Puesto que en modo usuario no se pueden tratar las interrupciones, no existen interrupciones de reloj disponibles para establecer el cuanto de un hilo. Éste se ejecuta hasta que voluntariamente decide ceder el uso del procesador o expira el cuanto del proceso. En dicho caso el sistema operativo debe seleccionar otro proceso para ser ejecutado.

Si se soportan hilos del núcleo, el sistema operativo planifica estos hilos usando algún determinado algoritmo de planificación. El planificador puede tener en cuenta consideraciones relativas al rendimiento del sistema a la hora de planificar hilos del núcleo de igual prioridad.

En sistemas con una configuración del tipo menor número de hilos del núcleo que hilos de usuarios, aparte de las dos planificaciones comentadas, es necesario implementar con la biblioteca de planificación local para establecer qué hilo de usuario puede usar un hilo del núcleo.

Preguntas de autoevaluación tema 4

4.1. Explicar cuándo se produce y en qué consiste el problema de la condición de carrera.

Se producen cuando varios procesos acceden a un recurso compartido, con lo cual en vez de cooperar compiten por el ya que el resultado final de la ejecución depende del orden en que se hayan planificado los procesos, es en que se hayan ejecutado las instrucciones .de lectura y escritura de cada proceso sobre el recurso.

Surge cuando múltiples procesos independientes se ejecutan concurrentemente y acceden para leer o escribir en un recurso compartido.

4.2. ¿Qué es una sección crítica?

Una instrucción o conjunto de instrucciones secuenciales del código de un proceso que requieren manipular un recurso compartido (variable, dato, fichero, …) con otros procesos. Dentro del código de un proceso pueden existir varias secciones críticas.

4.3. ¿En qué consiste la exclusión mutua?

Que el uso de un recurso por parte de un proceso excluya su uso para los restantes.

¿Qué problema permite resolver?

Las condiciones de carrera.

4.4. Explicar brevemente los problemas que puede producir el uso de la exclusión mutua si no se implementa correctamente.

Puede producir interbloqueo y la inanición de procesos.

El interbloqueo o *bloqueo mutuo* de dos o más procesos, se produce por ejemplo cuando dos procesos A y B necesitan dos recursos RI y R2 para realizar una cierta función. Si A posee RI y B posee R2 o puede progresar, ambos se quedan esperando por el recurso que necesitan, y no pueden liberar el recurso que poseen hasta obtener el otro recurso y realizar la función correspondiente.

La *inanición de un proceso* se produce cuando un proceso no puede progresar al necesitar un recurso al que nunca llega a acceder porque el sistema operativo siempre le relega dando prioridad a otros procesos.

4.5. ¿Qué requisitos debe cumplir el mecanismo o técnica que se utilice para implementar la exclusión mutua?

- ***Garantizar que solo un proceso puede estar a la vez dentro de una sección crítica asociada a un recurso R.*** Sólo se puede ejecutar a la vez una única sección crítica asociada a un cierto recurso compartido R.
- ***No puede realizar suposiciones sobre la velocidad relativa y prioridad de ejecución de los procesos concurrentes.***
- ***Garantizar que un proceso que no está ejecutando una sección crítica asociada a un recurso R no impide o bloquea el acceso a R a otros procesos.***
- ***Asegurar que ningún proceso se quede esperando indefinidamente a poder acceder a una sección crítica asociada a un recurso R.*** Es decir, debe evitar la inanición de procesos. Obviamente esto requiere que un proceso no debe permanecer dentro de su sección crítica por un tiempo ilimitado.
- ***Si ningún proceso está dentro de una sección crítica asociada a un recurso R, entonces si un proceso desea entrar en una sección crítica se le debe conceder rápidamente.***

4.6. ¿Por qué se caracterizan las dos soluciones software a las exclusiones mutuas descritas en el texto?

Porque no presuponen el apoyo de sistema operativo o del hardware del computador para su implementación.

¿Cuál es su principal inconveniente?

Que cualquier proceso que desea entrar en una sección crítica, si está siendo ejecutada por otro proceso, debe esperar a

que éste termine mediante la ejecución de un bucle que comprueba el valor de una condición basada en el valor de una o varias variables. Esta variable hace las veces de *cerrojo* (lock).

4.7. ¿Cómo se puede lograr la exclusión mutua usando un cerrojo y alternancia estricta?

En esta solución (código pag.125) se considera la existencia de una variable global de acceso *turno*, que se usa como cerrojo y que puede tomar como valor el identificador numérico de un proceso. En este caso el proceso A tiene el identificador 0 y el proceso B el identificador 1. Nótese que el valor al que se inicializa el cerrojo determina qué proceso puede entrar en primer lugar en su sección crítica, en este caso es el proceso A.
Cuando un proceso (A o B) quiere entrar en su sección crítica comprueba el valor de *turno*. Si coincide con su identificador numérico entonces puede ejecutar su sección crítica, en caso contrario deberá esperar. La espera que realiza el proceso es una espera activa ya que entra en un bucle para comprobar el valor de *turno*, del que no sale hasta que *turno* toma como valor el identificador numérico del proceso que está esperando.
Una vez que turno toma el valor del identificador numérico del proceso que desea entrar en su sección crítica, éste podrá ejecutar dicha sección. Cuando finalice debe configurar el valor de *turno* con el identificador numérico del otro proceso concurrente. En consecuencia, el acceso a la sección crítica se va alternando entre los procesos A y B. Hasta que un proceso no finaliza la ejecución de su sección crítica, el otro proceso no puede ejecutar la suya.

¿Qué problemas presenta esta solución software?

La velocidad de ejecución de los procesos A y B viene limitada por la velocidad de ejecución del proceso más lento.

Si un proceso falla (por ejemplo A) antes de poder cambiar el valor de turno al identificador del otro proceso (B), entonces éste (B) permanecerá bloqueado indefinidamente.

4.8. ¿Qué acciones se realizan en la función *acceso_sc* del algoritmo de Peterson?

En primer lugar se calcula el valor de *otro_pid*, es decir, el identificador asociado al otro proceso concurrente que se está ejecutando, en este caso el proceso B. Para ello realiza la operación *otro_pid== l-pid*, luego en este caso *otro_pid== 1*
A continuación el proceso A expresa su deseo de acceder al recurso compartido poniendo el valor TRUE en el elemento 0 del

vector de dos elementos enteros *petición_rec*. Luego asigna a la variable *turno* un valor igual al identificador numérico del otro proceso, en este caso 1. Finalmente entra en un bucle while en espera de poder usar el recurso compartido. Dicho bucle se ejecutará mientras se cumplan las siguientes dos condiciones:
1. *turno==otro pid*. La variable *turno* coincida con el identificador del otro proceso que se está ejecutando concurrentemente. En este caso el proceso es B luego turno vale 1.
2. *petición_rec [otro_pid] ==TRUE.* El otro proceso (en este caso B) haya solicitado previamente usar el recurso compartido.

¿Y en la función *salida_sc*?

Lo único que hace es desactivar la petición de uso del recurso por parte del proceso invocador; para lograrlo configura con el valor FALSE el elemento asociado al proceso invocador en el vector *petición _ rec*.

4.9. ¿Por qué el algoritmo de Peterson evita la posibilidad de interbloqueo de procesos?

Por que el proceso que primero ejecuta la instrucción *peticion_rec [id_proceso] =TRUE* se garantiza el uso del recurso. El otro proceso tendrá que esperar.

Nunca puede suceder que los dos procesos se queden bloqueados en espera de entrar en su sección crítica, ya que si A está bloqueado a la fuerza turno=l y en consecuencia el proceso B podrá entrar en su sección crítica.

4.10. Cuando se utiliza el algoritmo de Peterson ¿puede un proceso monopolizar el uso de un recurso?

No puede monopolizar el uso de un recurso ejecutando repetidas veces sección crítica ya que siempre concede una oportunidad de entrar al otro proceso al ejecutar la instrucción *turno=otro_pid.*

4.11. ¿Cómo se garantiza la exclusión mutua mediante el uso de instrucciones máquina especiales?

Las instrucciones especiales garantizan que si un procesador PI está accediendo a una posición de memoria ningún otro procesador acceder a dicha posición hasta que PI haya terminado de usar dicha posición de memoria.
Estas instrucciones máquina especiales realizan acciones tales como leer y comprobar, o leer y escribir una posición de memoria determinada. Se caracterizan por ser *atómicas,* es decir, se ejecutan en un ciclo de instrucción que no puede ser interrumpido,

por lo que su ejecución siempre se completa. De esta forma se garantiza el acceso exclusivo a la posición de memoria accedida.

¿Cuáles son sus principales inconvenientes?

- ***La espera que realiza un proceso para entrar en su sección crítica activa***. La espera activa supone un problema si se dilata en exceso en el tiempo.
- ***Puede producir inanición de procesos.***
- ***Puede presentar esta solución es la aparición de interbloqueos.***

4.12. ¿Cómo se garantiza la exclusión mutua mediante el uso del bloqueo de las interrupciones?

Al permitir que un proceso A, antes de entrar región crítica, ejecute una instrucción especial para bloquear el mecanismo de interrupciones del sistema, y cuando salga de la región crítica ejecute otra instrucción especial que lo vuelva a habilitar.

¿Cuáles son sus principales inconvenientes? (Respuesta en sección 4.2.6)

El rendimiento del sistema se puede degradar bastante al no permitirle atender las interrupciones más prioritarias en el momento en que llegan. Además se deja en manos del proceso la decisión de ceder el procesador y si éste nunca lo devuelve el sistema se quedará' colgado.

Esta solución no sirve en el caso de sistemas multiprocesadores, ya que un proceso B, que se estuviera ejecutando en un procesador P2 distinto al procesador P1 en el que ejecuta al proceso A podría acceder a una sección critica aunque estén deshabilitadas las interrupciones, ya que esto solo garantiza que el proceso A no puede ser expulsado del procesador P1, pero no que se ejecuten otros procesos en otros procesadores.

4.13. ¿Qué son los semáforos?

Son un mecanismo de sincronización de procesos concurrentes gestionado por los sistemas operativos.

4.14. Describir el funcionamiento de las operaciones *init_sem, wait _sem, signal_sem* cuando se aplican sobre un semáforo general y cuando se aplican sobre un semáforo binario.

Semáforo general.

- ***init_sem.*** Inicializa el semáforo S con el estado N.
- ***wait_sem.***

  Disminuye en una unidad el valor del semáforo. Es decir, S = S -1. Si el valor resultante es un número negativo entonces el

proceso que ha invocado esta operación es añadido a la cola de procesos bloqueados asociada al semáforo y se bloquea.

- ***signal_sem*.**
  Incrementa en una unidad el valor del semáforo, es decir, S = S + 1. Si el valor resultante es menor o igual a O entonces se elimina de la cola asociada al semáforo uno de los procesos bloqueados, y se le despierta, lo que hace que pase al estado preparado para ejecución

Semáforo binario.

- ***init_sem.*** Inicializa el semáforo S con el estado N.
- ***wait_sem (S).*** Esta operación comprueba el valor del semáforo S. Si S=0, entonces el proceso es colocado en la cola de procesos bloqueados asociada al semáforo y se bloquea. Si S= 1, entonces pone el semáforo a O y el proceso puede continuar su ejecución.
- ***signal_ sem (S).*** Esta operación comprueba si la cola de procesos bloqueados asociada al semáforo S está vacía. En caso afirmativo, pone el semáforo a 1, y continúa su ejecución. En caso negativo, es decir, hay procesos bloqueados en el semáforo, entonces el sistema operativo elimina de la cola asociada al semáforo a uno de los procesos bloqueados, y le despierta lo que hace que pase al estado preparado para ejecución.

4.15. ¿Cómo implementa el sistema operativo las operaciones sobre un semáforo?

El sistema operativo se encarga de asignar las estructuras de dato necesarias para los semáforos y de realizar las operaciones sobre ellos.
El núcleo implementa las operaciones wait_sem y signal_sem como primitivas o funciones elementales de su código que se ejecutan de forma atómica, es decir, se ejecutan en un único ciclo de instrucción que no puede ser interrumpido, por lo que su ejecución siempre se completa.
El núcleo tiene que garantizar que dos procesos no pueden realizar simultáneamente dos operaciones sobre un mismo semáforo.
Luego en la implementación de las primitivas del núcleo de las operaciones wait_sem y signal_sem se ha de resolver un problema de exclusión mutua.
En sistemas con un único procesador la exclusión mutua se garantiza bloqueando las interrupciones cuando se está ejecutando una operación sobre el semáforo.

4.16. Describir cómo se usan los semáforos para obtener el acceso con exclusión mutua a un recurso.

Hay utilizar tantos semáforos como recursos distintos compartan, a través de regiones críticas, los procesos recurrentes. Supóngase, por simplificar, que únicamente hay un recurso cuyo acceso se va a regular con un semáforo S. El semáforo se inicializa al, init_sem (S,1). Cuando un proceso desea entrar en una región crítica asociada a dicho recurso realiza una operación wait_sem (S). Si el valor de S es 1, entonces S se hace 0 y el proceso puede acceder inmediatamente a la sección crítica. A partir de momento cualquier proceso que desee entrar en su sección crítica e invoque una operación wait_sem (S) producirá que se decremente en una unidad el semáforo y se quede bloqueado en la cola de dicho semáforo.
Cuando el proceso que estaba dentro de su sección finalice entonces ejecutará signa1_sem (S) que incrementará en una unidad el valor del semáforo y desbloqueará a uno de los procesos bloqueado en la cola del semáforo.

4.17. Describir cómo se usan los semáforos para sincronizar procesos.

En el caso de dos procesos A y B basta con usar un semáforo S que se inicializa a 0, init_sem (S,0). Si el proceso A invoca una operación wait_sem (S), S toma el valor -1 (o 0 si era un semáforo binario) y se bloquea en la cola del semáforo en espera de que el proceso B complete alguna operación. Cuando el proceso B finaliza dicha operación ejecuta una operación signa1_sem (S) para notificárselo proceso A. Como resultado de dicha operación el semáforo S toma el valor 0, el proceso A se desbloquea y pasa al estado preparado para ejecución.

4.18. ¿Qué problemas potenciales se pueden producir debido a un mal uso de los semáforos?

La colocación inadecuada de las operaciones wait_sem y signal_ sem pueden producir errores de temporización y de funcionamiento (condición de carrera, inanición o interbloqueo) que pueden ser difíciles de detectar, ya que solo se producen bajo determinadas secuencias de ejecución, las cuales no se dan siempre.

4.19. ¿Qué es un monitor?

Es un módulo software, que consta de un conjunto de procedimientos, variables y estructuras de datos, definido en algunos lenguajes de alto nivel que posee la propiedad especial de que solo permite a un único proceso simultáneamente ejecutar alguno de sus procedimientos. Además, las variables contenidas dentro del monitor solo son accesibles por los procedimientos del monitor y no por procedimientos externos.

4.20. ¿Qué utilidad tienen las variables de condición de un monitor?

Cada variable de condición tiene asociada una cola de procesos bloqueados en espera de que se cumpla dicha condición.

¿Tienen asociada algún tipo cola?

Si, una cola de procesos asociada a la condición X.

4.21. Describir el funcionamiento de las operaciones *wait_mon y signal_mon* de un monitor.

wait_mon(X). El proceso dentro del monitor que realiza esta operación queda suspendido en una cola de procesos asociada al cumplimiento de la condición X. En consecuencia otro procedimiento puede entrar en el monitor.

signal_mon (X). Comprueba si la cola de procesos asociada a la condición X contiene algún proceso bloqueado. En caso afirmativo se desbloquea a un proceso. Si la cola estaba vacía esta operación no tiene ningún efecto. Si no existe ningún proceso en la cola de condición la señal de aviso se pierde.

4.22. Explicar qué sucede con un proceso que invoca una operación *signal_mon* según la solución de:

a) Hoare.

El proceso invocador A de la operación signa1_mon se bloquea y se permite ejecutar en monitor al proceso B desbloqueado por dicha operación. Cuando B finalice un procedimiento o se bloquee en una condición entonces el proceso A se desbloqueará.

b) Hansen.

El proceso invocador de la operación signa1_mon sale del monitor inmediatamente. Luego esta operación aparecería como la sentencia final del procedimiento de un monitor.

c) Lampson y Redell.

El proceso A invocador de la operación signa1_mon continua su ejecución hasta finalizar el procedimiento del monitor o

bloquearse en una condición. Luego se retomará la ejecución del proceso B desbloqueado por signa1_mon.

4.23. Describir cómo se obtiene en un monitor el acceso con exclusión mutua a un recurso.

Un monitor garantiza implícitamente la exclusión mutua sobre recursos compartidos por varios procesos concurrentes. Simplemente hay que definir dentro del monitor los recursos compartidos.

4.24. Describir cómo se obtiene en un monitor la sincronización de procesos.

Mediante el uso de las variables de condición y de las operaciones wait_mon y signa1_mon. Por lo tanto si un monitor está bien definido, la sincronización de los procesos está garantizada.

4.25. ¿Qué es el paso de mensajes?

Es un mecanismo de sincronización y comunicación entre procesos soportado por los sistemas operativos tanto de sistemas centralizados como distribuidos.

4.26. ¿Qué es un mensaje?

Es un conjunto de información que puede ser intercambiada entre un proceso emisor y un proceso receptor.

¿Cuáles son las dos operaciones básicas que se pueden realizar sobre un mensaje?

send(destino, mensaje)
receive(fuente, mensaje).

4.27. ¿Cuáles son los aspectos básicos en el diseño del mecanismo de paso de mensajes?

La especificación de la fuente y el destino del mensaje.
El esquema de sincronización o, el formato del mensaje.
El medio de almacenamiento de los mensajes.

4.28. Explicar cómo se realiza la comunicación directa mediante paso de mensajes.

El proceso emisor del mensaje especifica explícitamente en la operación send el proceso al que va dirigido el mensaje, send(P_receptor, mensaje). Por su parte, el proceso receptor del mensaje especifica explícitamente en la operación receive el proceso emisor del mensaje, receive(P_emisor, mensaje).

Otra posibilidad, es que el proceso receptor del mensaje especifique implícitamente en la operación receive el proceso emisor del mensaje, receive(ident, mensaje). Donde ident toma el valor del identificador del proceso con el que se ha realizado la comunicación, el cual no se sabe a priori, sino una vez que se completa la operación receive.

4.29. Explicar cómo se realiza la comunicación indirecta mediante paso de mensajes.

El proceso emisor envía el mensaje a una estructura de datos compartida denominada *buzón* que se implementa como una *cola de mensajes.* Por su parte, el proceso receptor solicita recibir el mensaje de dicho buzón. Además previamente a las operaciones de emisión o recepción un proceso (emisor o receptor) tiene que solicitar mediante una llamada al sistema la creación del buzón.

¿Qué esquemas de difusión de mensajes posibilita? (Respuesta en sección 4.6.2)

- **Un emisor - un receptor**. Un proceso emisor envía un mensaje a un buzón para que lo recoja determinado proceso receptor.
- **Un emisor - varios receptores**. Un emisor envía un mensaje a un buzón para que lo puedan leer varios receptores.
- **Varios emisores - un receptor.** Varios procesos emisores envían mensajes a un buzón, que en este caso se suele denominar *puerto,* para que los lea un único proceso receptor.
- **Varios emisores - varios receptores.** Varios procesos emisores envían mensajes a un buzón para que sean leídos por varios procesos receptores.

4.30. Describir el funcionamiento de un buzón que sea propiedad de:

a) Un proceso.

Entonces se implementa en el espacio de direcciones de dicho proceso. En este caso solo el proceso propietario puede recibir mensajes de dicho buzón. Cuando el proceso propietario finaliza entonces el buzón desaparece. Si algún proceso posteriormente envía mensajes a este buzón debe ser notificado de la no existencia del mismo.

b) El sistema operativo.

Entonces se implementa en el espacio de direcciones del núcleo, por lo que es independiente de cualquier proceso. En este

caso el sistema operativo debe proporcionar llamadas al sistema para crear y borrar el buzón. El proceso que crea un buzón es considerado su propietario. Inicialmente solo el propietario puede leer mensajes del buzón pero dicho privilegio se puede compartir con otros procesos mediante las llamadas al sistema adecuadas. En consecuencia pueden existir varios receptores.

4.31. Enumerar los cuatro esquemas de sincronización posibles en el mecanismo de paso de mensajes.

Operación send con bloqueo. El proceso emisor pasa al estado bloqueado hasta que el mensaje sea recibido por el proceso receptor o por el buzón.
Operación send sin bloqueo. El proceso emisor envía el mensaje a un proceso receptor o a un buzón y continúa su ejecución. Un proceso, por error o intencionadamente, puede estar enviando continuamente mensajes, lo que repercute en el rendimiento del sistema ya que se están consumiendo tiempo de procesador y espacio de almacenamiento.
Operación receive con bloqueo. El proceso receptor pasa al estado bloqueado en espera de recibir un mensaje de un proceso emisor o de que exista algún mensaje en el buzón. El receptor puede quedarse indefinidamente bloqueado si el proceso emisor finaliza antes de enviar el mensaje o si el mensaje se pierde en el caso de un sistema distribuido
Operación receive sin bloqueo. El proceso receptor obtiene de un proceso emisor o de un buzón un mensaje válido o uno nulo y continua su ejecución. Su principal desventaja es que si se ejecuta antes que llegue el mensaje, éste puede perderse.

4.32. Describir las partes de un mensaje.

La *cabecera del mensaje* contiene información de control del mensaje como por ejemplo: el tipo y la prioridad del mensaje, identificadores del destino y origen del mensaje, la longitud o tamaño del mensaje, punteros a otros mensajes de la cola de mensajes, etc.
El *cuerpo del mensaje* contiene el contenido propiamente dicho del mensaje.

4.33. Describir las principales ventajas e inconvenientes relativas a que la longitud de un mensaje sea fija o variable.

Los *mensajes de longitud fija* son más fáciles de implementar para un sistema operativo, ya que se almacenan en bufferes de tamaño fijo, lo que hace más sencilla y eficaz su asignación. Su

principal desventaja es que complican a los programadores la creación de los procesos emisores ya que los mensajes largos deben ser divididos y enviados en varios mensajes del tamaño de mensaje soportado, lo que puede producir problemas de secuenciamiento con los procesos receptores.

Los *mensajes de longitud variable* facilitan al programador la programación de los procesos emisores. Sin embargo, su implementación a nivel del sistema operativo es más compleja, ya que ahora tiene que asignar trozos de memoria de tamaño variable, según el tamaño de cada mensaje, lo cual aumenta la sobrecarga y puede producir problemas de fragmentación de memoria.

4.34. Describir los mecanismos de mensajes posibles en función de la capacidad de las colas de mensajes.

- ***Mecanismo de mensajes sin buffer.*** Las colas de mensajes son de capacidad nula, por lo que lo mensajes nunca pueden esperar. El proceso emisor se bloquea hasta que el receptor recibe el mensaje.
- ***Mecanismo de mensajes con buffer.*** Las colas de mensajes permiten almacenar temporalmente un número finito de mensajes. Cada vez que llega un mensaje se añade a la cola y cada vez que un proceso lee un mensaje se borra (salvo que se especifique lo contrario). Si la cola está llena, el proceso emisor se bloquea hasta que exista espacio disponible en la cola.

4.35. Supuesto que se dispone de la operación *send* sin bloqueo, de la operación *receive* con bloqueo y que la comunicación es indirecta a través de un buzón. Describir cómo se puede obtener mediante paso de mensajes:

a) La sincronización de procesos.

El mensaje que se va enviar no requiere tener ningún contenido.

Para conseguir la sincronización el proceso A debe ejecutar una operación receive cuando debe esperar por una acción que debe realizar el proceso B. Por su parte, el proceso B debe ejecutar una operación send cuando termine de ejecutar dicha acción.
Si se ejecuta antes la operación receive del proceso A que la operación send del proceso B, entones el proceso A se quedará bloqueado a la espera de la recepción del mensaje. En caso contrario el proceso A continuará su ejecución, eso significa que el proceso B ya ha realizado la acción por la que iba esperar el proceso A, y en consecuencia no tiene necesidad de esperar.

b) El acceso con exclusión mutua a un recurso.

La cola de mensajes se inicializa fuera del código de los procesos con un mensaje que no requiere tener ningún contenido y que se denotará como nulo.

Un proceso que desea ejecutar su sección crítica primero debe ejecutar una operación receive para recibir un mensaje. Si el buzón está vacío entonces se bloquea hasta que llegue un mensaje a la cola. Cuando dicho evento ocurra entonces el proceso obtiene el mensaje (que se supone que se borra de cola) y puede sea entrar en su sección crítica. Cuando sale ejecuta una operación send para enviar el mensaje de nuevo a la cola, y permitir que otro proceso que estuviera bloqueado en la cola de mensajes pueda desbloquearse y acceder a su región crítica.

Preguntas de autoevaluación tema 5

5.1. ¿Qué es un interbloqueo?

Aquella situación en la cual un conjunto de procesos está bloqueado en espera de la liberación de uno o varios recursos que se encuentran asignados a otro proceso del mismo conjunto. Como todos los procesos del conjunto están bloqueados, ninguno de ellos liberará los recursos que posee y que otro proceso necesita, y en consecuencia ninguno puede continuar su ejecución.

5.2. Enumerar y explicar las cuatro condiciones necesarias y suficientes para la existencia del interbloqueo.

- ***Exclusión mutua.*** Cada instancia de un recurso solo puede ser asignada a un proceso como máximo.
- ***Retención y espera.*** Cada proceso retiene los recursos que le han sido asignado mientras espera por la adquisición de los otros recursos que necesita.
- ***No existencia de expropiación.*** Si un proceso posee un recurso, éste no se le puede expropiar.
- ***Espera circular.*** Existe una cadena circular de dos o más procesos, de tal forma que cada proceso de la cadena se encuentra esperando por un recurso retenido por el siguiente proceso de la cadena.

5.3. Enumerar las principales estrategias que puede implementar un sistema operativo para el tratamiento de los interbloqueos.