Ej1)

Se cambió ligeramente el *trapframe* donde se mantiene el estado de los registros dentro del *process control block* de **xv6** (1ra columna).
Dada la función `mcd()` (2da columna), se la compila a i386 con `-O0` (3ra columna) y con `-O1` (4ta columna).

La opción `-O0` significa sin optimizaciones.
La opción `-O1` significa con optimizaciones básicas.

Para las dos versiones en ensablador, diga si el *context switch* funciona **siempre**, **a veces** o **nunca**.

```
struct trapframe {
  uint edi;
  uint esi;
  uint ebp;
  uint oesp;
  uint ebx;
  uint ecx;
  uint eax;
  uint gs;
  uint fs;
  uint es;
  uint ds;
  uint trapno;
  uint err;
  uint eip;
  uint cs;
  uint eflags;
  uint esp;
  uint ss;
}
```

```
int mcd(int a, int b)
{
  while(a!=b) {
    if (a<b)
      b = b - a;
    else
      a = a - b;
  }
  return a;
}
```

```
mcd:
    jmp L2
L4:
    movl 4(%esp), %eax
    cmpl 8(%esp), %eax
    jge L3
    movl 4(%esp), %eax
    subl %eax, 8(%esp)
    jmp L2
L3:
    movl 8(%esp), %eax
    subl %eax, 4(%esp)
L2:
    movl 4(%esp), %eax
    cmpl 8(%esp), %eax
    jne L4
    movl 4(%esp), %eax
    ret
```

```
mcd:
    movl 4(%esp), %edx
    movl 8(%esp), %eax
    cmpl %eax, %edx
    jne L5
L2:
    ret
L3:
    subl %eax, %edx
L4:
    cmpl %eax, %edx
    je L2
L5:
    cmpl %eax, %edx
    jge L3
    subl %edx, %eax
    jmp L4
```

Para la versión sin optimizaciones observamos que se utilizan los registros %esp y %eax los cuales si están contenidos en el trapframe. Entonces no debería haber problema y siempre ha de funcionar.

Para la versión con optimizaciones se utilizan los registros %esp, %eax y %edx, para este último la trapframe no nos asegura su correcto restaurado ya que no está presente. Entonces esta versión anda solo a veces, mientras que no ocurra ningún tipo de trap en la ejecución del programa.

Pregunta **2**

Correcta

Puntúa 1,00 sobre 1,00

Para los siguientes procesos CPU-bound completar la secuencia temporal de como se planifican los procesos en un **mono procesador** para una política de planificación SJF.

Proceso	Tarribo	TCPU
A	0	4
B	2	2
C	3	1

Ejemplo de respuesta: ABBAABBA.

SJF (Shortest Job First)

$T_{ejecucion}$	A	A	A	A	C	B	B
T_{arrivo}	A		B	C			
T_{real}	0	1	2	3	4	5	6

Respuesta correcta: AAAACBB

Pregunta **3**

Correcta

Puntúa 1,00 sobre 1,00

Para los siguientes procesos que alternan CPU e IO, completar la secuencia temporal de como se planifican los procesos en una computadora **monocore** para una política de planificación RR con Q=200.

Proceso	Tarribo	TCPU	TIO	TCPU
A	0	2	1	2
B	1	4	1	4

Ejemplo de la respuesta: ABBAABBA.

	A		B	B	B	B	A	A	B	B	B
	B	B	A	A	A	A		B			
		A					B				
	B										

	1	2	3	4	5	6	7	8	9		

Respuesta correcta: AABBBBAABBBB

Pregunta **4**

Correcta

Puntúa 1,00 sobre 1,00

Al ejecutar un proceso usando memoria segmentada se produce la siguiente secuencia de accesos a la memoria virtual donde **C** indica un acceso a segmento de código, **H** al segmento de heap y **S** al segmento de stack. El número hexadecimal indica la dirección de memoria virtual y el decimal luego de la coma, la longitud del acceso.

```
C 0x00401000,4
C 0x00401004,10
H 0x00404000,4
C 0x0040100e,10
H 0x00404004,4
C 0x00401018,10
H 0x00404008,4
C 0x00401022,10
H 0x0040400c,4
C 0x0040102c,5
C 0x00401031,5
S 0x1ffeffffe0,8
C 0x00401040,7
C 0x00401047,5
C 0x0040104c,5
C 0x00401051,2
C 0x00401065,2
C 0x00401067,2
```

Si se define como segmento base del código como 0xFE100000, indicar la dirección de memoria física del primer acceso al código.

Ejemplo de respuesta: 0x8BADF00D

0x00401000 = 0xb 0000 0000 0100 0000 0001 0000 0000 0000
0xFE100000 = 0xb 1111 1110 0001 0000 0000 0000 0000 0000

SUMA = 0xb 1111 1110 0101 0000 0001 0000 0000 0000 = 0xFE501000

Pregunta **5**

Correcta

Puntúa 1,00 sobre 1,00

Al ejecutar un proceso usando memoria segmentada se produce la siguiente secuencia de accesos a la memoria virtual donde **C** indica un acceso a segmento de código, **H** al segmento de heap y **S** al segmento de stack. El número hexadecimal indica la dirección de memoria virtual y el decimal luego de la coma, la longitud del acceso.

```
C 0x00401000,4
C 0x00401004,10
H 0x00404000,4
C 0x0040100e,10
H 0x00404004,4
C 0x00401018,10
H 0x00404008,4
C 0x00401022,10
H 0x0040400c,4
C 0x0040102c,5
C 0x00401031,5
S 0x1ffeffffe0,8
C 0x00401040,7
C 0x00401047,5
C 0x0040104c,5
C 0x00401051,2
C 0x00401065,2
C 0x00401067,2
```

Si se define como segmento base del heap como 0x1A000000, indicar la dirección de memoria física del primer acceso al heap.

Ejemplo de respuesta: 0xABADBABE

0x00404000 = 0xb 0000 0000 0100 0000 0100 0000 0000 0000
0x1A000000 = 0xb 0001 1010 0000 0000 0000 0000 0000 0000
suma = 0xb 0001 1010 0100 0000 0100 0000 0000 0000 = 0x1A404000

Pregunta **6**

Correcta

Puntúa 1,00 sobre 1,00

Al ejecutar un proceso usando memoria segmentada se produce la siguiente secuencia de accesos a la memoria virtual donde **C** indica un acceso a segmento de código, **H** al segmento de heap y **S** al segmento de stack. El número hexadecimal indica la dirección de memoria virtual y el decimal luego de la coma, la longitud del acceso.

```
C 0x00401000,4
C 0x00401004,10
H 0x00404000,4
C 0x0040100e,10
H 0x00404004,4
C 0x00401018,10
H 0x00404008,4
C 0x00401022,10
H 0x0040400c,4
C 0x0040102c,5
C 0x00401031,5
S 0x1ffefffe0,8
C 0x00401040,7
C 0x00401047,5
C 0x0040104c,5
C 0x00401051,2
C 0x00401065,2
C 0x00401067,2
```

Si se define como segmento base del stack como 0x0, indicar la dirección de memoria física del primer acceso al stack.

Ejemplo de respuesta: 0xB105F00D

0x1FFEFFFFE0 = 0001 1111 1111 1110 1111 1111 1111 1111 1110 0000
0x0 = 0

Rta: 0x1FFEFFFFE0

Pregunta **7**

Correcta

Puntúa 1,00 sobre 1,00

Considere la tabla de páginas lineal de abajo para un espacio de direcciones virtual y físico de 32 bits, con 20 bits para número de marco virtual y 12 bits de offset.
Determine que dirección física que se corresponde a la dirección virtual 0x0000301A.

Si es *page fault* poner **PF**, si no poner en hexadecimal, por ejemplo 0x0DEFACED.

```
CR3=0x00100

0x00100
--------
0: 0xC0CA9, -, RWX
1: 0xC0CAA, P, RWX
2: 0xC0CAB, -, RWX
3: 0xC0CAC, P, RWX
4: 0xC0CAD, -, RWX
5: ...
```

0x0000301A = 0xb 0000 0000 0000 0000 0011 0000 0001 1010

VPN = 3, OFFSET = 1A
→ PFN = 0xCOCAC01A

Pregunta **8**

Correcta

Puntúa 1,00 sobre 1,00

Considere la tabla de páginas lineal de abajo para un espacio de direcciones virtual y físico de 32 bits, con 20 bits para número de marco virtual y 12 bits de offset.
Determine que dirección física que se corresponde a la dirección virtual 0x00000FFF.

Si es *page fault* poner **PF**, en caso contrario poner en hexadecimal, por ejemplo 0xBAADF00D.

```
CR3=0x00100

0x00100
--------
0: 0xC0CA9, -, RWX
1: 0xC0CAA, P, RWX
2: 0xC0CAB, -, RWX
3: 0xC0CAC, P, RWX
4: 0xC0CAD, -, RWX
5: ...
```

0x00000FFF = 0xb 0000 0000 0000 0000 0000 1111 1111 1111

VPN = 0, OFFSET = FFF
→ PF

Pregunta **9**

Correcta

Puntúa 1,00 sobre 1,00

Considere la tabla de páginas lineal de abajo para un espacio de direcciones virtual y físico de 32 bits, con 20 bits para número de marco virtual y 12 bits de offset.
Determine que dirección **virtual** que se corresponde a la dirección física 0xC0CAADDA.

Si es no está mapeada poner PF, , en caso contrario poner en hexadecimal, por ejemplo 0xC00010FF.

```
CR3=0x00100

0x00100
--------
0: 0xC0CA9, -, RWX
1: 0xC0CAA, P, RWX
2: 0xC0CAB, -, RWX
3: 0xC0CAC, P, RWX
4: 0xC0CAD, -, RWX
5: ...
```

0xC0CAADDA = 0xb 1100 0000 1100 01010 1010 1101 1101 1010

PFN =C0CAA, OFFSET = DDA
→ VPN = 0x00001DDA

Pregunta **10**

Correcta

Puntúa 1,00 sobre 1,00

Tenemos un esquema de paginación i386 o sea (10,10,12).
10 bits de índice de directorio, 10 bits de índice de tabla de página y 12 bits de offset.

Dar la dirección física de la dirección virtual 0x00C03EEE.

Si hay *page fault* poner PF.

```
CR3=0x01011

0x01010               |  0x01011
---                   |
   0: 0x01010, P, RWX     0: 0x01011, P, RWX
   1: 0x01011, P, RWX     1: 0x01010, P, RWX
   2: 0x01010, P, RWX     2: 0x01011, P, RWX
   3: 0x01011, P, RWX     3: 0x01010, P, RWX
...
1023: 0x01011, P, RWX  1023: 0x01010, P, RWX
```

0x00C03EEE = 0xb 0000 0000 1100 0000 0011 1110 1110 1110

PDI = 3
PTE = 3
OFFSET = EEE
PFN → 0x01011EEE

Pregunta **11**

Correcta

Puntúa 1,00 sobre 1,00

Tenemos un esquema de paginación i386 o sea (10,10,12).
10 bits de índice de directorio, 10 bits de índice de tabla de página y 12 bits de offset.

Dar la dirección física de la dirección virtual 0x00C03AAA.

Si hay *page fault* poner PF.

```
CR3=0x01010

0x01010                |  0x01011
---                    |  ---
   0: 0x01010, P, RWX      0: 0x01011, P, RWX
   1: 0x01011, P, RWX      1: 0x01010, P, RWX
   2: 0x01010, P, RWX      2: 0x01011, P, RWX
   3: 0x01011, P, RWX      3: 0x01010, P, RWX
...                    ...
1023: 0x01011, P, RWX  1023: 0x01010, P, RWX
```

0x00C03AAA = 0xb 0000 0000 1100 0000 0011 1010 1010 1010

PDI = 3
PTE = 3
OFFSET = AAA
PFN → 0x01010AAA

Pregunta **12**

Correcta

Puntúa 1,00 sobre 1,00

Tenemos un esquema de paginación i386 o sea (10,10,12).
10 bits de índice de directorio, 10 bits de índice de tabla de página y 12 bits de offset.

Con el siguiente esquema de paginación, decir cuantas páginas físicas, incluyendo directorios y tablas de página, son accesibles desde **todo el mapa de memoria virtual**.

```
CR3=0x01010

0x01010                |  0x01011
---                    |  ---
   0: 0x01010, P, RWX     0: 0x01011, P, RWX
   1: 0x01011, P, RWX     1: 0x01010, P, RWX
   2: 0x01010, P, RWX     2: 0x01011, P, RWX
   3: 0x01011, P, RWX     3: 0x01010, P, RWX
...                    ...
1023: 0x01011, P, RWX  1023: 0x01010, P, RWX
```

2