1. Considerar el siguiente fragmento de programa para dos procesos A y B:

```
Proceso A Proceso B
for a:=1 to 10  for b:= 1 to 10
  x:=x+1;     x:=x+1;
```

   Suponer que la variable `x` está inicializada a cero, que ambos procesos se ejecutan una sola vez y que las variables `a` y `b` no son compartidas. Los dos procesos pueden ejecutarse a cualquier velocidad. ¿Cuáles son los posibles valores resultantes para `x`? Para cada valor, indique la secuencia de entrelazado que lleva a él. Nota: suponga que `x` debe ser cargada en un registro para incrementarse.

Los posibles valores para la variable (compartida) x es de 10 – 20.

Para que se produzca un valor < 20, se tiene que producir un entrelazado entre las operaciones de incremento. Recordemos que la operación de incremento a nivel de maquina requiere de: $temp = LOAD(x) (cargar el valor de x), $temp = $temp +1 (sumar el inmediato 1 al valor x y guardarlo), STORE(x,$temp) (guardar el nuevo valor de x). Teniendo todo esto en cuenta:
Si partimos desde el principio (x = 0), y tanto el proceso (ligero o hebra) A como el B, realizan la carga del valor de x (LOAD(x)) y realizan también, de forma simultánea, las operaciones de incremento y de guardado, se habrá perdido uno de los incrementos debido a que tanto A como B han escrito en el valor x 1. Nótese que esto podría ocurrir en cualquiera de los momentos (las 10 veces) en los que se incrementa x. Si pasara siempre obtendríamos 10, si pasara nunca obtendríamos 20, si pasara 1 vez obtendríamos 19 y así con el resto.

2. En vez de la instrucción `Test_and_Set`, algunos ordenadores proveen de una instrucción atómica que incrementa en 1 el valor de una variable `Lock`:

```
Function {ATOMICA} TestAndInc (Var Lock: Integer):Integer;
Begin
  TestAndInc:=Lock;
  Lock:=Lock+1;
End;
```

   Escribir protocolos de entrada y salida para proteger una sección crítica utilizando dicha solución.

```
Procedure increment_task
Begin
        int n; // Numero recibido
        for a:= 1 to 10
                TestAndInc(n);
End;

Procedure principal
Begin
        int n := 0;
        cobegin
                increment_task (n); increment_task (n);
        coend;
End;
```

(Creo que es así, a confirmar por Juan Carlos).

3. Escribir un programa concurrente para multiplicar matrices de $n \times n$ con n procesos concurrentes. Suponga que los elementos de las matrices se pueden leer simultáneamente. Extienda el programa para realizar concurrentemente el cálculo del producto de ambas diagonales de una matriz.

```
Procedure principal
Begin
    int n;
    int M1[n][n];
    int M2[n][n];
    int P[n][n];
    cobegin
        mulMat(0); mulMat(1); … mulMat(n-1);
    coend;
End;

Procedure mulMat
Begin
    int i; //Numero recibido
    for j := 0 to n-1
        for k =:0 to n-1
            P[i][j] = P[i][j] + (M1[i][k] * M2[k][j]);
End;
```

Hazte una traza y veras que sale. ;)

4. Escribir un programa para calcular el producto escalar de dos vectores de n componentes. Puede utilizar tantos procesos concurrentes como sea necesario.

```
Procedure principal
Begin
        int n;
        int V1[n];
        int V2[n];
        int resultado := 0;
        int resultadosParciales[n];
        cobegin
                prodVec(0); prodVec(1); … prodVec(n-1);
        coend;
        for i:= 0 to n-1
                resultado := resultado + resultadosParciales[i]
        Writeln(resultado);
End;


Procedure prodVec
Begin
        int i; //Numero recibido
        resultadosParciales[i] = V1[i]*V2[i]
End;
```

Creo que así vale.

5. ¿Qué diferencia fundamental existe entre un bucle de espera activa y una variable interruptor que da turnos para la sincronización de procesos?

Uno tiene un bucle en el que se pueden realizar cosas mientras que la variable interruptor solo permite la alternancia entre procesos (no se).

6. ¿Utiliza la espera ocupada el Algoritmo de Dekker, si el segundo proceso está en su sección crítica, `Turno=1` y el primer proceso está intentando entrar en su sección crítica?

Preguntar si asumimos que es la versión 5ta

7. Probar que el Algoritmo de Dekker hace imposible que un proceso espere indefinidamente a entrar en su sección crítica, con la suposición de que siempre que un proceso entra en la sección crítica eventualmente la abandonará.

Como probar esto? (hacer invariantes estilo capel 46)

8. Generalizar el Algoritmo de Dekker a tres procesos.

Preguntar si asumimos que es la versión 5ta

9. ¿Qué ocurriría si el Algoritmo de Dekker se hubiera programado así?

```
Procedure Pi;
Begin
  Repeat
    Turno := i;
    While Turno<>i Do;
        Sección_Crítica;
    Turno := j;
  Forever;
End;
```

Imagina 3 procesos, el 1º ejecuta la primera instrucción y luego queda en espera activa, el proceso 2º ejecuta la primera instrucción entra en espera activa y el 1º entra a ejecutar la zona crítica (sin terminar). El 3º ejecuta la primera instrucción y queda en espera activa, entra el 2º proceso en la zona critica. PROBLEMA: Tenemos 2 procesos ejecutando distintas partes de la zona critica, así que no se cumpliría los principios de exclusión mutua y el algoritmo sería incorrecto.

(Dekker te has vuelto a equivocar?)

10. Al siguiente algoritmo se le conoce como solución de Hyman al problema de la exclusión mutua. ¿es correcta dicha solución?

```
Proceso 1 ------------
c1 := 0
while turno <> 1 do
        Begin
                While c2 = 0 do;
                Turno := 1
        End;
SeccionCritica
C1 := 1
Proceso 2 ------------
c2 := 0
while turno <> 2 do
        Begin
                While c1 = 0 do;
                Turno := 2
        End;
SeccionCritica
C2 := 1
```

Inicialmente c1,c2 y turno = 1

Plantea la siguiente traza y se saca de ejecución. P2 comienza y hace c2 = 0, entra en el bucle de turno <> 2, en el bucle de c1 = 0 NO entra porque c1 = 1, así que no entra en el bucle y se saca de ejecución antes de realizar turno = 2. P1 comienza, como turno = 1 no entra en su bucle (turno no es distinto de 1) entra a la zona critica y se bloquea. Vuelve a ejecución P2 y ejecuta Turno = 2, sale de su bucle (turno no es distinto de 2) y entra en zona crítica, de aquí en adelante P2 y P1 están en zona crítica por lo que no se cumple la condición de exclusión mutua y se puede afirmar que la solución no es correcta.

11. La instrucción EX intercambia los contenidos de dos posiciones de memoria, y es equivalente a la ejecución atómica e indivisible de las tres operaciones siguientes:

```
temp :=a;
a:=b;
b:=temp;
```

Construya un mecanismo de exclusión mutua basado en la instrucción EX.

No se a que se refiere

12. El siguiente programa es una solución al problema de la exclusión mutua para dos procesos. Discutir la corrección de la solución; si es correcta, probarlo formalmente y si no lo es, proponer una secuencia de entrelazado que lleve a romper la exclusión mutua.

```
Var c1, c2: Integer;

Procedure P1
Begin
Repeat
        Codigo
        Repeat
                c1 := 1 – c2;
        until(c2<>0)
        SeccionCritica
        c1 := 1;
Forever
End;

Procedure P2
Begin
Repeat
        Codigo
        Repeat
                c2 := 1 – c1;
        until(c1<>0)
        SeccionCritica
        c2 := 1;
Forever
End;

Begin
        c1 := 1;
        c2 := 1;
        cobegin
                P1;P2;
        conend;
End;
```

La solución NO es correcta ya que puede producirse un entrelazado de instrucciones y se puede acceder a la sección crítica. Ejemplo: P1 comienza y comienza la operación c1 := 1 – c2; (solo puede cargar el valor de c2 que es 1) y es bloqueado. Ahora, P2 realiza la operación c2 := 1 – c1; (carga el valor de c1 que es 1, lo resta a 1 y lo guarda en c2 (c2 = 0)) y se bloquea. Vuelve el turno de P1 y continua por donde lo dejo (tiene que c2 =1 y y realiza la operación y guarda su valor en c1 (c1 = 0)) ahora cuando comprueba el bucle, c2 = 0, por lo que sale del bucle, comienza a ejecutar su sección critica y se bloquea. P2 pasa a ejecución y sale del bucle (c1 = 0) por lo que entra en su sección critica (PROBLEMA: LOS DOS ESTAN EN SECCIÓN CRITICA, Por lo que no se cumple el principio de exclusión mutua y estaría mal).

13. Si se quiere pasar un semáforo como parámetro, ¿cómo habría que pasarlo, por valor o por referencia? ¿Qué ocurre si se escoge la opción equivocada?

Un semáforo debe pasarse SIEMPRE por referencia, debido a que el valor de este puede cambiar en cualquier momento de la ejecución, ya sea porque un proceso realizo signal o wait, se tiene que saber en todo momento el valor real del semáforo y no una copia de este. Si se escogiera la opción equivocada no podríamos asegurar el acceso en exclusión mutua a la sección critica y por lo tanto la solución no sería correcta.

14. Añadir un semáforo al siguiente programa de modo que siempre imprima 40.

```
Program Aumentar;
Const m=20;
Var n: Integer;

Procedure inc;
Var i :Integer;
Begin
  For y:=1 to m Do
    n:=n+1;
End;

Begin {Principal}
  n:=0;
  Cobegin
    inc;inc
  Coend;
  Writeln (n);
End.
```

```
Program Aumentar;
Const m = 20;
Var n: Integer;
Var sem: Semaphore;

Procedure inc;
var i: Integer;
Begin
        For i:=1 to m Do
                wait(sem);
                        n := n +1;
                signal(sem);
end;


Begin {Principal}
        n := 0;
        Cobegin
                inc;inc
        Coend;
        Writeln(n);
End;
```

PREGUNTAR: como define al semaforo en forma binaria (y hace falta lo de type (palma 90)) y wait y signal están bien usados?

15. ¿Funcionaría una solución de espera ocupada basada en una variable de turno si los dos procesos se ejecutaran en paralelo en dos procesadores distintos, pero con un esquema de memoria común?

Habría problemas de vivacidad, como se puede ver en el primer algoritmo de Dekker:

```
PROGRAMA UNO;
Variables
        turno: Entero;
Inicialización
        turno = 1;
        cobegin P1;P2 coend;
```

P1

```
Repeat
    Hace_Cosas();
While turno =2 Do;

REGION_CRITICA();
turno = 2;
Hace_mas_cosas();
Until Fin
```

P2

```
Repeat
    Hace_Cosas();
    While turno = 1 Do;

REGION_CRITICA();
Turno = 1;
Hace_mas_cosas();
Until Fin
```

Los procesos lentos ralentizaran a los rápidos.

16. ¿Qué cualidad han de poseer las operaciones de los semáforos para que el semáforo funcione? ¿Cómo se puede conseguir dicha cualidad al implementar semáforos en el sistema operativo?

Las operaciones de los semáforos deben realizarse de forma atómica. Grosso modo, la operación wait o signal, puede ser ejecutada por un y SOLO por un proceso, es decir, de forma indivisible. Para lograr esto, se usan operaciones primitivas de control de procesos que el propio S.O. ofrece.

17. Considérese un programa concurrente con las tareas P y Q, que se muestran a continuación. A, B, C, D y E con sentencias arbitrarias atómicas (indivisibles).

```
Task body P is
 begin
   A;
   B;
   C;
 end;
```

```
Task body Q is
  begin
    D;
    E;
  end;
```

Indicar todos las posibles entrelazados de la ejecución de los dos procesos anteriores (indicarlo por medio de "trazas"de ejecución dadas en términos de sentencias atómicas).

ABC DE
DE ABC

Pero si P comienza su ejecución y comienza y termina A, puede comenzar Q con D y parar la ejecución?

18. ¿Qué diferencia fundamental existe entre un bucle de espera activa y una variable interruptor que da turno para la sincronización de procesos?

Misma que en la 5¿?

19. El siguiente código debe imprimir como salida 50 ó 110 ¿Lo hace correctamente?

```
Program Incógnita

Var x: integer;
Task Body P1 is          Task Body P2 is
Begin                    Begin
  x:=x+10;               If x>100 then write(x) else write(x-50)
End P1;                  End P2;

Begin
  Cobegin
    P1;P2;
  Coend;
End Incógnita.
```

El programa como tal esta mal planteado, pero para corregirlo asumiremos que x comienza valiendo 100 antes del cobegin.

Aun con lo anterior, el algoritmo sería incorrecto. Se podría ver con la siguiente traza:

P2 comienza y realiza la comparación de x > 100 (se obtiene el valor de x = 100 y da false) pasa al else y se bloquea el proceso.
P1 comienza y realiza su operación (obtiene el valor de x (100), se le suma 10 y se guarda en x (x=110)).
P2 vuelve a ejecución y continua por donde lo dejo, escribiendo por consola x-50 (se obtiene el valor de x (110) y se muestra en la consola 60 (110-50).

Por lo que el ejercicio NO siempre mostrará o 50 o 110.