API de Memoria

Ejercicio 1. Para cada una de las variables de este código indicar si están en el segmento de código, de pila o de montículo (heap). Si hay punteros indicar a que segmento apunta.

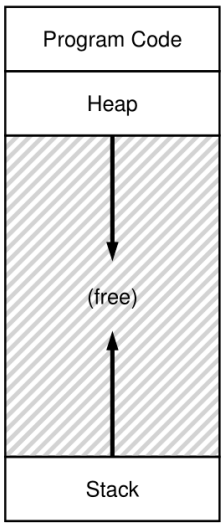
Extra:¿Dónde se ubica el arreglo global si lo declaramos inicializado a cero? `int a[N] = {0};`

```c
#include <stdlib.h>
#define N 1024

int a[N];

int main(int argc, char ** argv)
{
  int i;
  register int s = 0;
  int *b = calloc(N, sizeof(int));
  for (i=0; i<N; ++i)
    s += a[i]+b[i];

  free(b);
  return s;
}
```



Ejercicio 2. Debuggear el mal uso de memoria en los siguientes pedacitos de código.

```c
char *s = malloc(512);
gets(s);
```

```c
char *s = "Hello Waldo";
char *d = malloc(strlen(s));
strcpy(d,s);
```

```c
char *s = "Hello Waldo";
char *d = malloc(strlen(s));
d = strdup(s);
```

```c
int * a = malloc(16)
a[15] = 42;
```

Ejercicio 3. Verdadero o falso. Explique.

(a) `malloc()` es una syscall.

(b) `malloc()` siempre llama a una syscall.

(c) `malloc()` a veces produce una llamada a una syscall.

(d) Idem con `free()`.

(e) El tiempo de cómputo que toma `malloc(x)` es proporcional a `x`.

Traducción de direcciones

Ejercicio 4. Mostrar la secuencia de accesos a la memoria física que se produce al ejecutar este programa assembler `x86_32`, donde el registro `base=4096` y `bounds=256`.

```
0: movl $128,%ebx
5: movl (%ebx),%eax
8: shll $1, %ebx
10: movl (%ebx),%eax
13: retq
```

Ejercicio 5. Mostrar con un ejemplo de disposición de memoria física de varios procesos como el esquema de traducción de direcciones con base+límite puede producir **fragmentación interna** y **fragmentación externa**.

Ejercicio 6. Distinguir **relocalización estática** y **relocalización dinámica**.

Ejercicio 7. Verdadero o falso. Explique.

(a) Modificar los registros `base` y `bounds` son instrucciones privilegiadas.

(b) Hay un juego de registros `(base, bounds)` por cada proceso.

Segmentación

Ejercicio 8. Una computadora proporciona a cada proceso 65536 bytes de espacio de direcciones dividido en páginas de 4 KiB. Un programa específico tiene el segmento código de 32768 bytes de longitud, el segmento montículo de 16386 bytes de longitud, y un segmento pila de 15870 bytes.
¿Cabrá el programa en el espacio de direcciones? ¿Y si el tamaño de página fuera de 512 bytes?
Recuerde que una página no puede contener segmentos de distintos tipos asi se pueden proteger cada uno de manera adecuada.

Manejo del Espacio Libre

Ejercicio 9. Suponga un sistema de memoria contiguo con la siguiente secuencia de tamaños de huecos: 10 KiB, 4 KiB, 20 KiB, 18 KiB, 7 KiB, 9 KiB, 12 KiB, 15 KiB.
Para la siguiente **secuencia de solicitudes** de segmentos de memoria: 12 KiB, 10 KiB, 9 KiB.
¿Cuáles huecos se toman para las distintas políticas?

(a) Primer ajuste (*first fit*).

(b) Mejor ajuste (*best fit*).

(c) Peor ajuste (*worst fit*).

(d) Siguiente ajuste (*next fit*).

Paginación

Ejercicio 10. La TLB de una computadora con una pagetable de un nivel tiene una eficiencia del 95 %. Obtener un valor de la TLB toma 10ns. La memoria principal tarda 120ns. ¿Cuál es el tiempo promedio para completar una operación de memoria teniendo en cuenta que se usa tabla de páginas lineal?

Ejercicio 11. Considere el siguiente programa que ejecuta en un microprocesador con soporte de paginación, páginas de 4 KiB y una TLB de 64 entradas.

```
    int x[N];
    int step = M;
L1: for (int i=0; i<N; i+=step)
        x[i] = x[i]+1;
    goto L1;
```

(a) ¿Qué valores de `N`, `M` hacen que la TLB falle en cada iteración del ciclo?

(b) ¿Cambia en algo si el ciclo ser repite muchas veces? Explique.

Ejercicio 12. Dado un tamaño de página de 4 KiB = 2^{12} bytes y la tabla de paginado de la Fig. 1.

(a) ¿Cuántos bits de direccionamiento hay para cada espacio?

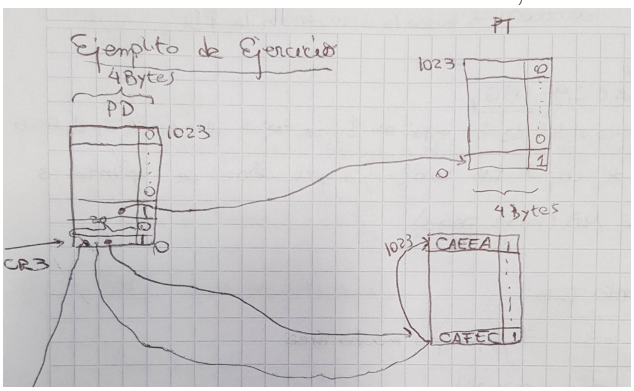
(b) Determine las direcciones físicas a partir de las virtuales: 39424, 12416, 26112, 63008, 21760, 32512, 43008, 36096, 7424, 4032.

(c) Determine el mapeo inverso, o sea las direcciones virtuales a partir de las direcciones físicas: 16385, 4321.

V	F	¿Válida?
0	000	1
1	111	1
2	000	0
3	101	1
4	100	1
5	001	1
6	000	0
7	000	0
8	011	1
9	110	1
10	100	1
11	000	0
12	000	0
13	000	0
14	000	0
15	010	1

Figura 1:

Ejercicio 13. Dado el siguiente esquema de paginación i386 $(10, 10, 12)$ traducir la direcciones virtuales `0x003FF666`, `0x00000AB0`, `0x00800B0B` a físicas.



Ejercicio 14. Dado el sistema de paginado de dos niveles del i386 direcciones virtuales de 32 bits, direcciones físicas de 32 bits, 10 bits de índice de *page directory*, 10 bits de índice de *table directory*, y 12 bits de *offset* dentro de la página, o sea un $(10, 10, 12)$, indicar:

(a) Tamaño de total ocupado por el directorio y las tablas de página para mapear 32 MiB al principio de la memoria virtual.

(b) Tamaño total del directorio y tablas de páginas si están mapeados los 4 GiB de memoria.

(c) Dado el ejercicio anterior ¿Ocuparía menos o más memoria si fuese una tabla de un solo nivel? Explicar.

(d) Mostrar el directorio y las tablas de página para el siguiente mapeo de virtual a física:

Virtual	Física
$[0MiB, 4MiB)$	$[0MiB, 4MiB)$
$[8MiB, 8MiB + 32KiB)$	$[128MiB, 128MiB + 32KiB)$

Ejercicio 15. Explique porque un i386 no puede mapear los 4 GiB completos de memoria virtual. ¿Cuál es el máximo?

Ejercicio 16. Explique como podría extender el esquema de memoria virtual del i386 para que, aunque cada proceso tenga acceso a 4 GiB de memoria virtual (32 bits), en total se puedan utilizar 64 GiB (36 bits) de memoria física[1].

Ejercicio 17. ¿Verdadero o Falso? Explique.

(a) Hay un *page directory* por cada proceso.

(b) La MMU siempre mapea una memoria virtual más grande a una memoria física más pequeña.

(c) Dos páginas virtuales de un mismo proceso se pueden mapear a la misma página física.

(d) Dos páginas virtuales de dos procesos se pueden mapear a la misma página física.

[1] Esto se conoce como *page address extension* – PAE.

(e) Dos páginas físicas de un mismo proceso se pueden mapear a la misma página virtual.

(f) En procesadores de 32 bits y gracias a la memoria virtual, cada proceso tiene 2^{32} direcciones de memoria virtuales.

(g) La memoria virtual se puede usar para ahorrar memoria.

(h) Toda la memoria virtual tiene que estar mapeada a memoria física.

(i) El *page directory* en i386 se comparte entre todos los procesos.

(j) Puede haber marcos de memoria física que no tienen un marco de memoria virtual que los apunte.

(k) Por culpa de la memoria virtual hacer un `fork` resulta muy caro en términos de memoria.

(l) Los procesadores tienen instrucciones especiales para acceder a la memoria física evitando la MMU.

(m) Es imposible hacer el mapeo inverso de física a vitrual.

(n) No se puede meter un todo un Sistema Operativo completo con memoria paginada i386 en 4 KiB.

Ejercicio 18. Se define un *page directory* donde la última entrada, la 1023, apunta a la base del mismo[2].

(a) ¿A dónde apunta la dirección virtual `0xFFC00000`?

(b) ¿Y la dirección virtual `0xFFFE0000`?

(c) Indique a donde apunta la dirección virtual `0xFFFFF000`.

(d) Finalmente, describa para que sirve este esquema de memoria virtual.

Ejercicio 19. Explique como se usa la paginación para hacer:

(a) Dereferenciamiento de puntero a `NULL` tira excepción.

(b) Archivo de intercambio o *swap file*.

(c) *Demand paging* para la carga de programas.

(d) Auto-growing stack.

(e) Non-executable stacks.

(f) Memory mapped files `mmap()`.

(g) Copy-on-write (COW) para el `fork()`.

(h) `sbrk()` barato y por lo tanto `malloc()` barato.

(i) `malloc();memset(0)` = `calloc()` barato.

(j) Código compartido entre procesos: *shared libraries*, código de kernel, etc.

(k) Memoria compartida entre procesos.

[2]Mirar *recursive page tables*, en "This OS is a Boot Sector", POC||GTFO `0x04`, 2014