Sistemas Operativos 1

Procesos

Concurrencia, exclusión mutua

Edwin Salvador

20 de mayo de 2015

Sesión 7

Contenido I

Deber

Examen miércoles 27 de mayo. Toda la materia hasta hoy.

- Verificar ingreso a PeerWise
  `https://peerwise.cs.auckland.ac.nz`

Deber

Examen miércoles 27 de mayo. Toda la materia hasta hoy.

- Verificar ingreso a PeerWise
  `https://peerwise.cs.auckland.ac.nz`
- Si ya se registraron deben iniciar sesión y dar clic en “join a course”.

Deber

Examen miércoles 27 de mayo. Toda la materia hasta hoy.

- Verificar ingreso a PeerWise
  `https://peerwise.cs.auckland.ac.nz`
- Si ya se registraron deben iniciar sesión y dar clic en “join a course”.
- ID del curso 11220, su identificador es su email.

Deber

Examen miércoles 27 de mayo. Toda la materia hasta hoy.

- Verificar ingreso a PeerWise
  `https://peerwise.cs.auckland.ac.nz`
- Si ya se registraron deben iniciar sesión y dar clic en "join a course".
- ID del curso 11220, su identificador es su email.
- Realizar al menos 5 preguntas en PeerWise relacionadas con la materia. Preguntas de opción múltiple. **Fecha límite domingo 25 de mayo 23:59**

Deber

Examen miércoles 27 de mayo. Toda la materia hasta hoy.

- Verificar ingreso a PeerWise `https://peerwise.cs.auckland.ac.nz`
- Si ya se registraron deben iniciar sesión y dar clic en "join a course".
- ID del curso 11220, su identificador es su email.
- Realizar al menos 5 preguntas en PeerWise relacionadas con la materia. Preguntas de opción múltiple. **Fecha límite domingo 25 de mayo 23:59**
- Responder al menos 5 preguntas (que no hayan sido realizadas por ustedes mismo) de PeerWise. **Fecha límite jueves 28 de mayo 23:59**

Deber

Examen miércoles 27 de mayo. Toda la materia hasta hoy.

- Verificar ingreso a PeerWise `https://peerwise.cs.auckland.ac.nz`
- Si ya se registraron deben iniciar sesión y dar clic en "join a course".
- ID del curso 11220, su identificador es su email.
- Realizar al menos 5 preguntas en PeerWise relacionadas con la materia. Preguntas de opción múltiple. **Fecha límite domingo 25 de mayo 23:59**
- Responder al menos 5 preguntas (que no hayan sido realizadas por ustedes mismo) de PeerWise. **Fecha límite jueves 28 de mayo 23:59**
- La nota será definida según la participación que hayan tenido en PeerWise, el tipo de preguntas que hayan realizado y el número de preguntas que hayan respondido y la dificultad de las preguntas respondidas.

Deber

- Presentar un avance del proyecto final.

Deber

- Presentar un avance del proyecto final.
- Puede ser una propuesta de lo que han investigado y como piensan estructurar su artículo y presentación.

Deber

- Presentar un avance del proyecto final.
- Puede ser una propuesta de lo que han investigado y como piensan estructurar su artículo y presentación.
- Incluir referencias más relevantes en las que basarán su investigación.

Deber

- Presentar un avance del proyecto final.
- Puede ser una propuesta de lo que han investigado y como piensan estructurar su artículo y presentación.
- Incluir referencias más relevantes en las que basarán su investigación.
- Mantener el formato de deberes.

Contenido I

Principios de la concurrencia

- La concurrencia es un factor a tomar en cuenta dentro de un SO, especialmente en tres áreas relacionadas con la gestión de procesos:
  - **multiprogramación**
  - **multiprocesamiento**
  - **procesamiento distribuido**.

Principios de la concurrencia

- La concurrencia es un factor a tomar en cuenta dentro de un SO, especialmente en tres áreas relacionadas con la gestión de procesos:
  - **multiprogramación** múltiples procesos en un monoprocesador.
  - **multiprocesamiento** múltiples procesos en un multiprocesador.
  - **procesamiento distribuido**. múltiples procesos que se ejecutan en múltiples sistemas de cómputo.
- La concurrencia influye en aspectos del diseño de un SO como:

Principios de la concurrencia

- La concurrencia es un factor a tomar en cuenta dentro de un SO, especialmente en tres áreas relacionadas con la gestión de procesos:
  - **multiprogramación** múltiples procesos en un monoprocesador.
  - **multiprocesamiento** múltiples procesos en un multiprocesador.
  - **procesamiento distribuido**. múltiples procesos que se ejecutan en múltiples sistemas de cómputo.
- La concurrencia influye en aspectos del diseño de un SO como:
  - la comunicación entre procesos

Principios de la concurrencia

- La concurrencia es un factor a tomar en cuenta dentro de un SO, especialmente en tres áreas relacionadas con la gestión de procesos:
  - **multiprogramación** múltiples procesos en un monoprocesador.
  - **multiprocesamiento** múltiples procesos en un multiprocesador.
  - **procesamiento distribuido**. múltiples procesos que se ejecutan en múltiples sistemas de cómputo.
- La concurrencia influye en aspectos del diseño de un SO como:
  - la comunicación entre procesos
  - la compartición de recursos

Principios de la concurrencia

- La concurrencia es un factor a tomar en cuenta dentro de un SO, especialmente en tres áreas relacionadas con la gestión de procesos:
  - **multiprogramación** múltiples procesos en un monoprocesador.
  - **multiprocesamiento** múltiples procesos en un multiprocesador.
  - **procesamiento distribuido**. múltiples procesos que se ejecutan en múltiples sistemas de cómputo.
- La concurrencia influye en aspectos del diseño de un SO como:
  - la comunicación entre procesos
  - la compartición de recursos
  - la competencia por recursos

Principios de la concurrencia

- La concurrencia es un factor a tomar en cuenta dentro de un SO, especialmente en tres áreas relacionadas con la gestión de procesos:
  - **multiprogramación** múltiples procesos en un monoprocesador.
  - **multiprocesamiento** múltiples procesos en un multiprocesador.
  - **procesamiento distribuido**. múltiples procesos que se ejecutan en múltiples sistemas de cómputo.
- La concurrencia influye en aspectos del diseño de un SO como:
  - la comunicación entre procesos
  - la compartición de recursos
  - la competencia por recursos
  - la sincronización de actividades de múltiples procesos

Principios de la concurrencia

- La concurrencia es un factor a tomar en cuenta dentro de un SO, especialmente en tres áreas relacionadas con la gestión de procesos:
  - **multiprogramación** múltiples procesos en un monoprocesador.
  - **multiprocesamiento** múltiples procesos en un multiprocesador.
  - **procesamiento distribuido**. múltiples procesos que se ejecutan en múltiples sistemas de cómputo.
- La concurrencia influye en aspectos del diseño de un SO como:
  - la comunicación entre procesos
  - la compartición de recursos
  - la competencia por recursos
  - la sincronización de actividades de múltiples procesos
  - la reserva de tiempo de procesador para los procesos.

Principios de la concurrencia

- Existen 3 situaciones en las que aparece la concurrencia:

Principios de la concurrencia

- Existen 3 situaciones en las que aparece la concurrencia:
  - **Múltiples aplicaciones** compartir dinámicamente el tiempo de procesador.

Principios de la concurrencia

- Existen 3 situaciones en las que aparece la concurrencia:
  - **Múltiples aplicaciones** compartir dinámicamente el tiempo de procesador.
  - **Aplicaciones estructuradas** Algunas aplicaciones pueden ser programadas como un conjunto de procesos concurrentes.

Principios de la concurrencia

- Existen 3 situaciones en las que aparece la concurrencia:
  - **Múltiples aplicaciones** compartir dinámicamente el tiempo de procesador.
  - **Aplicaciones estructuradas** Algunas aplicaciones pueden ser programadas como un conjunto de procesos concurrentes.
  - **Estructura del SO** Los SO son implementados comúnmente como un conjunto de procesos e hilos.

Términos clave de la concurrencia

- **Sección crítica:** sección de código de un proceso que requiere acceso a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.

Términos clave de la concurrencia

- **Sección crítica:** sección de código de un proceso que requiere acceso a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.
- **Interbloqueo (*deadlock*):** cuando dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno de los otros haga algo.

Términos clave de la concurrencia

- **Sección crítica:** sección de código de un proceso que requiere acceso a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.
- **Interbloqueo (*deadlock*):** cuando dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno de los otros haga algo.
- **Círculo vicioso (*livelock*):** cuando dos o más procesos cambian continuamente su estado en respuesta a cambios en los otros procesos, sin realizar ningún trabajo útil.

Términos clave de la concurrencia

- **Sección crítica:** sección de código de un proceso que requiere acceso a recursos compartidos y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.
- **Interbloqueo (*deadlock*):** cuando dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno de los otros haga algo.
- **Círculo vicioso (*livelock*):** cuando dos o más procesos cambian continuamente su estado en respuesta a cambios en los otros procesos, sin realizar ningún trabajo útil.
- **Exclusión mutua:** requisito de que cuando un proceso esté en una sección crítica que accede a recursos compartidos, ningún otro proceso pueda estar en una sección crítica que acceda a ninguno de esos recursos compartidos.

Términos clave de la concurrencia

- **Condición de Carrera** cuando múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende de la coordinación relativa de sus ejecuciones.

Términos clave de la concurrencia

- **Condición de Carrera** cuando múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende de la coordinación relativa de sus ejecuciones.
- **Inanición:** Cuando un proceso preparado para avanzar es puesto en espera indefinidamente por el **planificador**; aunque es capaz de avanzar, nunca se le escoge para ser ejecutado.

Principios de la concurrencia

- En un sistema multiprogramado monoprocesador, los procesos se entrelazan para aparentar ejecución simultánea. Esto ofrece varios beneficios pero no se consigue un procesamiento paralelo y supone cierta sobrecarga. Los sistemas multiprocesadores pueden entrelazar la ejecución y también solaparlas.
- El entrelazado y solapamiento pueden verse como ejemplos de procesamiento concurrente y ambas presentan los mismos problemas.
- Los problemas surgen de la característica básica de los sistemas multiprogramados: no puede predecirse la velocidad relativa de ejecución de los procesos. Esta velocidad depende de la actividad de otros procesos, de como el SO maneja las interrupciones, y de las políticas de planificación del SO.

Problemas presentes en sistemas multiprogramados y multiprocesador

Principios de la concurrencia

Ambos sistemas presentas los siguientes problemas:

- **La compartición de recursos globales**. Si 2 procesos utilizan la misma variable global y ambos realizan lecturas y escrituras, entonces el orden de las lecturas y escrituras es crítico.

Problemas presentes en sistemas multiprogramados y multiprocesador

Principios de la concurrencia

Ambos sistemas presentas los siguientes problemas:

- **La compartición de recursos globales**. Si 2 procesos utilizan la misma variable global y ambos realizan lecturas y escrituras, entonces el orden de las lecturas y escrituras es crítico.
- **Es complicado gestionar la asignación de recursos de manera óptima**. El proceso A puede solicitar el uso de un canal de E/S y serle concedido el control y luego ser suspendido justo antes de utilizar el canal. Si el SO bloquea el canal e impide su utilización por otros procesos puede causar una condición de interbloqueo.

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

```
void eco(){
  cent = getchar()
      ;
  csal = cent;
  putchar(csal);
}
```

- Este programa realiza un eco de un caracter; la entrada se obtiene del teclado, una tecla a la vez. Cada caracter es almacenado en `cent`, luego es transferido a `csal` y es enviado a la pantalla. Cualquier programa puede llamar repetidamente a este procedimiento para aceptar la entrada del usuario y mostrarla por pantalla.

Considerando que tenemos un sistema multiprogramado monoprocesador y monousuario

- El usuario puede saltar de una aplicación a otra y cada aplicación utiliza el mismo teclado y la misma pantalla.
- Cada aplicación necesita usar el procedimiento `eco`, este procedimiento será compartido y estará cargado en una porción de memoria global para todas las aplicaciones (solo utilizaremos una copia de este procedimiento.)

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- La compartición de memoria principal entre procesos es útil para permitir una interacción eficiente entre procesos. Sin embargo esta interacción puede acarrear problemas. Consideremos la siguiente secuencia:

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- La compartición de memoria principal entre procesos es útil para permitir una interacción eficiente entre procesos. Sin embargo esta interacción puede acarrear problemas. Consideremos la siguiente secuencia:
  1. `P1` invoca a `eco` y es interrumpido inmediatamente después de que `getchar` devuelva su valor y sea almacenado en `cent`. En este punto en caracter introducido más recientemente, `x`, está almacenado en `cent`.

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- La compartición de memoria principal entre procesos es útil para permitir una interacción eficiente entre procesos. Sin embargo esta interacción puede acarrear problemas. Consideremos la siguiente secuencia:
  1. `P1` invoca a `eco` y es interrumpido inmediatamente después de que `getchar` devuelva su valor y sea almacenado en `cent`. En este punto en caracter introducido más recientemente, `x`, está almacenado en `cent`.
  2. `P2` se activa, invoca a `eco` y se ejecuta hasta concluir, habiendo leído y mostrado en pantalla un único caracter `y`.

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- La compartición de memoria principal entre procesos es útil para permitir una interacción eficiente entre procesos. Sin embargo esta interacción puede acarrear problemas. Consideremos la siguiente secuencia:
  1. `P1` invoca a `eco` y es interrumpido inmediatamente después de que `getchar` devuelva su valor y sea almacenado en `cent`. En este punto en caracter introducido más recientemente, `x`, está almacenado en `cent`.
  2. `P2` se activa, invoca a `eco` y se ejecuta hasta concluir, habiendo leído y mostrado en pantalla un único caracter `y`.
  3. `P1` retoma. En este instante, el valor de `x` ha sido sobrescrito en `cent` y por tanto se ha perdido. En su lugar, `cent` contiene `y` y es transferido a `csal` y mostrado en pantalla.

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- La compartición de memoria principal entre procesos es útil para permitir una interacción eficiente entre procesos. Sin embargo esta interacción puede acarrear problemas. Consideremos la siguiente secuencia:
  1. `P1` invoca a `eco` y es interrumpido inmediatamente después de que `getchar` devuelva su valor y sea almacenado en `cent`. En este punto en caracter introducido más recientemente, `x`, está almacenado en `cent`.
  2. `P2` se activa, invoca a `eco` y se ejecuta hasta concluir, habiendo leído y mostrado en pantalla un único caracter `y`.
  3. `P1` retoma. En este instante, el valor de `x` ha sido sobrescrito en `cent` y por tanto se ha perdido. En su lugar, `cent` contiene `y` y es transferido a `csal` y mostrado en pantalla.
- ¿Qué problema tenemos aquí?

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- La compartición de memoria principal entre procesos es útil para permitir una interacción eficiente entre procesos. Sin embargo esta interacción puede acarrear problemas. Consideremos la siguiente secuencia:
  1. `P1` invoca a `eco` y es interrumpido inmediatamente después de que `getchar` devuelva su valor y sea almacenado en `cent`. En este punto en caracter introducido más recientemente, `x`, está almacenado en `cent`.
  2. `P2` se activa, invoca a `eco` y se ejecuta hasta concluir, habiendo leído y mostrado en pantalla un único caracter `y`.
  3. `P1` retoma. En este instante, el valor de `x` ha sido sobrescrito en `cent` y por tanto se ha perdido. En su lugar, `cent` contiene `y` y es transferido a `csal` y mostrado en pantalla.
- ¿Qué problema tenemos aquí? el primer caracter se pierde y el segundo es mostrado 2 veces. Esto se da porque la variable `cent` es global y compartida, múltiples procesos tienen acceso a esta variable.

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- Podemos solucionar este problema si permitimos que solo un proceso pueda estar en `eco`. Entonces la secuencia anterior sería:

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- Podemos solucionar este problema si permitimos que solo un proceso pueda estar en `eco`. Entonces la secuencia anterior sería:
  1. `P1` invoca a `eco` y es interrumpido....

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- Podemos solucionar este problema si permitimos que solo un proceso pueda estar en `eco`. Entonces la secuencia anterior sería:
  1. `P1` invoca a `eco` y es interrumpido....
  2. `P2` se activa e invoca a `eco`. Pero como `P1` aún está en `eco` (aunque suspendido), `P2` no puede entrar a `eco` y por lo tanto se suspende hasta que le procedimiento `eco` esté disponible.

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- Podemos solucionar este problema si permitimos que solo un proceso pueda estar en `eco`. Entonces la secuencia anterior sería:
  1. `P1` invoca a `eco` y es interrumpido....
  2. `P2` se activa e invoca a `eco`. Pero como `P1` aún está en `eco` (aunque suspendido), `P2` no puede entrar a `eco` y por lo tanto se suspende hasta que le procedimiento `eco` esté disponible.
  3. `P1` retoma y completa la ejecución de `eco` y se muestra el caracter x

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- Podemos solucionar este problema si permitimos que solo un proceso pueda estar en `eco`. Entonces la secuencia anterior sería:
  1. `P1` invoca a `eco` y es interrumpido....
  2. `P2` se activa e invoca a `eco`. Pero como `P1` aún está en `eco` (aunque suspendido), `P2` no puede entrar a `eco` y por lo tanto se suspende hasta que le procedimiento `eco` esté disponible.
  3. `P1` retoma y completa la ejecución de `eco` y se muestra el caracter x
  4. `P1` sale de `eco`, se elimina el bloqueo de `P2`. Y este puede acceder a `eco`.

Ejemplo de problemas de concurrencia

Sistema multiprogramado monoprocesador monousuario

- Podemos solucionar este problema si permitimos que solo un proceso pueda estar en `eco`. Entonces la secuencia anterior sería:
  1. `P1` invoca a `eco` y es interrumpido....
  2. `P2` se activa e invoca a `eco`. Pero como `P1` aún está en `eco` (aunque suspendido), `P2` no puede entrar a `eco` y por lo tanto se suspende hasta que le procedimiento `eco` esté disponible.
  3. `P1` retoma y completa la ejecución de `eco` y se muestra el caracter x
  4. `P1` sale de `eco`, se elimina el bloqueo de `P2`. Y este puede acceder a `eco`.
- Este ejemplo nos sirve para demostrar que los problemas de concurrencia se pueden dar incluso con un sistema monoprocesador.

Ejemplo de problemas de concurrencia

Sistema multiprocesador

- En un sistema multiprocesador aparecen los mismos problemas de recursos compartidos protegidos y podemos utilizar las mismas soluciones.
- Si seguimos la misma secuencia que en el ejemplo anterior con un multiprocesador tendríamos:
  1. `P1` y `P2` se ejecutan cada uno en un procesador distinto, ambos invocan a `eco`.
  2. Ocurren los siguientes eventos (los eventos en la misma línea suceden en paralelo):

Proceso P1	**Proceso P2**
•	•
cent = getchar();	•
•	cent = getchar();
csal = cent;	csal = cent;
putchar(csal);	•
•	putchar(csal);
•	•

El resultado es que el caracter introducido por `P1` se pierde antes de ser mostrado y el caracter introducido por `P2` es mostrado por ambos.

Ejemplo de problemas de concurrencia

Sistema multiprocesador

- Si de nuevo añadimos la condición de que un solo un proceso puede acceder a `eco`:

Ejemplo de problemas de concurrencia

Sistema multiprocesador

- Si de nuevo añadimos la condición de que un solo un proceso puede acceder a `eco`:
  1. **P1** y **P2** se ejecutan en un procesadores distintos y ambos invocan a `eco`.

Ejemplo de problemas de concurrencia

Sistema multiprocesador

- Si de nuevo añadimos la condición de que un solo un proceso puede acceder a `eco`:
  1. **P1** y **P2** se ejecutan en un procesadores distintos y ambos invocan a `eco`.
  2. Mientras `P1` está dentro de `eco`, `P2` invoca a `eco`. Como `P1` está dentro, a `P2` se le bloquea la entrada al procedimiento. Por tanto, `P2` se suspende a la espera de la disponibilidad de `eco`.

Ejemplo de problemas de concurrencia

Sistema multiprocesador

- Si de nuevo añadimos la condición de que un solo un proceso puede acceder a `eco`:
  1. **P1** y **P2** se ejecutan en un procesadores distintos y ambos invocan a `eco`.
  2. Mientras `P1` está dentro de `eco`, `P2` invoca a `eco`. Como `P1` está dentro, a `P2` se le bloquea la entrada al procedimiento. Por tanto, `P2` se suspende a la espera de la disponibilidad de `eco`.
  3. Cuando `P1` completa la ejecución de `eco`, sale del procedimiento y continúa ejecutando e inmediatamente `P2` comienza la ejecución de `eco`.

Contenido I

Condición de carrera

- Qué es?

Condición de carrera

- Qué es? Cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Veamos dos ejemplos:

Condición de carrera

- Qué es? Cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Veamos dos ejemplos:
- **Ejemplo 1:** `P1` y `P2` comparten la variable global `a`. En algún punto `P1` actualiza `a` al valor 1 y en el mismo punto en su ejecución `P2` actualiza `a` al valor 2. Así, las dos tareas están compitiendo por escribir la variable `a`. El proceso que actualiza último (el perdedor) determina el valor de `a`.

Condición de carrera

- Qué es? Cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Veamos dos ejemplos:
- **Ejemplo 1:** `P1` y `P2` comparten la variable global `a`. En algún punto `P1` actualiza `a` al valor 1 y en el mismo punto en su ejecución `P2` actualiza `a` al valor 2. Así, las dos tareas están compitiendo por escribir la variable `a`. El proceso que actualiza último (el perdedor) determina el valor de `a`.
- **Ejemplo 2:** `P3` y `P4` comparten las variables `b` y `c`, con valores iniciales `b = 1` y `c = 2`. En algún punto `P3` ejecuta la asignación `b = b + c` y en algún punto `P4` ejecuta la asignación `c = b + c`. Los valores finales dependen del orden en que los procesos ejecuten estas dos asignaciones.

Condición de carrera

- Qué es? Cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Veamos dos ejemplos:
- **Ejemplo 1:** `P1` y `P2` comparten la variable global `a`. En algún punto `P1` actualiza `a` al valor 1 y en el mismo punto en su ejecución `P2` actualiza `a` al valor 2. Así, las dos tareas están compitiendo por escribir la variable `a`. El proceso que actualiza último (el perdedor) determina el valor de `a`.
- **Ejemplo 2:** `P3` y `P4` comparten las variables `b` y `c`, con valores iniciales `b = 1` y `c = 2`. En algún punto `P3` ejecuta la asignación `b = b + c` y en algún punto `P4` ejecuta la asignación `c = b + c`. Los valores finales dependen del orden en que los procesos ejecuten estas dos asignaciones.
  - Cuáles serían los valores si `P3` ejecuta su sentencia primero?

Condición de carrera

- Qué es? Cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Veamos dos ejemplos:
- **Ejemplo 1:** `P1` y `P2` comparten la variable global `a`. En algún punto `P1` actualiza `a` al valor 1 y en el mismo punto en su ejecución `P2` actualiza `a` al valor 2. Así, las dos tareas están compitiendo por escribir la variable `a`. El proceso que actualiza último (el perdedor) determina el valor de `a`.
- **Ejemplo 2:** `P3` y `P4` comparten las variables `b` y `c`, con valores iniciales `b = 1` y `c = 2`. En algún punto `P3` ejecuta la asignación `b = b + c` y en algún punto `P4` ejecuta la asignación `c = b + c`. Los valores finales dependen del orden en que los procesos ejecuten estas dos asignaciones.
  - Cuáles serían los valores si `P3` ejecuta su sentencia primero? `b = 3` y `c = 5`

Condición de carrera

- Qué es? Cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Veamos dos ejemplos:
- **Ejemplo 1:** `P1` y `P2` comparten la variable global `a`. En algún punto `P1` actualiza `a` al valor 1 y en el mismo punto en su ejecución `P2` actualiza `a` al valor 2. Así, las dos tareas están compitiendo por escribir la variable `a`. El proceso que actualiza último (el perdedor) determina el valor de `a`.
- **Ejemplo 2:** `P3` y `P4` comparten las variables `b` y `c`, con valores iniciales `b = 1` y `c = 2`. En algún punto `P3` ejecuta la asignación `b = b + c` y en algún punto `P4` ejecuta la asignación `c = b + c`. Los valores finales dependen del orden en que los procesos ejecuten estas dos asignaciones.
  - Cuáles serían los valores si `P3` ejecuta su sentencia primero? `b = 3` y `c = 5`
  - Cuáles serían los valores si `P4` ejecuta su sentencia primero?

Condición de carrera

- Qué es? Cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos. Veamos dos ejemplos:
- **Ejemplo 1:** `P1` y `P2` comparten la variable global `a`. En algún punto `P1` actualiza `a` al valor 1 y en el mismo punto en su ejecución `P2` actualiza `a` al valor 2. Así, las dos tareas están compitiendo por escribir la variable `a`. El proceso que actualiza último (el perdedor) determina el valor de `a`.
- **Ejemplo 2:** `P3` y `P4` comparten las variables `b` y `c`, con valores iniciales `b = 1` y `c = 2`. En algún punto `P3` ejecuta la asignación `b = b + c` y en algún punto `P4` ejecuta la asignación `c = b + c`. Los valores finales dependen del orden en que los procesos ejecuten estas dos asignaciones.
  - Cuáles serían los valores si `P3` ejecuta su sentencia primero? `b = 3` y `c = 5`
  - Cuáles serían los valores si `P4` ejecuta su sentencia primero? `b = 4` y `c = 3`

Contenido I

Requisitos del SO para manejar la concurrencia

- Los aspectos de diseño y gestión que surgen por la concurrencia son:

Requisitos del SO para manejar la concurrencia

- Los aspectos de diseño y gestión que surgen por la concurrencia son:
  1. El SO debe ser capaz de seguir la pista de varios procesos. Utilizando el BCP.

Requisitos del SO para manejar la concurrencia

- Los aspectos de diseño y gestión que surgen por la concurrencia son:
  1. El SO debe ser capaz de seguir la pista de varios procesos. Utilizando el BCP.
  2. El SO debe ubicar y desubicar varios recursos para cada proceso activo, incluyendo: tiempo de procesador, memoria, ficheros, dispositivos E/S.

Requisitos del SO para manejar la concurrencia

- Los aspectos de diseño y gestión que surgen por la concurrencia son:
  1. El SO debe ser capaz de seguir la pista de varios procesos. Utilizando el BCP.
  2. El SO debe ubicar y desubicar varios recursos para cada proceso activo, incluyendo: tiempo de procesador, memoria, ficheros, dispositivos E/S.
  3. El SO debe proteger los datos y recursos físicos de cada proceso frente a interferencias involuntarias de otros procesos.

Requisitos del SO para manejar la concurrencia

- Los aspectos de diseño y gestión que surgen por la concurrencia son:
  1. El SO debe ser capaz de seguir la pista de varios procesos. Utilizando el BCP.
  2. El SO debe ubicar y desubicar varios recursos para cada proceso activo, incluyendo: tiempo de procesador, memoria, ficheros, dispositivos E/S.
  3. El SO debe proteger los datos y recursos físicos de cada proceso frente a interferencias involuntarias de otros procesos.
  4. El funcionamiento de un proceso y el resultado que produzca, debe ser independiente de la velocidad a la que suceda su ejecución en relación con la velocidad de otros procesos concurrentes.

Contenido I

Interacción de procesos

Podemos clasificar las formas de interacción entre procesos en base al grado en que perciben la existencia de de los otros. Así tenemos:

- **Procesos que no se perciben entre sí** Son independientes y no se pretende que trabajen juntos. Multiprogramación de múltiples procesos independientes. Aunque no trabajen juntos, el SO debe ocuparse de la **competencia** por recursos. Dos aplicaciones independientes pueden necesitar el mismo disco, impresora o fichero.

Interacción de procesos

Podemos clasificar las formas de interacción entre procesos en base al grado en que perciben la existencia de de los otros. Así tenemos:

- **Procesos que no se perciben entre sí** Son independientes y no se pretende que trabajen juntos. Multiprogramación de múltiples procesos independientes. Aunque no trabajen juntos, el SO debe ocuparse de la **competencia** por recursos. Dos aplicaciones independientes pueden necesitar el mismo disco, impresora o fichero.
- **Procesos que se perciben indirectamente entre sí** No están necesariamente al tanto de la presencia de los demás mediante sus respectivos ID de proceso, pero comparten accesos a algún objeto como bufer de E/S. Estos **cooperan** en la compartición del objeto.

Interacción de procesos

Podemos clasificar las formas de interacción entre procesos en base al grado en que perciben la existencia de de los otros. Así tenemos:

- **Procesos que no se perciben entre sí** Son independientes y no se pretende que trabajen juntos. Multiprogramación de múltiples procesos independientes. Aunque no trabajen juntos, el SO debe ocuparse de la **competencia** por recursos. Dos aplicaciones independientes pueden necesitar el mismo disco, impresora o fichero.
- **Procesos que se perciben indirectamente entre sí** No están necesariamente al tanto de la presencia de los demás mediante sus respectivos ID de proceso, pero comparten accesos a algún objeto como bufer de E/S. Estos **cooperan** en la compartición del objeto.
- **Procesos que se perciben directamente entre sí** Son capaces de comunicarse entre sí vía el ID de proceso y son diseñados para trabajar conjuntamente en cierta actividad. Estos también exhiben **cooperación**.

Competencia entre procesos por recursos

Interacción de procesos

- Los procesos concurrentes entran en conflicto entre ellos cuando compiten por el mismo recurso.
- Cuando los procesos compiten entre sí, ninguno debe verse afectado por la ejecución de otros procesos. Es decir, cada proceso debe dejar inalterado el estado de los recursos (dispositivos E/S, memoria, tiempo de procesador).
- La ejecución de un proceso puede afectar el comportamiento de los procesos en competencia. Es decir, si dos procesos desean ambos el mismo recurso, el SO reservará el recurso para uno de ellos y el otro tendrá que esperar, por lo tanto será ralentizado (en ciertos casos el proceso bloqueado puede no conseguir nunca el recurso y por tanto nunca terminar).

Problemas de control en procesos en competencia

Los procesos en competencia deben afrontar tres problemas: exclusión mutua, interbloqueo e inanición.

- **Exclusión mutua**:

Problemas de control en procesos en competencia

Los procesos en competencia deben afrontar tres problemas: exclusión mutua, interbloqueo e inanición.

- **Exclusión mutua**:
  - Si dos o más procesos requieren acceso a un recurso no compartible como una impresora (**recurso crítico**), es importante que solo se permita un programa en su **sección crítica**.

Problemas de control en procesos en competencia

Los procesos en competencia deben afrontar tres problemas: exclusión mutua, interbloqueo e inanición.

- **Exclusión mutua**:
  - Si dos o más procesos requieren acceso a un recurso no compartible como una impresora (**recurso crítico**), es importante que solo se permita un programa en su **sección crítica**.
  - No podemos dejar que el SO entienda y aplique esta restricción porque en ciertos casos los detalles de requisitos no son obvios.

Problemas de control en procesos en competencia

Los procesos en competencia deben afrontar tres problemas: exclusión mutua, interbloqueo e inanición.

- **Exclusión mutua**:
  - Si dos o más procesos requieren acceso a un recurso no compartible como una impresora (**recurso crítico**), es importante que solo se permita un programa en su **sección crítica**.
  - No podemos dejar que el SO entienda y aplique esta restricción porque en ciertos casos los detalles de requisitos no son obvios.
  - En el caso de una impresora, un proceso debe tener el control total mientras imprime un fichero. Caso contrario, las líneas de los procesos se intercalarían.

Problemas de control en procesos en competencia

Los procesos en competencia deben afrontar tres problemas: exclusión mutua, interbloqueo e inanición.

- **Exclusión mutua**:
  - Si dos o más procesos requieren acceso a un recurso no compartible como una impresora (**recurso crítico**), es importante que solo se permita un programa en su **sección crítica**.
  - No podemos dejar que el SO entienda y aplique esta restricción porque en ciertos casos los detalles de requisitos no son obvios.
  - En el caso de una impresora, un proceso debe tener el control total mientras imprime un fichero. Caso contrario, las líneas de los procesos se intercalarían.
  - La exclusión mutua genera dos problemas: interbloqueo e inanición.

Problemas de control en procesos en competencia

- **Interbloqueo:** Si tenemos dos procesos P1 y P2 y dos recursos R1 y R2. Si los dos procesos necesitan acceder a **ambos** recursos para realizar parte de su función, entonces es posible la siguiente situación:

Problemas de control en procesos en competencia

- **Interbloqueo:** Si tenemos dos procesos P1 y P2 y dos recursos R1 y R2. Si los dos procesos necesitan acceder a **ambos** recursos para realizar parte de su función, entonces es posible la siguiente situación:
  - El SO asigna R1 a P2 y R2 a P1.

Problemas de control en procesos en competencia

- **Interbloqueo:** Si tenemos dos procesos P1 y P2 y dos recursos R1 y R2. Si los dos procesos necesitan acceder a **ambos** recursos para realizar parte de su función, entonces es posible la siguiente situación:
  - El SO asigna R1 a P2 y R2 a P1.
  - Cada proceso está esperando por uno de los recursos y ninguno liberará el recurso que posee hasta conseguir el otro recurso y realizar la tarea que requiere ambos recursos.

Problemas de control en procesos en competencia

- **Interbloqueo:** Si tenemos dos procesos P1 y P2 y dos recursos R1 y R2. Si los dos procesos necesitan acceder a **ambos** recursos para realizar parte de su función, entonces es posible la siguiente situación:
  - El SO asigna R1 a P2 y R2 a P1.
  - Cada proceso está esperando por uno de los recursos y ninguno liberará el recurso que posee hasta conseguir el otro recurso y realizar la tarea que requiere ambos recursos.
  - Ambos procesos están interbloqueados.

Problemas de control en procesos en competencia

- **Inanición:** Tenemos tres procesos P1, P2 y P3 que requieren todos accesos periódicos al recurso R. Podemos tener la siguiente situación:

Problemas de control en procesos en competencia

- **Inanición:** Tenemos tres procesos P1, P2 y P3 que requieren todos accesos periódicos al recurso R. Podemos tener la siguiente situación:
  - P1 está en posesión del recurso y P2 y P3 están ambos esperando por ese recurso.

Problemas de control en procesos en competencia

- **Inanición:** Tenemos tres procesos P1, P2 y P3 que requieren todos accesos periódicos al recurso R. Podemos tener la siguiente situación:
  - P1 está en posesión del recurso y P2 y P3 están ambos esperando por ese recurso.
  - Cuando P1 termine su sección critica, debería permitírsele el acceso a R a P2 o P3.

Problemas de control en procesos en competencia

- **Inanición:** Tenemos tres procesos P1, P2 y P3 que requieren todos accesos periódicos al recurso R. Podemos tener la siguiente situación:
  - P1 está en posesión del recurso y P2 y P3 están ambos esperando por ese recurso.
  - Cuando P1 termine su sección critica, debería permitírsele el acceso a R a P2 o P3.
  - Supongamos que el SO le concede el acceso a P3 y que P1 solicita acceso otra vez antes de completar su sección crítica.

Problemas de control en procesos en competencia

- **Inanición:** Tenemos tres procesos P1, P2 y P3 que requieren todos accesos periódicos al recurso R. Podemos tener la siguiente situación:
  - P1 está en posesión del recurso y P2 y P3 están ambos esperando por ese recurso.
  - Cuando P1 termine su sección critica, debería permitírsele el acceso a R a P2 o P3.
  - Supongamos que el SO le concede el acceso a P3 y que P1 solicita acceso otra vez antes de completar su sección crítica.
  - Si el SO le concede acceso a P1 después de que P3 haya terminado y luego concede acceso alternativamente a P1 y P3, entonces P2 puede denegársele indefinidamente el acceso al recurso aunque no suceda un interbloqueo.

Mecanismo de la exclusión mutua

- El SO es quien ubica los recursos y este debe estar involucrado en el control de la competencia por recursos.
- Tenemos *n* procesos para ser ejecutados concurrentemente. Cada proceso tiene: una sección crítica y código anterior y posterior a la sección crítica.
- Para aplicar exclusión mutua se necesitan dos funciones: `entrarcritica` y `salircritica` las cuales toman como argumento el nombre del recurso sujeto de la competencia `Ra`.
- Un proceso que intente entrar en su sección crítica mientras otro proceso está en su sección crítica, por el mismo recurso, se le hace esperar.

```
/* PROCESO 1 */
void P1
{
    while (true)
    {
        /* código anterior */;
        entrarcritica (Ra);
        /* sección crítica */;
        salircritica (Ra);
        /* código posterior */;
    }
}
```

```
/* PROCESO 2 */
void P2
{
    while (true)
    {
        /* código anterior */;
        entrarcritica (Ra);
        /* sección crítica */;
        salircritica (Ra);
        /* código posterior */;
    }
}
```

. . .

```
/* PROCESO n */
void Pn
{
    while (true)
    {
        /* código anterior */;
        entrarcritica (Ra);
        /* sección crítica */;
        salircritica (Ra);
        /* código posterior */;
    }
}
```

Cooperación entre procesos vía compartición

Interacción de procesos

- **Cubre procesos que interaccionan con otros procesos sin tener conocimiento explícito de ellos.**
- Por ejemplo, muchos procesos pueden tener acceso a ficheros o bases de datos compartidas.
- Los procesos pueden usar y actualizar los datos compartidos sin referenciar otros procesos pero saben que otros procesos pueden acceder a los mismos datos.
- Así, los procesos deben cooperar para asegurar la integridad de los datos compartidos.
- Los problemas de exclusión mutua, interbloqueo e inanición también están presentes ya que los datos están contenidos en recursos (dispositivos, memoria, archivos). Pero en este caso los datos individuales pueden ser accedidos para lectura y escritura, y solo las operaciones de escritura deben ser mutuamente exclusivas.

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- A más de los problemas anteriores, surge un nuevo problema: **la coherencia de datos**.

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- A más de los problemas anteriores, surge un nuevo problema: **la coherencia de datos**.
- **Ejemplo**: Consideremos una aplicación de contabilidad en la que pueden ser actualizados varios datos individuales. Supongamos que dos datos individuales `a` y `b` deben mantener la relación `a = b`. Cualquier programa que actualice un valor debe también actualizar el otro. Si tenemos dos procesos:

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- A más de los problemas anteriores, surge un nuevo problema: **la coherencia de datos**.
- **Ejemplo**: Consideremos una aplicación de contabilidad en la que pueden ser actualizados varios datos individuales. Supongamos que dos datos individuales `a` y `b` deben mantener la relación `a = b`. Cualquier programa que actualice un valor debe también actualizar el otro. Si tenemos dos procesos:

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- A más de los problemas anteriores, surge un nuevo problema: **la coherencia de datos**.
- **Ejemplo**: Consideremos una aplicación de contabilidad en la que pueden ser actualizados varios datos individuales. Supongamos que dos datos individuales `a` y `b` deben mantener la relación `a = b`. Cualquier programa que actualice un valor debe también actualizar el otro. Si tenemos dos procesos:

P1:

```
a = a + 1;
b = b + 1;
```

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- A más de los problemas anteriores, surge un nuevo problema: **la coherencia de datos**.
- **Ejemplo**: Consideremos una aplicación de contabilidad en la que pueden ser actualizados varios datos individuales. Supongamos que dos datos individuales `a` y `b` deben mantener la relación `a = b`. Cualquier programa que actualice un valor debe también actualizar el otro. Si tenemos dos procesos:

P1:

```
a = a + 1;
b = b + 1;
```

P2:

```
b = 2 * b;
a = 2 * a;
```

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- A más de los problemas anteriores, surge un nuevo problema: **la coherencia de datos**.
- **Ejemplo**: Consideremos una aplicación de contabilidad en la que pueden ser actualizados varios datos individuales. Supongamos que dos datos individuales `a` y `b` deben mantener la relación `a = b`. Cualquier programa que actualice un valor debe también actualizar el otro. Si tenemos dos procesos:

P1:

```
a = a + 1;
b = b + 1;
```

P2:

```
b = 2 * b;
a = 2 * a;
```

- Si el estado inicial es consistente, cada proceso por separado dejará los datos compartidos en un estado consistente.

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- Ahora consideremos que los procesos se ejecutan **concurrentemente** y los dos procesos respetan la exclusión mutua sobre cada dato `a` y `b`.

```
a = a + 1;
b = 2 * b;
b = b + 1;
a = 2 * a;
```

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- Ahora consideremos que los procesos se ejecutan **concurrentemente** y los dos procesos respetan la exclusión mutua sobre cada dato `a` y `b`.

```
a = a + 1;
b = 2 * b;
b = b + 1;
a = 2 * a;
```

- Que pasará al final de esta secuencia si `a = b = 1`?

.

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- Ahora consideremos que los procesos se ejecutan **concurrentemente** y los dos procesos respetan la exclusión mutua sobre cada dato `a` y `b`.

```
a = a + 1;
b = 2 * b;
b = b + 1;
a = 2 * a;
```

- Que pasará al final de esta secuencia si `a = b = 1`? la condición `a = b` no se mantiene. Tendremos `a = 4` y `b = 3`.

Problema de coherencia de datos

Cooperación entre procesos vía compartición

- Ahora consideremos que los procesos se ejecutan **concurrentemente** y los dos procesos respetan la exclusión mutua sobre cada dato `a` y `b`.

```
a = a + 1;
b = 2 * b;
b = b + 1;
a = 2 * a;
```

- Que pasará al final de esta secuencia si `a = b = 1`? la condición `a = b` no se mantiene. Tendremos `a = 4` y `b = 3`.
- Esto puede ser evitado declarando en cada proceso la secuencia completa como una sección crítica.

Cooperación entre procesos vía comunicación

- En los casos anteriores, cada proceso tiene su propio entorno aislado que no incluye a los otros procesos, es decir, las interacciones entre procesos son indirectas.
- Cuando los procesos cooperan vía comunicación, los procesos involucrados participan en un esfuerzo común que los vincula a todos ellos. La comunicación proporciona una manera de sincronizar o coordinar actividades varias.
- La comunicación es básicamente el paso de mensajes entre los procesos.
- Ya que en el paso de mensajes los procesos no comparten nada, la exclusión mutua no es un requisito de control en este tipo de cooperación. Pero los problemas de inanición e interbloqueo si están presentes (en este caso con referencia a la comunicación).

Contenido I

Requisitos para la exclusión mutua

Cualquier técnica que vaya a proporcionar exclusión mutua debería cumplir los siguientes requisitos:

1. Sólo se permite un proceso dentro de su sección crítica.

Requisitos para la exclusión mutua

Cualquier técnica que vaya a proporcionar exclusión mutua debería cumplir los siguientes requisitos:

1. Sólo se permite un proceso dentro de su sección crítica.
2. Un proceso que se pare en su sección **no** crítica debe hacerlo sin interferir con otros procesos.

Requisitos para la exclusión mutua

Cualquier técnica que vaya a proporcionar exclusión mutua debería cumplir los siguientes requisitos:

1. Sólo se permite un proceso dentro de su sección crítica.
2. Un proceso que se pare en su sección **no** crítica debe hacerlo sin interferir con otros procesos.
3. No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente: ni interbloqueo ni inanición.

Requisitos para la exclusión mutua

Cualquier técnica que vaya a proporcionar exclusión mutua debería cumplir los siguientes requisitos:

1. Sólo se permite un proceso dentro de su sección crítica.
2. Un proceso que se pare en su sección **no** crítica debe hacerlo sin interferir con otros procesos.
3. No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente: ni interbloqueo ni inanición.
4. Si ningún proceso está en una sección crítica, y un proceso solicita entrar en su sección crítica debe permitírsele entrar sin demora.

Requisitos para la exclusión mutua

Cualquier técnica que vaya a proporcionar exclusión mutua debería cumplir los siguientes requisitos:

1. Sólo se permite un proceso dentro de su sección crítica.
2. Un proceso que se pare en su sección **no** crítica debe hacerlo sin interferir con otros procesos.
3. No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente: ni interbloqueo ni inanición.
4. Si ningún proceso está en una sección crítica, y un proceso solicita entrar en su sección crítica debe permitírsele entrar sin demora.
5. No se deben hacer suposiciones de las velocidades de ejecución de los procesos ni sobre el número de procesadores.

Requisitos para la exclusión mutua

Cualquier técnica que vaya a proporcionar exclusión mutua debería cumplir los siguientes requisitos:

1. Sólo se permite un proceso dentro de su sección crítica.
2. Un proceso que se pare en su sección **no** crítica debe hacerlo sin interferir con otros procesos.
3. No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente: ni interbloqueo ni inanición.
4. Si ningún proceso está en una sección crítica, y un proceso solicita entrar en su sección crítica debe permitírsele entrar sin demora.
5. No se deben hacer suposiciones de las velocidades de ejecución de los procesos ni sobre el número de procesadores.
6. Un proceso debe permanecer en su sección crítica sólo por un tiempo finito.