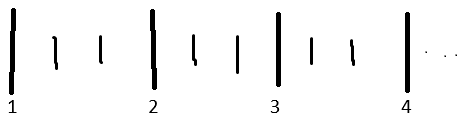
1)Planificar con RR, Q=2, para los siguientes procesos que tienen mezcla de cómputo CPU y espera I/O. Suponga que hay 3 nucleos de ejecución (por eso cada celda de tiempo esta subdividida en 3 partes)

| Proceso | Inicio | Cpu | I/O | CPU |
| --- | --- | --- | --- | --- |
| A | 0 | 3 | 6 | 5 |
| B | 4 | 4 | 3 | 1 |
| C | 6 | 10 |  |  |

La tablita era algo asi hasta 16.

Scheduling Policies. Recordar 1 proceso por núcleo.

**FIFO** (First In, First Out): Algoritmo de planificación donde el primer proceso que llega es el primero en ejecutarse, como una fila de espera.

**SJF** (Shortest Job First): Ejecuta primero el proceso con el tiempo de ejecución más corto, optimizando el tiempo total.

**JTCF** (Job Turnaround Completion First): Prioriza procesos según su tiempo de finalización, minimizando el tiempo de espera (parecido a SJF en enfoque).

**RR** (Round Robin): Asigna un tiempo fijo (quantum) a cada proceso en orden circular, asegurando equidad.

**MLFQ** (Multilevel Feedback Queue): Usa múltiples colas con prioridades que se ajustan dinámicamente según el comportamiento del proceso.1 NASIUU

Q=2

i/o: A A A A A A B B B B \\\

A A A —------ B B B C B C C A C A C A C A C B A C C C

2) Mostrar la secuencia de acceso a la memoria virtual (traza) que se produce al ejecutar este programa assembler x86-32 que comienza en la direccion de memoria 0xA0.

| 0xA0: mov $0x0, %edx  0xA6: mov 0x0(%edx),%ecx  0xAC: mov 0x4(%edx), %eax  0xB2: add $0x8, %edx  0xB5: add (%edx), %ecx  0xB7: add $0x4, %edx  0xBA: sub $0x1, %eax  0xBD: jne 0xB5 | 0x00: .long 0x0  0x04: .long 0x4  0x08: .long 0x3  0x0C: .long 0x1  0x10: .long 0x2  0x14: .long 0x1  0x18: .long 0x5 |
| --- | --- |

Traza:

| MEMORIA ACCEDIDA  0xA0  0xa6  0x00  0xAc  0x04  0xb2  0xb5  0x08  0xb7  0xba  0xbd  0xb5  0x0c  0xb7  0xba  0xbd  0xb5  0x10  0xb7  0xba  0xbd  0xb5  0x14  0xb7  0xba  0xbd | ANOTACIONES  edx = 0x0  ecx = 0x0  eax = 0x4  edx= 0x8  ecx = 0x3  edx = 0xc  eax = 0x3  ecx =0x4  edx = 0x10  eax = 0x2  ecx = 0x6  edx = 0x14  eax = 0x1  ecx = 0x7  edx = 0x18  eax = 0x0 |
| --- | --- |

3) c(10,10,12)->(20,12) supongamos que tenemos el registro de paginacion apuntando al marco fisico CR3=0xC0CA

| 0X0C0CA  —-----------------------------------  dir : marco fisico, XWRV  0x3FF: 0x0CO1A, XWRV  .  .  .  0x003: 0x0CO1A, XWRV  0x002: 0x0CO1A, XWRV  0x001:0x0CO1A, XWRV  0x000: 0x0CO1A, XWRV | 0x0C01A  —-----------------------------------  dir : marco fisico, XWRV  0x3FF: 0x0CAFE XRWV  .  0x300 : 0x0CAFE, XWR --  .  0x003: 0x\_ \_ \_ \_ \_, \_ \_ \_ \_  0x002: 0x\_ \_ \_ \_ \_, \_ \_ \_ \_  0x001: 0x\_ \_ \_ \_ \_, \_ \_ \_ \_  0x000: 0x0CAFE, XWRV | 0x0CAFE  —-----------------------------------  dir : marco fisico, XWRV  0x3FF: 0x0C0CA, –RV  .  .  .  0x003: 0x\_ \_ \_ \_ \_, \_ \_ \_ \_  0x002: 0x\_ \_ \_ \_ \_, \_ \_ \_ \_  0x001: 0x\_ \_ \_ \_ \_, \_ \_ \_ \_  0x000: 0x0060D , xwrv |
| --- | --- | --- |

1. Completar el PD y la PT para que mapee las direcciones virtuales a fisicas como sigue:

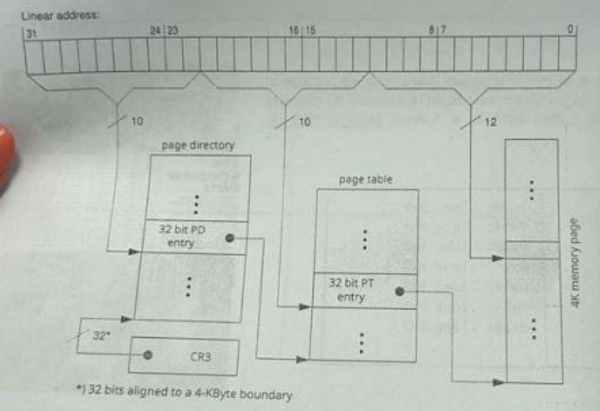
0x00000BAD ->0x0060DBAD

0xC0000FEE -> PF

1. Continuando con el mapeo, hacer que ademas mapee de virtual a fisica.

0xFFFFF000->0x0C0CA000

0xFFC00000->0x0CAFE000



1. 0x0BAD= 0X0 0X0 0XBAD

DIRECCION FISICA = marco <<3 + offset

= 0x0060D \* 4kib + offset = 0x0060D000 + 0xBAD = 0x006DBAD

0xC0000FEE = 1100000000 | 00 0000 0000 | 1111 1110 1110 = 0x300, 0x0, oxFEE

B) 0xFFFFF000->0x0C0CA000= 0xFFFFF000 = 1111111111 |111111 1111 | 0000 0000 0000 = 0X3FF, 0X3FF, 0X000

C) 0xFFC00000->0x0CAFE000 = 11 1111 1111 | 00 0000 0000 | 0000 0000 0000 = 0x 3FF. 0x0,0x00

4) Para el siguiente programa tomar atomicidad linea a linea y << es el operador de desplazamiento logico a la izq

| s=0 | | |
| --- | --- | --- |
| while(s<32){  s=s+1;  } | while(s<32){  s=s \* 2;  } | while(s<32){  s=s<<3;  } |
| s=? | | |

1. Dar todos los valores posibles de s al terminar el multiprograma (de menor a mayor).

[32, 64,256,512] Tal vez hayan mas :$

1. Agregar semaforos y su inicializacion para que siempre de el valor minimo.

| s=0; sem2 = 0; | | |
| --- | --- | --- |
| while(s<32){  s=s+1;  –  }  sem\_post(sem2)  sem\_post(sem2) | sem\_wait(sem2)  while(s<32){  —  s=s \* 2;  } | sem\_wait(sem2)  while(s<32){  –  s=s<<3;  } |
| s=? | | |

5) Escribir la solucion a la region critica usando semaforos.

sem1=1;sem2=0;

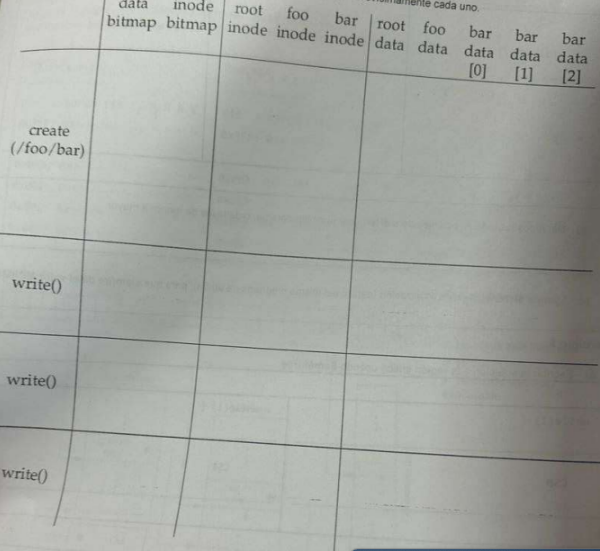
| while (1){  sem\_wait(sem1)  CS0  sem\_post(sem2)  } | while(1){  sem\_wait(sem2)  CS1  sem\_post(sem1)  } |
| --- | --- |

b)ahora usando PETErson

turn = 0 && !flag0 && !flag1

| while (1){  flag0 = true  turn =1  if(flag1&&turn=1){waits}  CS0  flag0=false;  } | while(1){  flag1 = true  turn=0  if(flag0&&turn=0){waits}  CS1  flag1 = false;  } |
| --- | --- |

6) Completar con read/write y explicar muy brevemente.



read

read