguiente al bloque i.
unsigned int[] FAT;

// entrada de directorio
```

```
struct FATDirEntry{
    char filename[8];
    char extension[3];
    char attribute; // 0x1: archive, 0x2: directory
    ...
    unsigned short starting_cluster; // primer bloque
    unsigned int size;
};

//dado un path, devuelve su primer bloque
unsigned int FATFS::get_first_block_from_path(const char * path);
```

Se puede considerar que `BLOCK_SIZE` es múltiplo de `FAT_DIR_ENTRY_SIZE`.

b) Explicar en lenguaje natural las diferencias entre un sistema de archivos FAT y uno Ext2 haciendo foco especialmente en el manejo de directorios, y detallar cómo esas diferenicas cambian la función implementada en el inciso anterior para un sistema de archivos tipo Ext2.

Ejercicio 2. Drivers (25 puntos)

El doctor Augusto Losano, experto en tecnología aplicada a logística, empezó a trabajar en BanderEnvios, una nueva empresa de distribución de paquetes. Inspirados en el diseño de otras empresas, se está construyendo una central inteligente de transferencias cerca de Ciudad Universitaria. Diferentes camiones recogen los paquetes de sus clientes, llegan a la central y descargan su contenido. Estos paquetes son puestos en cintas transportadoras, donde diferentes personas, al final de su recorrido, agarran los paquetes y los clasifican.

La cinta transportadora tiene un motor bastante simple y discreto (marca Sumoza Corporation) que tiene la capacidad de regular la velocidad a la cual se mueve la cinta. Cuenta con 3 modos: 0 para nula velocidad, 1 para baja velocidad y 2 para velocidad rápida. También se dispone de una balanza que mide cuánto peso está soportando la cinta transportadora actualmente.

Un detalle que notó el doctor Augusto, es que si la cinta contiene demasiado peso, se debe regular la velocidad del motor para no forzarlo demasiado.

Además, desde la última invasión (hace unos meses) de anfibios saboteadores provenientes de los pabellones de la facultad, muchos dispositivos sufrieron daños, y la balanza, cada cierto tiempo, devuelve valores incorrectos. La empresa tiene un dispositivo viejo (un beeper) que avisa a los técnicos cuando se produce un problema para que vengan a revisarlo lo antes posible.

El doctor Augusto quiere poder coordinar los diferentes dispositivos mediante software con una computadora con Linux y nos encargó a nosotros la tarea.

Nos pide diseñar un driver para cada dispositivo (balanza, beeper, motor) que cumpla las siguiente características:

- Se tiene que leer por entrada estándar, valores enteros positivos, T_1, T_2.
- Si la balanza detecta que el peso es mayor a T_2, entonces hay que detener la cinta porque no soporta más peso. Como esta situación es problemática, hay que avisar por el beeper con el valor EXCEEDED_WEIGHT.
- Si el peso está entre T_1 y T_2, el motor tiene que ir a una velocidad lenta.
- Si el peso es menor que T_1, el motor tiene que ir a una velocidad rápida.
- Si la balanza da 0, se tiene que parar el motor ya que no hay paquetes que transportar.
- Si la balanza da un resultado negativo, significa que hubo un inconveniente. Hay que volver a leer 10 veces el dispositivo, una cada 100ms. Asumir que la computadora cuenta con timer interno que podemos usar con interrupciones. Si en alguna de las 10 veces arroja un resultado válido, lo tomamos como el valor correcto. Si en ningún caso nos devuelve algo válido, hay que avisar por el beeper el valor BROKEN_B.

El acceso a los dispositivos es lento y consume mucha energía, así que se quiere minimizar los accesos. No podemos usar sleep() ni funciones similares.

Se pide:

a) Proponer un diseño, indicando los registros que tendría cada dispositivo, junto con su utilidad. Indicar y justificar el tipo de interacción que deberá soportar cada dispositivo (interrupciones, polling, etc.). Detallar cuántos drivers deberán ser implementados, qué dispositivo/s manejará cada driver, y qué funciones soportarán los mismos, describiendo su comportamiento en lenguaje natural.

b) A partir del diseño del punto anterior, escribir los drivers correspondientes y el software de usuario. Escribir en pseudocódigo (estilo C) las funciones mínimas necesarias para poder satisfacer 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 implementadas deberán estar debidamente justificadas.

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)

Se tiene una conexión de nodos de un sistema distribuido en donde se requiere aplicar el algoritmo de Paxos. El conjunto de nodos se fraccionó en dos partes, quedando el nodo C aislado, A y B en un conjunto y D y E, en otro. El siguiente esquema muestra el estado del sistema para ese momento específico de separación.

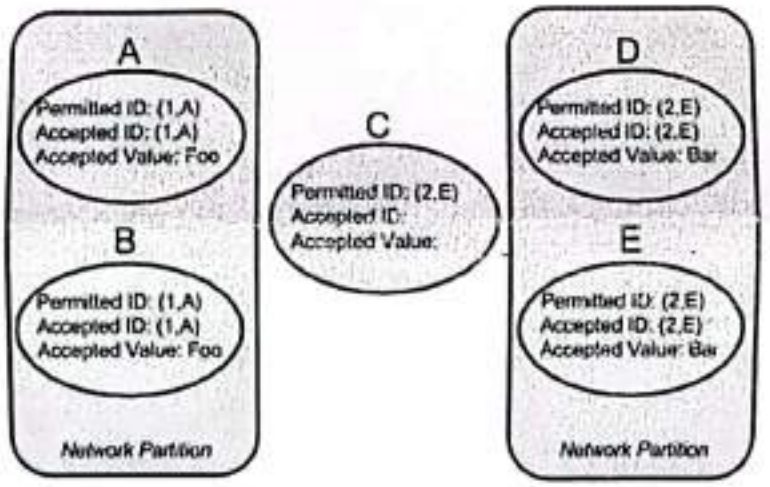


Figura 1: Esquema de conexión de nodos en un sistema distribuido

a) Se solicita que indique cuáles son los pasos que se realizarían al aplicar el algoritmo de Paxos si se restablece la conexión entre los nodos C, D y E, para que estos nodos lleguen a un consenso. ¿Cuál es el resultado final del consenso?

b) Si después se restablece la comunicación entre A, B y C ¿Cuáles son los pasos que sigue el algoritmo de Paxos? ¿Es posible elegir Foo?

Ejercicio 4. Seguridad: (25 puntos)
Se ha detectado una vulnerabilidad crítica de "Path Traversal" en la API de R.R.H.H, comparable a la vulnerabilidad conocida como "StringFormat", la cual permite comportamientos no autorizados mediante la manipulación inadecuada de las entradas de datos. Al revisar los registros del sistema, se descubrió que algunos usuarios eran capaces de acceder a información confidencial de otros usuarios, incluyendo detalles como sus salarios y direcciones personales. Esto se lograba al modificar el path del `documento` como se muestra a continuación:

```
1       GET /api/leer_documento?usuario_id=id4&documento=../id3/datos_secretos.txt
```

Lo que podemos ver en este *request*, es que hay dos claves (llamadas *query params*), *documento* y *usuario_id*, con valores `id4` y `../id3/datos_secretos.txt` respectivamente

Se solicita una revisión exhaustiva del código para identificar la presencia de otras posibles vulnerabilidades:

```
const std::string CARACTERES_PROHIBIDOS = ":&\";[$-]|-^?{}=$%+*!@",<>\n\t\r";

void sanitizarRecursivo(const std::string &entrada) {
    char buffer[20];
    size_t posicion = entrada.find_first_of(CARACTERES_PROHIBIDOS);

    // Si encontramos un carácter prohibido se elimina, npos indica -1
    if (posicion != std::string::npos) {
        strcpy(buffer, entrada.substr(0, posicion + 1).c_str());
        printf("[LOG] Caracter prohibido encontrado: %c en %s\n", entrada[posicion], buffer);
        entrada.erase(posicion, 1);
        sanitizarRecursivo(entrada.substr(posicion + 1));
    }
}

int main() {
    httplib::Server svr;

    char MAX_LENGHT = 100;

    svr.Get("/api/leer_documento", [](const httplib::Request& req, httplib::Response& res) {
        std::string documento = req.get_param_value("documento"); // Retorna input de la API para la clave documento
        std::string usuario_id = req.get_param_value("usuario_id"); // Idem id usuario

        checkValidId(usuario_id); // Revisa que el usuario exista y se puede tomar como una función segura

        sanitizarRecursivo(documento);

        boost::filesystem::path ruta = boost::filesystem::canonical(DIR_BASE + usuario_id + "/" + documento);
        if (ruta.string().find(DIR_BASE) == 0 && boost::filesystem::exists(ruta)) {
            std::ifstream archivo(ruta.string());
            res.set_content(std::string((std::istreambuf_iterator<char>(archivo)), std::istreambuf_iterator<char>()), "text/plain");
        // Responde al usuario con el contenido del archivo solicitado

            char mensaje[MAX_LENGHT];

            //Funciona como printf, pero guarda el contenido dentro de un buffer pasado como primer parametro
            snprintf(mensaje, sizeof(mensaje), "[LOG] Ruta accedida: %s\n", ruta.string().c_str());
            printf("%100s", mensaje);
        } else {
            res.set_content("Acceso denegado", "text/plain");
        }
    });

    svr.listen(SERVER_ADDRESS.c_str(), SERVER_PORT);
}
```

Puede asumir que las funciones **checkValidId** y **boost::filesystem** son seguras, así como las funciones de **httplib**. Además, `req.get_param_value("documento")` retorna un **string** con el input de la API para la key **documento**. Tampoco deben procuparse por el uso de **res.set_content(..)** y **std::ifstream archivo(..)**.

a) Indique los problemas de seguridad (si existen) que permiten un exploit del sistema. Indicar problemas no explotables restará puntos.

b) Proporcione ejemplos de entradas (inputs) que podrían ser utilizadas para explotar las vulnerabilidades identificadas en la API. Justificar.

c) Para cada vulnerabilidad existente indique el impacto analizándolas según los tres requisitos fundamentales de la seguridad de la información (incluir Path Traversal).

d) Para cada vulnerabilidad existente proponga una solución, indicando cómo modificaría el código en caso de corresponder (incluir Path Traversal). Justificar.

E)1 [redacted] 1 2023-11-28

24/25

```
void mi_ls_r(char * path) {
  uint block = get_first_block_from_path(path);
  uint dir_size = get_size(path);
  char* buffer = (char *) malloc(BLOCK_SIZE);
  while (block ≠ EOF) {
    read_block(block, buffer);
    struct FATDirEntry * entry = buffer;
    while (dir_size > 0) {
      if (entry->attribute == 0x1) { //archivo
        printf("%s.%s\n", entry->filename, entry->extension);
      } else if (entry->attribute == 0x2) { //directorio
        printf("%s/\n", entry->Filename);
        mi_ls_r(path + "/" + entry->Filename); //abuso de notación
      }
      entry++; //aritmética de punteros
      dir_size -= FAT_DIR_ENTRY_SIZE;
    }
    block = FAT[block];
  }
  free(buffer);
}
```

bien

ok, pero podrían usar starting_cluster

sizeof(struct FATDirEntry) = FAT_DIR_ENTRY_SIZE ok

Corregido: Hernán G.

La mayor diferencia entre FAT y Ext2 respecto al manejo de directorios es que las entradas (de directorio) de FAT tienen un tamaño fijo, lo cual limita el largo máximo de los nombres. En cambio Ext2 admite nombres arbitrariamente largos, por lo tanto las entradas tienen un largo variable.

Esto impacta sustancialmente cuando revisamos todas las entradas de un directorio. FAT es más simple, podemos revisar un bloque a la vez, ya que BLOCK_SIZE es múltiplo de FAT_DIR_ENTRY_SIZE. Es decir, nunca pasa que una entrada está partida entre 2 bloques.

No así en Ext2 (por el largo variable de las entradas). Esto requiere recorrer las entradas de Ext2 utilizando un buffer de tamaño BLOCK_SIZE * 2. Cargamos 2 bloques consecutivos y vamos revisando las entradas. Cuando detectamos que la entrada a revisar comienza en la parte alta del buffer (es decir en alguna posición del buffer $\geq$ BLOCK_SIZE) ya tenemos garantía que revisamos todo el bloque de la parte baja del buffer en su totalidad, y si había una entrada que comenzaba en el 1er bloque y terminaba en el 2do, ya la revisamos.

Ahora tenemos que copiar la parte alta del buffer a la parte baja, y cargar en la parte alta el siguiente bloque si hay. A su vez ajustar el puntero del siguiente entrada a revisar restándole BLOCK_SIZE.

Bien!

Excelente!

EJ2	[redacted]	2
		2023-11-28

Beeper

Registros:

- MSG

Al escribir en MSG el beeper hace un sonido distinto según el mensaje. Los valores posibles son: EXCEEDED_WEIGHT y BROKEN_B.

Motor

Registros:

- SPEED

Al escribir en SPEED el motor regula su velocidad según el valor escrito: STOP, SLOW=1, FAST=2. Si se escribe el mismo valor 2 veces ignora la segunda escritura.

Balanza

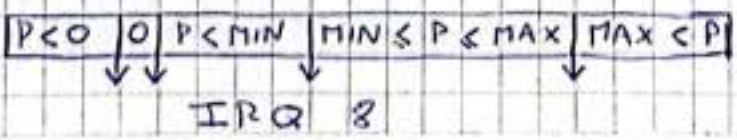
Registros:

- MIN
- MAX
- WEIGHT

Interrupciones:

- IRQ 8: el peso cambió

Los valores escritos en MIN y MAX controlan los umbrales de notificación por parte de la balanza. Cuando el peso cruza algún umbral, ó llega a 0, ó se vuelve negativo, la balanza envía la interrupción IRQ 8.



Programa de usuario

```
typedef struct {
  int min;
  int max;
} weight_config_t;

int main (int argc, char** argv) {
  int T1; int T2;
  cin >> T1 >> T2;
  weight_config_t wc = {.min = T1, .max = T2};
  int beeper = open("/dev/beeper");
  int motor  = open("/dev/motor");
  int balanza = open("/dev/balanza");
  write(balanza, &wc, sizeof(wc));

  int w; int speed; int msg;
```

```
while(1) {
  read(balanza, &w, sizeof(w));
  speed = STOP;
  msg = NULL;
  if (w < 0) {
    speed = STOP;
    msg = BROKEN_B;
  } else if (w == 0) {
    speed = STOP;
  } else if (w < wc.min) {
    speed = FAST;
  } else if (w <= wc.max) {
    speed = SLOW;
  } else {
    speed = STOP;
    msg = EXCEEDED_WEIGHT;
  }
  write(motor, &speed, sizeof(speed));
  if (msg != NULL) write(beeper, &msg, sizeof(msg));
 } // while
} // main
```

Driver: Beeper

```
uint driver_write(char *ubuffer, uint size) {
  int kbuffer;
  copy_from_user(&kbuffer, ubuffer, size);
  OUT(MSG, kbuffer);
  return size;
}
```

Driver: Motor

```
uint driver_write(char *ubuffer, uint size) {
  int kbuffer;
  copy_from_user(&kbuffer, ubuffer, size);
  OUT(SPEED, kbuffer);
  return size;
}
```

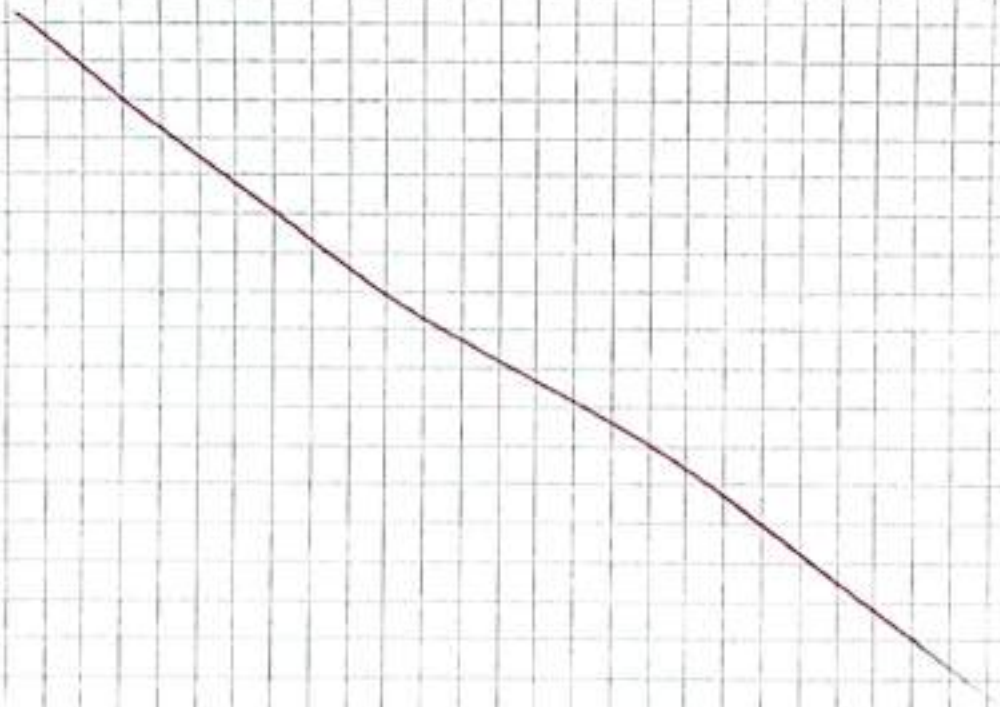
Driver. Balanza

```
semaphore new_weight;
semaphore timer;
bool retry = false
int retries = 10;
int weight_value;
bool first_read = true;

int driver_init() {
    sema_init(&new_weight, 0);
    sema_init(&timer, 0);
    request_irq(8, handle_change);
    request_irq(10, handle_timer); return 0;
}

int driver_remove() {
    free_irq(8);
    free_irq(10);
    return 0;
}

void handle_change() {
    sema_signal(&new_weight);
}
```

```
void handle_timer () {
  if(retry) {
    weight_value = max(weight_value, IN(WEIGHT));
    if (--retries == 0) {
      retry = false;
      sema_signal (&timer);
    }
  }
}

int driver_write (char *ubuffer, uint size) {
  weight_config_t wc;
  copy_from_user(&wc, ubuffer, size);
  OUT (MIN, wc.min);
  OUT(MAX, wc.max);
  return size;
}
```

el primer valor positivo habilita que devuelva

```
uint driver_read (char * ubuffer, uint size) {
  if (first_read) { → 처음 읽을 때 first_read
    first_read = False;
  }else {
    sema_wait(&new_weight);
  }
  weight_value = IN(WEIGHT);
  if (weight_value < 0) {
    retries = 10;
    retry = true;
    sema_wait(&timer);
  }
  copy_to_user(ubuffer, &new_weight, size);
  return size;
}
```



E) 3 [redacted] 6

2023-11-28

25

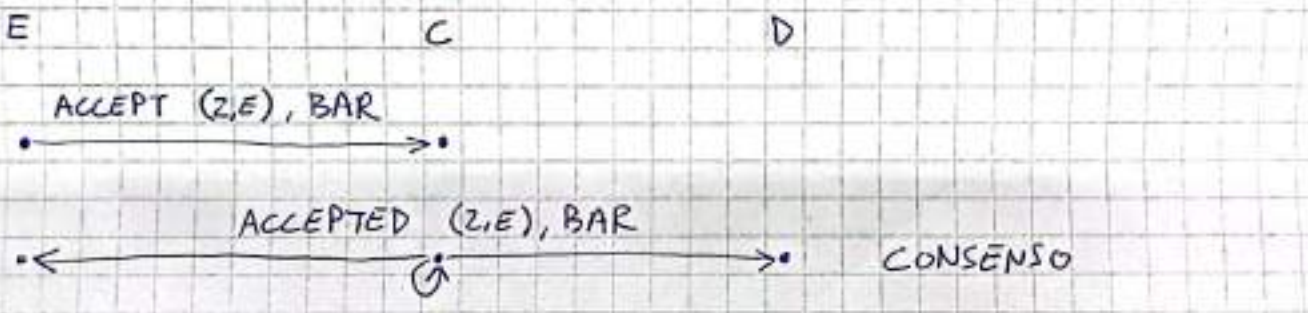
Suposiciones:

- Cuando los nodos se reconectan a la red, la infraestructura de comunicación (independiente a Paxos) notifica al resto de los nodos para que puedan reenviar mensajes que tuvieron timeout.
- Si bien todos los nodos cumplen los 3 roles: Proponedor, Aceptador y Aprendiz, los nodos A y E son los únicos proponedores en esta ronda de Paxos.
- Tomamos la definición de Paxos vista en clase en donde se buscan el consenso del mejor valor, no de la propuesta con mayor número de propuesta.

(6) Cuando C reconecta con D y E, el proponedor E sabe que C le respondió promise (2,E) pero nunca le llegó el accept (2,E), BAR, el cual se envió porque obtuvo el quorum de promesas para la propuesta (2,E) por parte de C, D y E (actuando en rol de Aceptador).

Entonces E reenvía accept (2,E), BAR a C el cual acepta, y al enviar accepted (2,E), BAR a los aprendizes C, D, E estos detectan el consenso pues ya tenían los accepted (2,E), BAR enviados por D y E previamente.

En este momento el consenso es sobre (2,E), BAR.



Cuando toda la red se reconecta, de forma similar al caso anterior, el nodo A detecta que puede hablarle a los nodos C, D y E para intentar avanzar su propuesta (1,A), Foo.

Se envían mensajes de prepare (1,A) los cuales son ignorados porque C, D y E ya prometieron (2,E) y (1,A) < (2,E). Más tarde A reintenta con un número de propuesta más alto hasta obtener promises de los nodos. para el número (3,A) > (2,E).

En este momento los promises de (3,A) vienen junto a las propuestas ya aceptadas por C, D, E. El nodo A tiene que elegir "la mejor" propuesta para aceptar. En clase vimos que tomamos el mejor valor. No obstante, Basic Paxos estipula que se toma el valor de la propuesta aceptada con el mayor número de propuesta. Vamos con la def. vista en clase.

A compara su valor Foo con BAR, y como Foo > BAR, procede a enviar mensajes de accept (3,A), Foo a todos los nodos y este es el valor que se termina consensuando.


A
B
C
D
E
PREPARE (1,A)
PREPARE (2,A)
PREPARE (3,A)
PROMISE (3,A) (1,A) FOO
PROMISE (3,A) (2,E) BAR
ACCEPT (3,A) FOO
ACCEPTED (3,A) FOO

E) 4	[redacted]	8
		2023-11-28

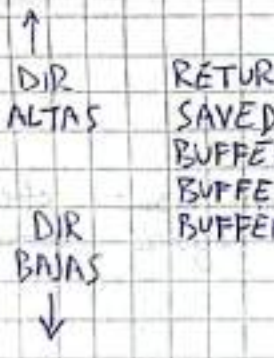
a)

5/5 En la línea 4 se declara un buffer de solo 20 bytes, el cual vive en el stack. Luego en la línea 9 se hace un strcpy al buffer que admite un buffer overflow si posicion > 20 y genera un DoS ✓

b)

4/4 Podemos construir un payload donde el query param documento tiene un caracter prohibido en alguna posicion > 20.
El exploit que podríamos hacer es pisar el return address lo cual podría generar un segfault y matar el proceso de la API. ✓

Asumiendo que el servidor tiene un procesador x86 de 64 bits, la pila se ve así:



Necesitamos un payload de 8·5 bytes/caracteres para pisar el return address con los últimos 8 bytes del payload.

Como es una API HTTP, debemos continuamente hacer requests para matar el proceso cada vez que revive. No tenemos acceso para colocar un return address específico y controlar a dónde retorna la función sanitizarRecursivo.

Podría usar un:

"a—á" + "¡" suficientemente largo como para

c) 6/6

Buffer overflow

Por lo dicho ya en b) se compromete la disponibilidad del servicio, es un ataque tipo DoS (denial of service). La integridad y confidencialidad no se ven comprometidas. ✓ ⊛

Path Traversal

Se compromete la confidencialidad porque el atacante puede acceder a archivos del servidor (solo los accesibles por el usuario que corre el proceso del servidor).
Como solo puede leer estos archivos, la integridad se preserva al igual que la disponibilidad. ✓

9/10)

Buffer overflow

Solo lograría el caracter solito en vez de mostrar parte del documento. Así se elimina por completo el buffer.

Path Traversal

Si, pero igual la idea es no usar strcopy sino una función segura como snprintf

Mejorar la lista de caracteres prohibidos e incluir el punto . y la barra /. ✓ considerar caso .txt

⊛ También se podría intentar un stack overflow pero posiblemente no sea suficiente el largo máximo permitido para las URLs.

Con hacer el smash del stack alcanza para terminar el programa.