

Condición de aprobación: 2 puntos de los Ejercicio 1, 1 punto del Ejercicio 2, y 1 punto del Ejercicio 3
No utilizar color rojo ó verde para la resolución del examen.

Ejercicio 1 (4 puntos)

Considere una función en ASM de 32 bits denominada CheckLong. A esta función se le pasan dos parámetros por la pila:

- El primer parámetro es la dirección de inicio de un vector de caracteres.
- El segundo parámetro es un valor entero que indica la cantidad de elementos del vector de caracteres.

La función CheckLong debe recorrer el vector hasta encontrar un cero, luego compara con el valor siguiente al cero (el cual debería ser la cantidad de elementos del vector) y verifica si la cuenta es correcta comparando este número con la cuenta realizada.

Checklong retorna la diferencia que hubo entre el valor calculado y el valor informado. Es decir, si son iguales retorna cero, si el valor calculado es mayor al informado retorna la diferencia en valor positivo, si el valor informado es mayor, retorna la diferencia en valor negativo.

Además se le pide a Ud. que está rutina tiene que quedar lista para poder ser llamada desde un programa en C

Se pide:

- Indique de qué manera recibirá CheckLong cada uno de los parámetros a la función.
- Programar utilizando lenguaje ASM el código de la función CheckLong usando el siguiente fragmento de programa:

```
section .data
msg: db "Hola Mundo",0
len: db 10
```

- Escriba un programa en C en un archivo llamado *principal.c* que solo llame a la función realizada en el punto b) y que imprima en pantalla si el chequeo de la longitud fue exitoso, o no.
- ¿Que consideraciones debe tener para poder compilar y linkeditar ambos archivos? ¿El archivo .asm y el archivo .c?

Ejercicio 2 (3 puntos)

Dado el siguiente programa que se compila para un procesador de 32 bits:

```
#include<stdio.h>

int calculo(int param1,int param2, char tipo) {

    int resul;

    if(tipo=='s')
        resul=param1+param2;
```

```

else
    resul=param1*param2;

return resul;

}

int main(void) {

    int valor1=7;
    int valor2=3;
    char operacion;
    printf("Ingrese el tipo de operación: "s" suma y "m" multiplicar : \n");
    scanf("%c",&operacion);
    printf("Resultado: %d\n",calculo(valor1, valor2, operacion));
    return 0;
}

```

- a. Explique con dibujos respectivos el contenido de la pila, durante toda la ejecución del programa.
 b. Muestre los registros del microprocesador que intervienen en los dibujos de la pila.

Ejercicio 3 (3 puntos)

Un alumno que todavía no cursó esta materia le pide a ud. que le explique:

- a) ¿Qué es una system call? ¿Quién la escribió? ¿Dónde se encuentran ubicadas?
- b) ¿Cómo se llama a una system call desde Assembler de Linux? Muestre con un ejemplo de llamada a read.
- c) ¿Qué es el número de ID que aparece en la clase teórica?

Como ud no recuerda de memoria el funcionamiento ingresa a un sistema operativo Linux y ejecuta los comandos “man 2 read” y obtiene la siguiente salida:

READ(2)	Linux Programmer's Manual	READ(2)
NAME	top	
read - read from a file descriptor		
SYNOPSIS	top	
#include <unistd.h>		
ssize_t read(int fd, void *buf, size_t count);		
DESCRIPTION	top	
read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.		
On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.		
If count is zero, read() may detect the errors described below. In the absence of any errors, or if read() does not check for errors, a read() with a count of 0 returns zero and has no other effects.		
According to POSIX.1, if count is greater than SSIZE_MAX, the result is implementation-defined; see NOTES for the upper limit on Linux.		

%eax	Name	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	int	-	-	-	-
2	sys_fork	struct pt_regs	-	-	-	-
3	sys_read	unsigned int	char *	size_t	-	-
4	sys_write	unsigned int	const char *	size_t	-	-
5	sys_open	const char *	int	int	-	-
6	sys_close	unsigned int	-	-	-	-

Ejercicio 1 (4 puntos)

Considera una función en ASM de 32 bits denominada CheckLong. A esta función se le pasan dos parámetros por la pila:

- El primer parámetro es la dirección de inicio de un vector de caracteres.
- El segundo parámetro es un valor entero que indica la cantidad de elementos del vector de caracteres.

La función CheckLong debe recorrer el vector hasta encontrar un cero, luego compara con el valor siguiente al cero (el cual debería ser la cantidad de elementos del vector) y verifica si la cuenta es correcta comparando este número con la cuenta realizada.

Checklong retorna la diferencia que hubo entre el valor calculado y el valor informado. Es decir, si son iguales retorna cero, si el valor calculado es mayor al informado retorna la diferencia en valor positivo, si el valor informado es mayor, retorna la diferencia en valor negativo.

$$\text{val_calc} - \text{val_inf}$$

Además se le pide a Ud. que está rutina tiene que quedar lista para poder ser llamada desde un programa en C

Se pide:

- Indique de qué manera recibirá CheckLong cada uno de los parámetros a la función.
- Programar utilizando lenguaje ASM el código de la función CheckLong usando el siguiente fragmento de programa:

```
section .data
msg: db "Hola Mundo",0
len: db 10
```

- Escriba un programa en C en un archivo llamado *principal.c* que solo llame a la función realizada en el punto b) y que imprima en pantalla si el chequeo de la longitud fue exitoso, o no.
- ¿Que consideraciones debe tener para poder compilar y linkeditar ambos archivos? ¿El archivo .asm y el archivo .c?

a) int checklong(char *str, int dim-teo)

ebp
retmain
LCS
dim-teo

recibe los parámetros a través del stack si el checklong implementa el creando del stack se acceden [ebp+8] en caso de la dirección de memoria de la cadena de caracteres.

[ebp + 12] sera la cantidad de caracteres informándolos que debemos chequear

b) GLOBAL check long

```
section .data
msg db "Hola mundo!",0
len db 10
```

```
section .text
```

```
check long:
```

```
push ebp
```

```
mov ebp, esp
```

```
push ebx
```

```
mov eax, 0
```

```
mov ebx, [ebp+8] ; *str
```

.for :

```
    cmp byte [ebx], 0
    je .end
    inc ebx
    inc eax
    jmp .for
```

.end :

```
    mov ebx, [ebp + 12] ; dim
    sub eax, ebx ; eax = eax - ebx
    pop ebx
    mov esp, ebp
    pop ebp
    ret
```

c) #include <stdio.h>

```
extern char msg [];
extern char len;

int checkLong (char *msg, char dim);

int main (int argc, char *argv[]){
    if (checkLong(msg, len)){
        printf ("Chequeo fue exitoso \n");
    } else {
        printf ("Chequeo NO fue exitoso \n");
    }
    return 0;
}
```

d) los flags -m32 en gcc y -f elf32 asi nasm y gcc saben que se tratan de archivos de 32 bits.

-m32 en gcc producira un ejecutable que està destinado a ser ejecutado en un entorno de 32 bits.

-f elf32 en nasm hace lo mismo que gcc -m32 produce ejecutable para un entorno de 32 bits.

Ejercicio 2 (3 puntos)

Dado el siguiente programa que se compila para un procesador de 32 bits:

```
#include<stdio.h>

int calculo(int param1,int param2, char tipo) {

int resul;

if(tipo=='s')
    resul=param1+param2;
```

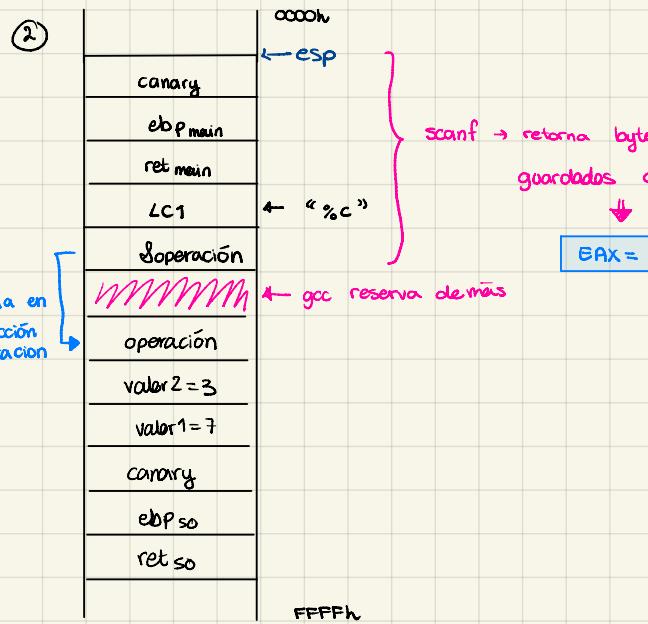
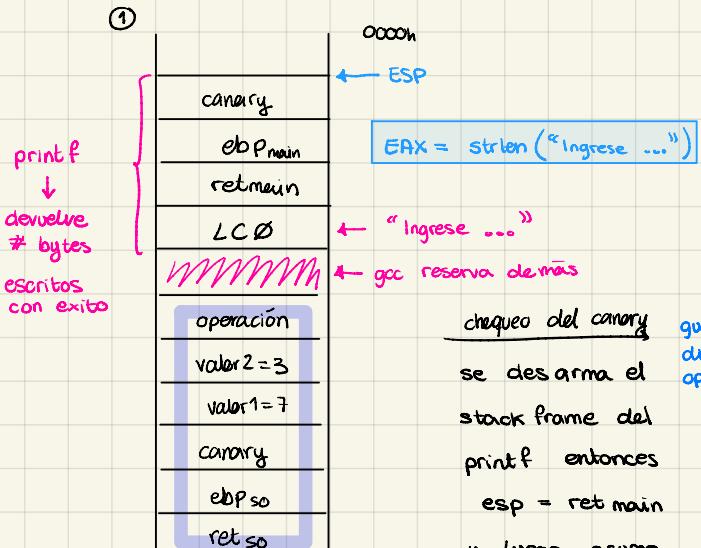
```
else
    resul=param1*param2;

return resul;
}

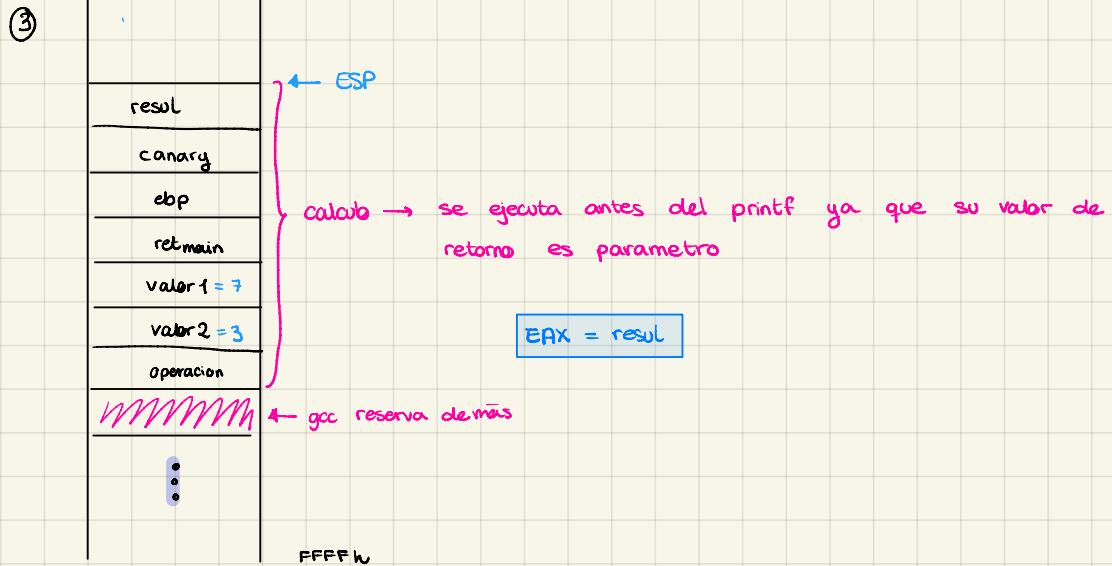
int main(void) {

int valor1=7;
int valor2=3;
char operacion;
printf("Ingrese el tipo de operación: 's' suma y 'm' multiplicar : \n");
scanf("%c",&operacion);
printf("Resultado: %d\n",calculo(valor1, valor2, operacion));
return 0;
}
```

- a. Explique con dibujos respectivos el contenido de la pila, durante toda la ejecución del programa.
b. Muestre los registros del microprocesador que intervienen en los dibujos de la pila.

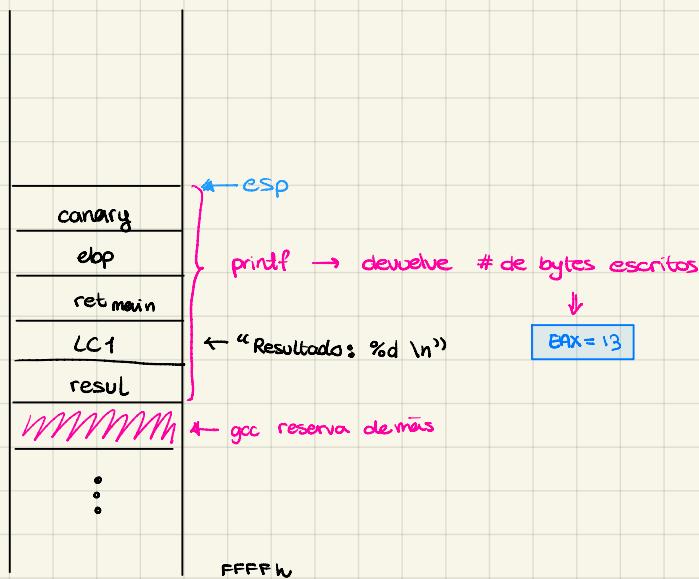


chequeo de canary
por si se sobre-escribió el stack
→
se desarma el stack frame del scanf entonces esp = ret main y luego asumo que se ejecuta add esp, 4*2 para "liberar" parámetros



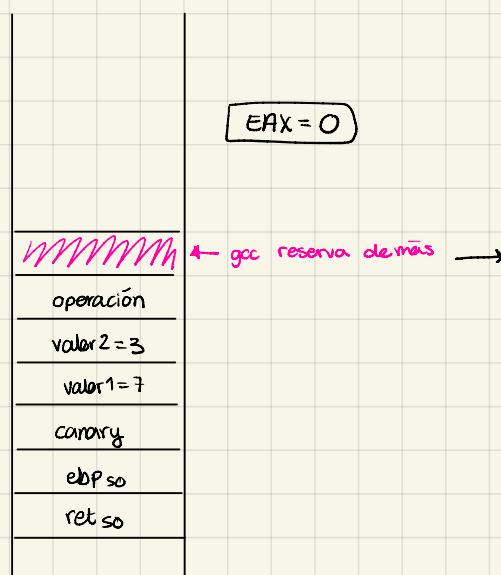
chequeo de canary
por si se sobre-escribió
el stack

se desarma el
stack frame del
calcúlo entonces
 $esp = ret\ main$
y luego asumo
que se ejecuta
 $add esp, 4*3$
para "liberar" parámetros



chequeo de canary
por si se sobre-escribió
el stack

se desarma el
stack frame del
printf entonces
 $esp = ret\ main$
y luego asumo
que se ejecuta
 $add esp, 4*2$
para "liberar" parámetros



- 1) Luego se mueve esp para liberar variable locales y espacio reservado
- 2) se chequea el valor del canary para verificar que no se sobre-escribió el stack
- 3) se desarma el stackframe y el $esp = ret so$

Ejercicio 3 (3 puntos)

Un alumno que todavía no cursó esta materia le pide a ud. que le explique:

- ¿Qué es una system call? ¿Quién la escribió? ¿Dónde se encuentran ubicadas?
- ¿Cómo se llama a una system call desde Assembler de Linux? Muestre con un ejemplo de llamada a read.
- ¿Qué es el número de ID que aparece en la clase teórica?

Como ud no recuerda de memoria el funcionamiento ingresa a un sistema operativo Linux y ejecuta los comandos "man 2 read" y obtiene la siguiente salida:

READ(2) Linux Programmer's Manual READ(2)

NAME read - read from a file descriptor

SYNOPSIS

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

DESCRIPTION

read() attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`.

On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.

If `count` is zero, read() may detect the errors described below. In the absence of any errors, or if read() does not check for errors, a read() with a count of 0 returns zero and has no other effects.

According to POSIX.1, if `count` is greater than `SSIZE_MAX`, the result is implementation-defined; see NOTES for the upper limit on Linux.

- a) system call → encuentran en kernel space
so escritas

un system call es una interfaz para comunicarse desde el userland al kernel space. de una forma controlada y protegida

Se encuentran escritas en el kernel space y están vinculadas a la interrupción int 80h (Linux x86). Se filtran las syscalls con su 'id' valor de eax y se busca en la tabla de system calls donde encuentra la routine asociada a tal 'id'

b)

```
mov eax, 3
mov ebx, 0 ; salida estandar
mov ecx, buffer ; sección .bss
mov edx, dim ; dimension bytes a leer
int 80h
```

- c) numero de ID es el numero único que identifica que tipo de syscall queres usar en tiempo de ejecución.

Como mencionado en a se usa el ID como indexación de las syscall en una tabla de system calls donde contiene la routine asociada al ID