

Si tenemos dos procesos en ejecución, se puede decir que: (marque todas las afirmaciones correctas)

- ✓ ☒ No se sabe cuál termina primero
- ✓ ☐ Termina primero el primero que se haya lanzado
- ✓ ☒ Lo que hace uno puede afectar el resultado del otro
- ✓ ☐ Termina primero el más corto (más rápido)
- ✓ ☐ Esto solo es posible si la máquina tiene 2 cores o más

Ocultar comentario

La ejecución de un programa puede afectar el resultado de otro si utilizan los mismos recursos. El orden de ejecución lo determina el sistema operativo de acuerdo con la carga que tenga en ese momento. Si solo hay un procesador disponible, los procesos lo compartirán (sistemas de multiprogramación)

¿Cuál de las siguientes acciones **no** se considera un reto al momento de diseñar aplicaciones para sistemas multi-procesador? Por ahora piense solo en la parte funcional de la aplicación, no en aspectos como el desempeño de la misma.

- ☒ ➡ Asegurarse que hay suficientes procesadores
- ☐ Determinar si un dato debe estar compartido para que sea accedido por distintos procesos
- ☒ ✖ Decidir qué actividades pueden ejecutarse en paralelo
- ☐ Identificar la dependencia de dato entre tareas

Ocultar comentario

De lo único que como desarrolladores no tenemos que preocuparnos al momento de diseñar es del número de procesadores reales de la máquina en la que se va a ejecutar una aplicación. Esto será una restricción que permitirá al programa ejecutarse más o menos rápido, pero no es un determinante del diseño mismo a menos que se esté optimizando para lograr un cierto desempeño.

Cuando se realiza señalamiento, el proceso que debe esperar lo hace de manera pasiva

✓ ☒ Verdadero


☐ Falso

▼ [Ocultar comentario](#)

Los mecanismos de sincronización buscan no desperdiciar procesador. Por esa razón, implementan espera pasiva

Cuando un proceso invoca a V(sem) y había algún proceso esperando por el recurso controlado por sem, ¿qué transición de estado inicia el sistema operativo?

- ✓ ☒ De Dormido a Listo
- ☐ De Listo a Ejecutando
- ☐ Ninguna, se espera a terminar el quantum
- ☐ De Dormido a Ejecutando
- ☐ De Listo a Dormido
- ☐ De Ejecutando a Dormido
- ☐ De Ejecutando a Listo

 Ocultar comentario

V() indica que se liberó un recurso, se debe pasar un proceso que esté esperando (Dormido) a Listo

En DMA

- ☐ Se utilizan instrucciones específicas de E/S para iniciar una operación en un dispositivo
- ☒ Se utiliza un procesador alternativo para descargar al procesador principal de transferencias palabra por palabra
- ☐ El procesador central transfiere un bloque de datos directo de la memoria al dispositivo (o viceversa)
- ☐ Se realiza un sondeo sobre todos los dispositivos para determinar cuál fue el que completó la operación solicitada
- ☐ Todas las anteriores

[Ocultar comentario](#)

en DMA el procesador principal controla este procesador subordinado para E/S

Establezca la correspondencia entre las dos columnas . A la izquierda están restricciones de sincronización y a la derecha los nombres de dichas restricciones

- ✓ 2

Dos procesos no pueden acceder al mismo tiempo a una zona de datos

1. encuentro
- ✓ 1

Un proceso debe acceder al mismo tiempo a los datos que otro

2. exclusión mutua
- ✓ 3

Un proceso se debe ejecutar antes que otro

3. señalamiento

Ocultar comentario

En exclusión mutua no importa el orden en el que se acceda a los datos siempre y cuando no se haga al mismo tiempo.

En señalamiento el orden si importa

El encuentro exige que para poder acceder a unos datos, otro proceso también pueda.




Cuando un proceso invoca a $P()$ y no hay suficientes recursos, ¿qué transición de estado inicia el sistema operativo?

- ☐ De Listo a Dormido
- ☐ De Dormido a Listo
- ☐ De Ejecutando a Listo
- ☒ De Ejecutando a Dormido
- ☐ De Listo a Ejecutando
- ☐ De Dormido a Ejecutando
- ☐ Ninguna, se espera a terminar el quantum

☐ Ocultar comentario

Para ejecutar $P()$ debe estar en Ejecutando. Como no hay recursos disponibles debe pasar a Dormido

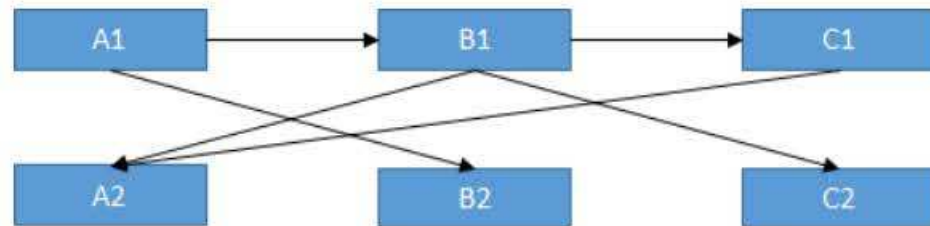
Si sucedió un evento esperado por algún proceso, seleccione todas las transiciones que suceden en el diagrama de estados de procesos

-  ☒ ☒ Dormido a Listo
- ☒ ☐ Ejecutando a Terminado
-  ☒ Listo a Ejecutando
- ☒ ☐ Listo a Dormido
-  ☒ Ejecutando a Dormido
- ☒ ☐ Ejecutando a Listo
- ☒ ☐ Dormido a Ejecutando

 [Ocultar comentario](#)

Un proceso en espera de un evento está en el estado Dormido. Cuando sucede el evento, pasa a Listo

Suponga los siguientes threads A, B, C:



Cada thread tiene dos bloques de código: 1 y 2. El origen de las flechas indica que ese bloque de código se debe ejecutar ANTES que el bloque de código destino de la flecha. Así A2, solo se puede ejecutar después de B1.

Escriba el código de A, B y C agregando las primitivas de semáforos necesarias para asegurar que la ejecución sucede según el diagrama. Indique el valor inicial para cada semáforo que utilice.

A

B

C

La inanición se evita por el uso de yield ()

 ☒ Verdadero

 ☐ Falso

 Ocultar comentario

El yield() no aporta nada para solucionar un problema de inanición

En Entrada/Salida por mapeo de puertos

☐ El procesador principal se encarga directamente de la transferencia a través de las direcciones de memoria

☐ El procesador principal delega en un DMA la transferencia desde la memoria hacia el dispositivo

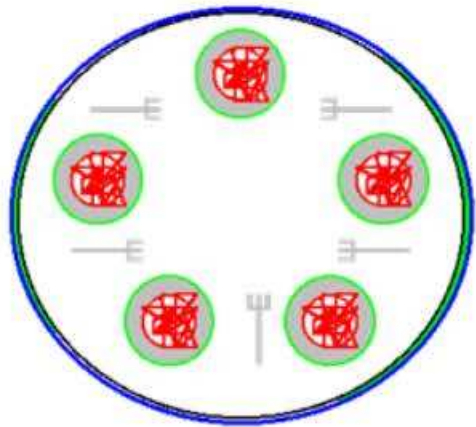
➡ ☐ El espacio de direcciones del dispositivo se utiliza para la comunicación con el dispositivo utilizando instrucciones especiales de Entrada/Salida

✗ ☒ El espacio de direcciones del dispositivo se utiliza para la comunicación con el dispositivo utilizando las instrucciones de transferencia de memoria

▼ Ocultar comentario

En mapeo de puertos se tiene un espacio de direcciones común entre la memoria y los dispositivos, pero se diferencian por las instrucciones que se utilizan

Cinco filósofos se sientan a la mesa, cada uno con un plato de espagueti. El espagueti es tan escurridizo que un filósofo necesita dos tenedores para comerlo. Entre cada dos platos hay un tenedor. En la figura se muestra la mesa. La vida de un filósofo consta de periodos alternos de comer y pensar. Cuando un filósofo tiene hambre, intenta obtener un tenedor para su mano derecha, y otro para su mano izquierda, cogiendo uno a la vez y en cualquier orden. Si logra obtener los dos tenedores, come un rato y después deja los tenedores y continúa pensando.



Suponga la siguiente solución:

```
void filosofo (int i) {           // i: id del filósofo (0 <= i <= N-1)
    while (1) {
        pensar () ;               // el filósofo está pensando
        coger_tenedores (i) ;     // obtiene los dos tenedores, espera
                                   // activamente si no puede

        comer () ;
        dejar_tenedores (i)       // Deja ambos tenedores en la mesa
    }
}
```

- (50%) ¿Esta solución funciona? Si funciona, explique el funcionamiento y las limitaciones que tiene. Si falla, indique por qué. La función `coger_tenedores()` intenta tomar un tenedor, si no está disponible, espera (activamente) hasta que el tenedor esté disponible y lo coge. Luego intenta tomar el segundo tenedor, si no puede, espera (activamente) hasta que el tenedor esté disponible y lo coge. Identifique características que deben cumplir las funciones `coger_tenedores()` y `dejar_tenedores()`. Utilice los conceptos vistos en el curso para su explicación.
- (10%) ¿Cambiaría su respuesta en función de si la máquina donde se ejecutan estos procesos es monoprocesador o multiprocesador? Explique de acuerdo a lo visto en el curso.
- (20%) ¿Cómo cambiaría su respuesta a la pregunta a) si la espera que realizan los procesos en `coger_tenedores()` es pasiva? Utilice los conceptos vistos en el curso para su explicación.
- (20%) ¿Qué pasaría si en lugar de esperar con un tenedor en la mano, liberara el tenedor que cogió, esperara cierto tiempo y volviera a intentar tomar ambos? Utilice los conceptos vistos en el curso para su explicación.

¿Cuál es la diferencia, a nivel de los estados de los procesos, entre sincronizar por espera activa o por espera pasiva?