

0x1000 = 0… 0 0001 0000 0000 0000 offset

t2 = 0x0

t1 = 0x0

t0 = 0x1

page fault -> dir no valida

0x2000 = 0010 0000 0000 0000 Offset bien maleducado.

L2 = 9 0

L1 = 9 0

L0 = 0002

0xDECADA\* 4kib + offset

0xDECADA000 nasi

0xDECADA\* 4kib + offset

de fisica a virtual: 0xDECADA980

0xDECADA980 / 4kib - offset

0XDECADA

| 0xFE0  0x2  0x1  0x0 | 0xFEA  0x2  0x1  0x0 | 0xAD0BE  0x02 0xDECADA |
| --- | --- | --- |

000000000 000000000 000000010

000000000 000000001 000000010

000000000 000000010 000000010

000000001 000000000 000000010

000000001 000000001 000000010 + offset 1001 1000 0000

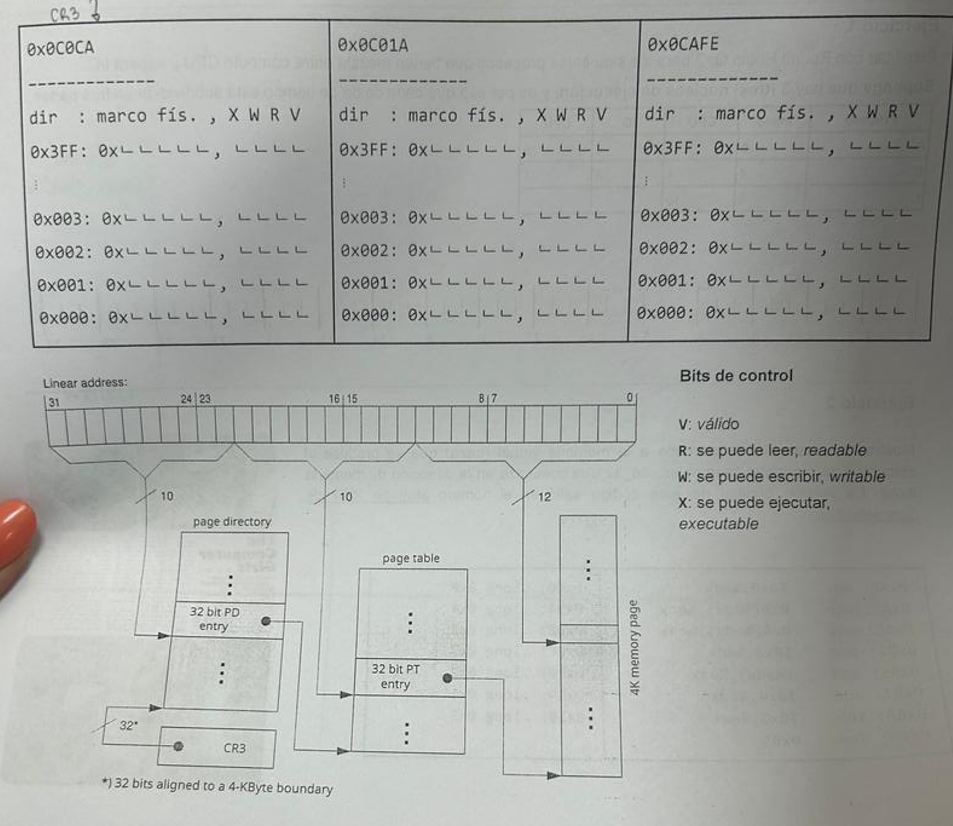
000000001 000000010 000000010

000000010 000000000 000000010

000000010 000000001 000000010

000000010 000000010 000000010

cr3 apunta si o si a 0xC01A



0co1a xwrv 0CAFE x w r v 0COCA x w r v

0x300: 0x0CAFE X W R -

LO MISMO PARA TODO

COCA

0C01A XWRV 0 c a f e x w r v 0 0 6 0 D X W R V

0x0BAD -> 0x60DBAD

offset = 0xBAD

l1= 0x0

l0 = 0x0

0x60DBAD = dir virtual \* kib + offset

0x60D000 = dir virtual \* 4kib

0x60D = Dir virtual

0xC0000FEE -> PF

1100 0000 00 |. 00 0000 0000

(0011) (00 00 )(00 00) = 0x 3 0 0

0xFFFFF000 -> 0x0C0CA\000

0xffff\*4kib + offset 0

1111 1111 11 11 1111 1111

t1= 1111 1111 11=3FF

t2 = 11 1111 1111= 3FF

0xFFC00 000 -> 0x0CAFE000

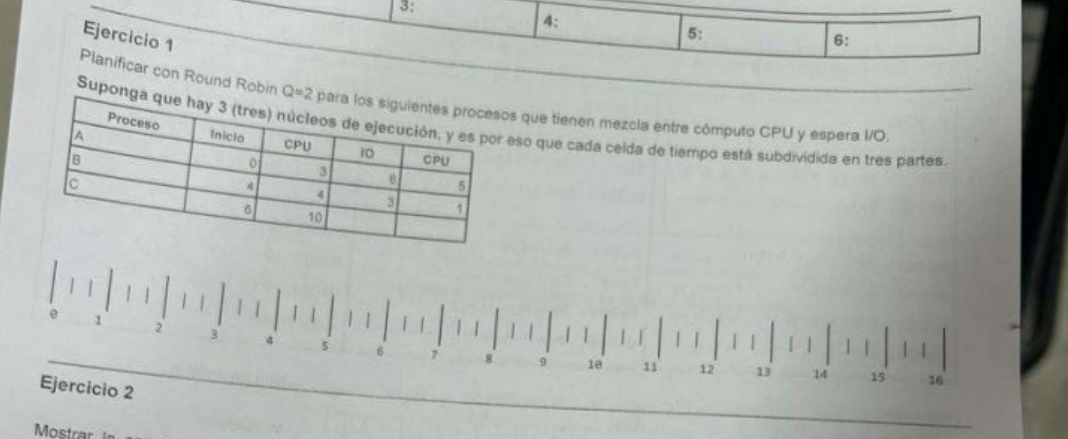
0xffc00 \* 4kib + offset 0

1111111110 1000000000

3FE 200

0xFFC00 000 = 1111 1111 11 . | 00 0000 0000 | 0000 0000 0000

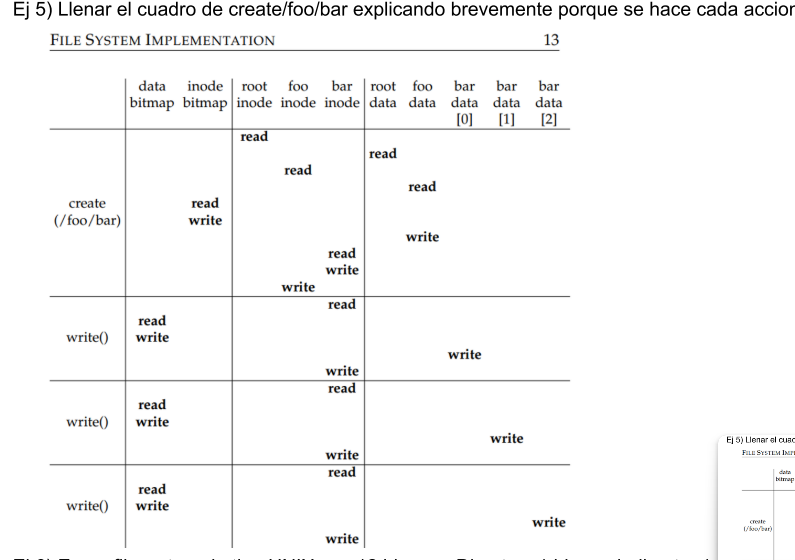
3FF 0

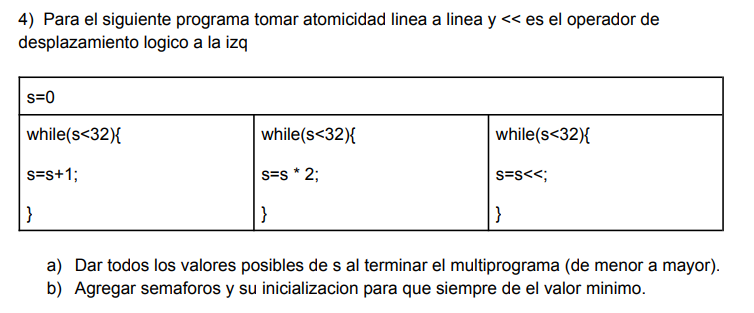


I/O A A A A A A B B B

A A A

B B C B C B C C A C A C A B CA CA C C











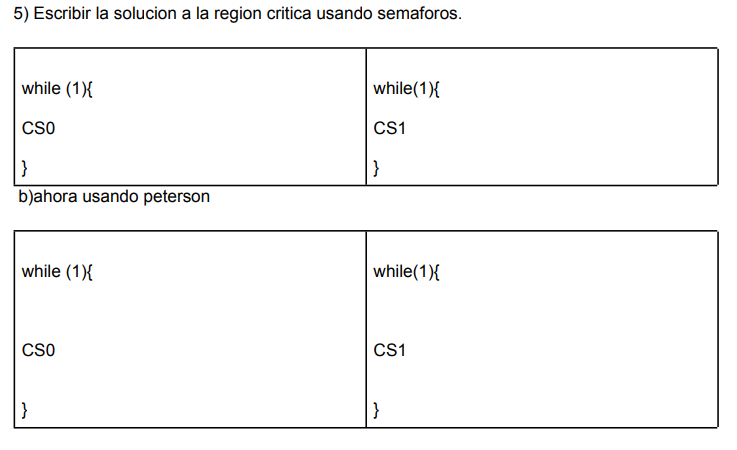




| void\* P0(void\* arg) {   sem\_wait(&semP0);  **while**(i<32){    i = i +1;  printf("P0\n");   }  sem\_post(&semP1);  **return** NULL; } |
| --- |

| void\* P1(void\* arg) {   **if**(i< 32){  sem\_post(&semP0);  }  sem\_wait(&semP1);  **while**(i<32){    i = i\*2;  printf("P1\n");  sem\_post(&semP0);  }  **return** NULL; }   void\* P2(void\* arg) {  **if**(i< 32){  sem\_post(&semP1);  }  sem\_wait(&semP2);    **while**(i<32){  i=i\*8;  printf("P2\n");  sem\_post(&semP1);  }  **return** NULL; } |
| --- |

| #include <stdio.h> #include <stdlib.h> #include <pthread.h> #include <semaphore.h> #include <unistd.h>   // Variables compartidas **int** i = 0;   sem\_t semP0, semP1, semP2;   **int** main() {  pthread\_t t0, t1, t2;   // Inicialización de semáforos  sem\_init(&semP0, 0, 0); // P1 espera **a** P2  sem\_init(&semP1, 0, 0); // P0 espera **a** P1  sem\_init(&semP2, 0, 1);     // Crear hilos  pthread\_create(&t0, NULL, P0, NULL);  pthread\_create(&t1, NULL, P1, NULL);  pthread\_create(&t2, NULL, P2, NULL);   // Esperar indefinidamente  pthread\_join(t0, NULL);  pthread\_join(t1, NULL);  pthread\_join(t2, NULL);   // Mostrar **el** resultado final del arreglo  printf("Resultado final de i: %d\n", i);     // Nunca llega aquí **en** bucle infinito, pero por limpieza:  sem\_destroy(&semP0);  sem\_destroy(&semP1);  sem\_destroy(&semP2);  **return** 0; } |
| --- |



El a) es trivial simplemente asegura de hacer la sincronización dentro del while

b)

| //Variables compartidas bool flag[2] = {false,false} **int** turn; |
| --- |

| **while**(1){  flag[0] = true;  turn = 1;  **while**(flag[1] && turn == 1){};  CS0  flag[0] = false; } | **while**(1){  flag[0] = true;  turn = 0;  **while**(flag[1] && turn == 0){};  CS1  flag[0] = false; } |
| --- | --- |

Algoritmo de exclusión mutua de Peterson general

| // Variables compartidas bool flag[2] = {false, false}; **int** turn; // indica **a** quién **le** cedemos prioridad  // Proceso P0 void P0() {  flag[0] = true;  turn = 1;  **while** (flag[1] && turn == 1) {  ; // espera activa  }  // ★ Sección **cr**ítica ★  flag[0] = false; }  // Proceso P1 void P1() {  flag[1] = true;  turn = 0;  **while** (flag[0] && turn == 0) {  ; // espera activa  }  // ★ Sección **cr**ítica ★  flag[1] = false; } |
| --- |