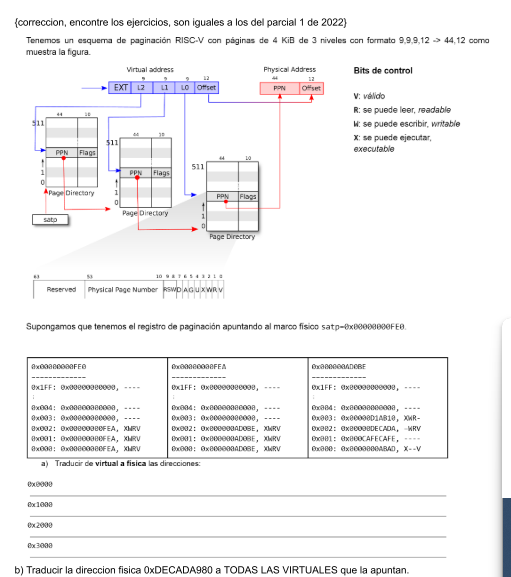
hola yago :3

si sos alguien distinto que haces aca



1. 0x1000 = 0xABAD000

El resto es page fault x los permisos

dir fisica = marco de pagina \* 0x1000(4kib = 2^12) + offset

dir fisica = marco de pagina << 12 + offset

1. para hacerlo hay que conseguir el marco de pagina

0xDECADA980 >> 12 bits(por 4kib) = 0xDECADA

offset = 0x980

ahora buscamos en la ultima page(3) table algun entry que se corresponda con lo obtenido

coincide 0x002, ahora veo quienes referencian a mi pt3

pt2 lo hace en 0x0, 0x1 y 0x2

quien referencia a pt2?

pt1 en 0x0 0x1 y 0x2

entonces las posibles combinaciones serian dadas por

PT1={0x000,0x001,0x002}

PT2={0x000,0x001,0x002}

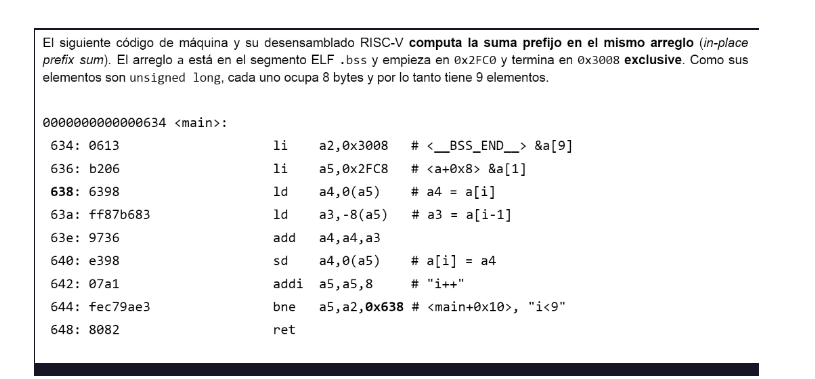
PT3={0x002}

offset = 0x980

ejemplo de como queda una porque alta paja, ya te hice alto resumen

0x 000 000 002 980

3) esta mal, abajo lo mejoro



pide: instrucciones ejecutadas y datos leidos y escritos

el programa guarda el arreglo desde 0x3008 hasta 0x2fc8

empieza a iterar en el segundo elemento i=1

en cada paso suma el anterior con el actual (a4 y a3)

y lo guarda en la posicion actual

i++

mientras q i<9

direcciones virtuales accedidas:

hay q identificar donde estan las instrucciones: 

y los datos? -> el arreglo esta desde 0x3008 hasta 0x2fc8

->para cada instruccion q accede a la memoria (ld,sd,etc) anotar:

direccion virtual de acceso

lectura o escritura?

por cada acceso hacer una traducción a direccion fisica

Usás el mismo esquema de paginación de 3 niveles del ejercicio 2. Entonces:

Tomás la dirección virtual → extraés:

VPN[2], VPN[1], VPN[0], Offset

Usás las tablas P1, P2, P3 del Ejercicio 2 para traducir.

Si encontrás una entrada válida y hoja, usás:

marco\_físico << 12 | offset

Si no hay entrada válida → page fault (pero asumimos que el programa corre, así que todas deben traducirse bien).

Armá la **traza de memoria física**

Por cada instrucción ejecutada o acceso a dato, anotá:

<tipo> <virtual> → <física>

LOAD 0x2FC8 → 0x\_\_\_\_\_\_

STORE 0x2FD0 → 0x\_\_\_\_\_\_

FETCH 0x063a → 0x\_\_\_\_\_\_

eu, si llegaste hasta aca sos el masca

veamos q sale:

se me acabo el plan free de chatrpt

—-----------correccion—---------

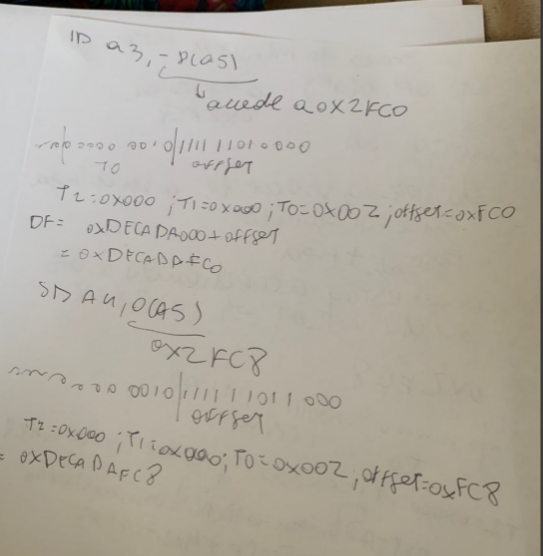
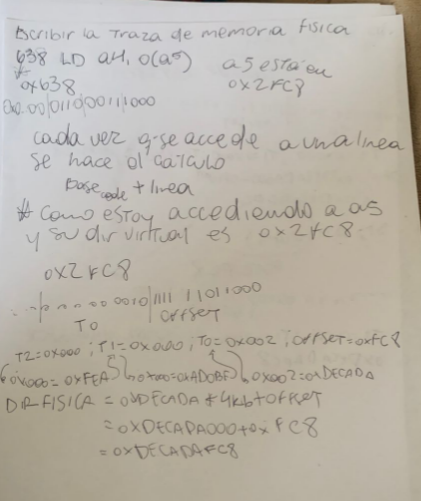
Primera instrucción a analizar: línea 638: ld a4, 0(a5)

Sabemos que:

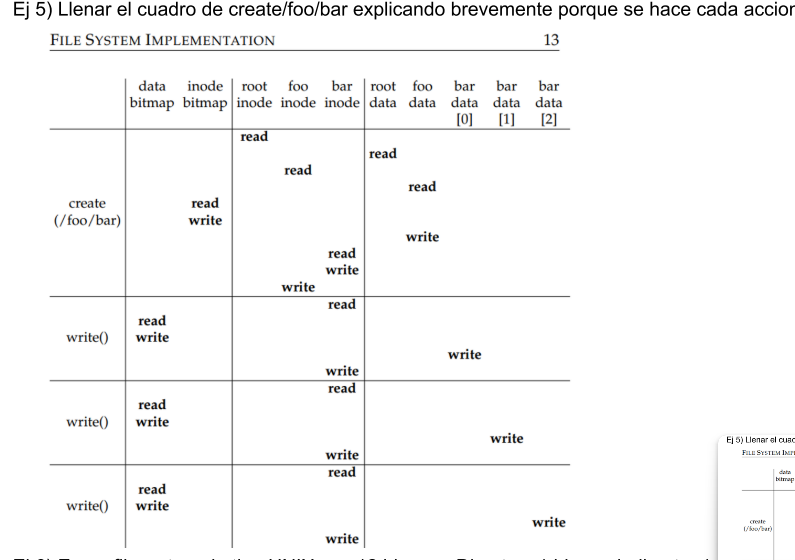
a5 = 0x2FC8 (lo cargaste en la línea anterior con li a5, 0x2FC8)

La instrucción ld a4, 0(a5) accede a dirección virtual = 0x2FC8

Paso 1: Descomponer dirección virtual 0x2FC8 en SV39



no estoy muy segura de q sea asi igual



el primer write: primero lee bar inode para investigar la informacion del archivo y su inode, luego en data bitmap con read write asigna la seccion utilizada al archivo bar. se inicializa el primer bloque de datos dle archivo en bar data {0} con write y se actualiza la info en bar inode

el segundo write lee el estado actual del inodo de bar (verifica cuanto ocupa, que bloques estan disponibles para seguir escribiendo info, etc). lee el data bitmap para saber que region de memoria esta libre para asignarle a bar y se lo asigna. ahora se escribe en el bloque de memoria {1} de bar. finalmente se actualizan los valores de bar inode

el tercer write vuelve a investigar el estado actual del inodo de bar para ver cuanta memoria utiliza y que bloques de memoria ya han sido asignados , luego se verifica en el data bitmap que seccion de memoria puede ser utilizada y se la asigna a bar. se escribe en el bloque de memoria de bar data [2] y finalmente se actualiza el estado de bar inode

.

primero obtengo los datos del enunciado:

12 bloques directos, 1 bloque indirecto, 1 bloque doble indirecto, 1 bloque triple indirecto

tamaño de bloque = 4Kib = 1024bytes

indices de bloque = 32 bits

cada inode ocupa 128bytes

i-bitmap -> 1 bit por inodo

entonces para 1024 inodos> 1\*1024=1024bits=128 bytes

para 2^20 inodos -> 1\*2^20 = 1048576 bits / 8 = 131072 bytes = 128Kib

d-bitmap -> cantidad de bloques \* cant de inodos

1024 inodos: (12+1024+1024^2+1024^3)\*1024 = 1100586430464bits = 128gb aprox

2^20 inodos = (12+1024+1024^2+1024^3)\*2^20 = 128tb aprox

inode table = cant inodes \* tamaño

1024 : 1024\*128=131072 bytes = 128kb

2^20 = 1048576 \* 128 =134217728 bytes = 128 mb

data region = cantidad de bloques \* tamaño de bloque \*cant inodos

1024: (12+1024+1024^2+1024^3) \* 4096(4kb) \* 2bytes= 8804691443712 bytes = 8tb

2^20 = (12+1024+1024^2+1024^3) \* 4096 \* 2^20

na deja ni idea

entonces la tabla queda

|  | 1024 | 2^20 |
| --- | --- | --- |
| ibitmap | 128bytes | 128kib |
| dbitmap | 128gb | 128tb |
| inode table | 128kib | 128mib |
| data region | ni idea | :( |