Parcial 2020

1. Describa las diferencias entre un cambio de modo y un cambio de contexto en un sistema operativo. ¿Por qué el sistema operativo necesita realizar esos cambios?

Un cambio de contexto ocurre cuando, por necesidad del proceso que se ejecutando (por ej, necesita realizar una E/S) o porque lo determina el planificador (por ej, se termino su tiempo de uso de procesador), se debe frenar la ejecucion del proceso y comenzar ejecutar otro proceso restaurando su PCB, en otras palabras, se cambia el estado de los procesos(?).
Por otro lado, un cambio de modo ocurre cuando se necesita cambiar los privilegios de un proceso entre modo usuario y modo kernel (por ej, cuando ocurre una interrupcion se cambia a modo kernel por si el codigo de procesamiento de la interrupcion tiene instrucciones privilegiadas).

El cambio de contexto es necesario para que el CPU pueda ejecutar multiples programas a la vez y guardar o restaurar el contexto en el que se estaba ejecutando la ultima vez que utilizo el procesador (el contexto incluye su estado, su PC, sus punteros a memoria, etc)
El cambio de modo es necesario para que solo ciertos procesos tengan acceso completo al procesador, sus instrucciones, sus registros y memoria, con el fin de mantener la seguridad del SO y evitar interferencias de los programas de usuario.
Por ej, las instrucciones relacionadas con el manejo de memoria solo deberian ser utilizadas por procesos que esten en modo kernel.

1. Si tiene un sistema operativo que está diseñado para una sola CPU y desea adaptarlo a un sistema de múltiples CPU, describa los tipos de cambios / adiciones al software o hardware que necesitaría realizar en relación con la administración de procesos y por qué.

Para adaptar el diseño de una sola CPU a uno de multiples CPU, debe decidir si tener una cola de listos comun o agregarle a cada CPU su propia cola privada.
En el caso de elegir la segunda se debe agregar algun mecanismo de balanceo de carga para evitar que algunos procesos tengan mucha carga y otros poca. Esto se lo puede realizar a traves de push migration (una tarea es enviada de un procesador a otro) o de pull migration (un procesador inactivo se trae una tarea de otro procesador mas cargado).
Relacionado al balanceo de carga, se agrega la necesidad de definir una politica de afinidad de procesador ya que es costoso cambiar a una tarea de procesador (se debe vaciar la cache del procesador del cual se va y llenar la cache del procesador al cual llega). En la politica soft affinity el SO intenta mantener al un proceso corriendo en el mismo procesador pero no garantiza que lo hara, de forma contraria, en la politica hard affinity se puede especificar al proceso en que procesadores se puede ejecutar a traves de system calls.

(G – 220 hasta 225)

2. En relación con los threads es cierto que: (Justifique su respuesta)

1. Cuando se crea un thread, su código hay que cargarlo desde el fichero ejecutable a memoria.
2. Cada thread tiene su propia pila.
3. Todos los threads que existen en el sistema en un momento dado, comparten el mismo código.
4. Descomponer un proceso en dos threads sólo puede ejecutarse más rápido si el sistema cuenta, al menos, con dos procesadores.

1) Falso, porque el codigo del hilo es compartido con el proceso que lo creo (y por lo tanto, el fichero ejecutable) entonces cuando el hilo es creado, utiliza ese mismo codigo que ya fue cargado.

2) Verdadero, ya que cada hilo llama a distintos procedimientos por lo que cada uno tiene su historial de ejecucion, entonces, es necesario que cada hilo tenga su propia pila. (T – 105)

3) Falso, los hilos solo comparten el codigo del proceso al que pertencen, es decir, cada grupo de hilos que pertenece a un proceso comparte la seccion de codigo del mismo, pero otro grupo de hilos que pertenece a otro proceso distinto comparte otro codigo. Por lo tanto, es falso que todos los hilos que existen comparten el mismo codigo.

4) Falso. Ejemplo: uno de los dos hilos necesita realizar una E/S, entonces, se realiza un cambio de contexto y comienza a ejecutarse el segundo hilo. Si no hubiera dos hilos, todo el proceso hubiera quedado en espera fuera de ejecucion debido a que se llego a un punto donde se necesitaba de una E/S, en cambio, como si tenia dos hilos, mientras la otra parte esperaba, la segunda parte seguia ejecutandose. De esta manera, gracias a la concurrencia, a pesar de tener un solo procesador, si puede ejecutarse un proceso mas rapido.

1.- Explique cómo se puede implementar un semáforo. Brinde un escenario dónde sea más eficiente utilizar spinlock que locks bloqueantes.

Un semaforo se puede implementar a traves de la espera ocupada, es decir, cuando un proceso no puede pasar el semaforo, queda ciclando hasta poder acceder a decrementar el semaforo, esto claramente genera un gasto innecesario del tiempo del CPU que podria ser aprovechado por otros procesos. Una mejor alternativa es implementar el semaforo sin esta espera ocupada, de manera que cuando un proceso no pueda decrementar un semaforo, se suspenda a si mismo, sea colocado en una cola de espera asociada a dicho semaforo y su estado cambie de corriendo a en espera. Luego, el planificador elige otro proceso para ejecutar y no se desperdicia tiempo de CPU.

(G – 274)

A diferencia de un lock bloqueante, un spin lock tiene como ventaja que no necesita realizar un cambio de contexto, por lo tanto, en un sistema multicore puede resultar en una mayor eficiencia en caso que el lock sea utilizado por poco tiempo, menos que el que tomaria realizar un cambio de contexto.
Por ej: un proceso da vueltas en un core y el otro proceso esta en la seccion critica, este ultimo termina rapidamente y el primero accede a la seccion critica, todo sin haber necesitado un cambio de contexto, lo cual hubiera resultado mas costoso.

(G – 272)

2. ¿Cuál de las siguientes políticas de planificación es más adecuada para un sistema de multiproceso de tiempo compartido? Justifique su respuesta

1. Primero el trabajo más corto.
2. Round-Robin.
3. Prioridades.
4. FIFO.

La planificacion mas adeducado es Round-Robin ya que al haber multiples usuarios utilizando el sistema, todos pueden ir ejecutando sus tareas de manera equitativa sin que algunos usuarios monopolicen el uso del CPU ya sea porque sus procesos llegaron antes o por tener mayor prioridad.
Por lo tanto, el reparto del tiempo del procesador se reparte de manera justo y no hay posibilidad de inanicion, aunque, para que el rendimiento sea optimo, se debe elegir un tiempo de quantum adecuado, ni muy grande ni muy chico.

1.- ¿Puede un sistema detectar inanición? Justifique su respuesta, si es Sí explique cómo lo puede realizar, si es No explique cómo puede manejar este problema.

La inanicion es un problema que surge principalemnte en la planificacion por prioridades y no puede ser detectada por el sistema. Sin embargo, a este problema se lo puede manejar a traves del “envejecimiento” de los procesos, es decir, cuando un proceso esta demasiado tiempo esperando para ejecutar y no lo consigue (debido a que otros procesos con mayor prioridad son seleccionados para ejecutar) se le aumenta la prioridad hasta que consiga ser ejecutado.

1.- Suponga que tiene un sistema con cuatro hilos (threads), una variable compartida y un lock compartido. El código se escribió de tal manera que cualquier acceso de un hilo a la variable compartida siempre tendría el lock en ese hilo para su acceso. ¿Puede este sistema tener una condición de carrera que involucre la variable compartida? Justifica tu respuesta

Un lock permite que un hilo acceda al mismo y lo bloquee para que ningun otro hilo puede acceder a dicho lock hasta que el hilo que lo accedio lo libere. Por lo tanto, este sistema no puede tener una condicion de carrera que involucre a la variable compartida ya que, como se describe en el enunciado, cada vez que se quiere acceder a dicha variable se debe obtener el control del lock que la “protege”, es decir, pueden modificar/leer la variable de a uno a la vez respetando la exclusion mutua, lo cual permite evitar la condicion de carrera.

2.-Suponga dos hilos con una variable compartida x que ejecutan las siguientes instrucciones

P1	P2
x = 0;	x = 0;
x = x + 1;	x = x + 1;
x = x + 1;	x = x + 1;
x = x - 1;	x = x – 1;
x = x - 1;	x = x – 1;
if (x==1)	if (x==1)
Printf("Hola ...")	Printf("Hola ...")

¿Este código puede mostrar el mensaje "Hola ..."? Si la respuesta es afirmativa, Ilustre su secuencia numerando las instrucciones con el orden en que se ejecutan. Si es negativa, explique su respuesta.

Si, es posible

Instruccion	**Proceso**	**Valor X**
x=0	P1	0
Cambia la ejecucion a P2		
x=0	P2	0
x=x+1	P2	1
x=x+1	P2	2
Cambia la ejecucion a P1		
x=x+1	P1	3
x=x+1	P1	4
x=x-1	P1	3
Cambia la ejecucion a P2		
x=x-1	P2	2
x=x-1	P2	1
If(x==1)	P2	1
printf("Hola")	P2	1

1.- Suponga que la operación Signal () en una variable de condición no garantiza despertar solo un hilo, y no garantiza que los despierte en orden FIFO. ¿Podría esto afectar a los programas que utilizan variables de condición? Explica tu respuesta, brinde un ejemplo de un programa que fallaría usando esta semántica más débil.

Si puede afectar a los programas que utilizan variables de condicion ya que puede perderse el orden en el que debian suceder ciertos eventos o realizarse ciertas tareas.
Por ejemplo:
(aclaracion: cv = condition variable)

P1	P2
cv → wait() lock → acquire() …. lock → release()	cv → wait() lock → acquire() …. lock → release()

P1 y P2 hicieron cv → wait() y estan suspendidos, luego, un P3 realiza el cv → signal() y ambos se despiertan pero no en el orden FIFO, entonces, un proceso que debia adquirir el lock antes lo terminara haciendo despues ya que el otro proceso consiguio el acceso perdiendose de esta manera la sincronizacion.

(??? NI IDEA SI ESTA BIEN)

2.- Una pareja se va a divorciar. Para dividir sus propiedades, han acordado el siguiente algoritmo. Cada mañana, cada uno puede enviar una carta al abogado del otro solicitando un bien. Dado que se tarda un día en entregar las cartas, han acordado que si ambos descubren que han solicitado el mismo artículo el mismo día, al día siguiente enviarán una carta anulando la solicitud. Entre sus propiedades se encuentran su perro Woofer, la caseta del perro, su canario Tweeter y la jaula del Tweeter. Los animales aman sus casas, por lo que se ha acordado que cualquier división de propiedad que separe a un animal de su casa es inválida, requiriendo que toda la división comience desde cero. Ambos quieren desesperadamente a Woofer. Para que puedan irse de vacaciones (por separado), cada cónyuge ha programado una computadora personal para manejar la negociación. Cuando regresan de vacaciones, las computadoras aún están negociando. ¿Por qué? ¿Es posible un interbloqueo? ¿Es posible inanición? Explique su respuesta

Como ambos quieren a Woofer, seguramente cada computadora envia una carta a otra solicitando al perro, pero como ambos solicitan el mismo bien el mismo dia, al dia siguiente ambos envian una carta anulando la solicitud. Entonces, al tercer dia volveran ambas

computadoras a solicitar a Woofer y asi se repite el ciclo de manera que cuando regresan de las vacaciones, las computadoras siguen negociando. De esta manera se llega a una inanicion ya que todos los dias se reinicia la negociacion y ninguna camputadora puede terminar su ejecucion.
Deadlock no es posible.

Es la pregunta 4 de: https://www.cs.hmc.edu/~jsmallma/cs110/assignment6.html

1.- ¿Por qué la planificación de colas multinivel con retroalimentación varían el quantum de tiempo asociado con diferentes colas? ¿Cuál sería la desventaja de tener el mismo quantum de tiempo en todos los niveles?

El objetivo de tener distintas colas con diferente quantum asociado es que los procesos se separen dependiendo de su tiempo de uso del CPU, de esta manera los procesos que necesitan menor rafaga de CPU (por ej: los ligados a E/S) tienen prioridad para utilizar el tiempo del procesador y no necesitan esperar a aquellos que consumen mayor rafaga de tiempo. Ademas, aquellos procesos mas largos que son movidos a colas inferiores con mayor quantum, al tener un quantum mas largo provocaran menos cambios de contexto, obteniendose una mayor eficiencia.

(G – 216) (T – 161)

Si se tiene el mismo quantum de tiempo en todos los niveles, se pierde la razon de utilizar colas multinivel ya que si un proceso completa su quantum no habria justificacion para moverlo de cola debido a que todas tienen el mismo quanatum. Entonces la desventaja es que se pierde el beneficio de que aquellos procesos con poco tiempo de rafaga de CPU puedan acceder rapido al procesador y tambien se reduce la eficiencia.

2- ¿Qué es falso acerca del paso de mensajes? Justifique su respuesta

1. Implica una sección crítica en los datos del mensaje.
2. Una recepción bloqueante implica sincronización con el emisor.
3. Un envío bloqueante permite al emisor utilizar el buffer del mensaje cuando se desbloquea.
4. El paso de mensajes se implementa siempre con memoria compartida.

1. Falso, por ejemplo, si un proceso envia un entero a traves de un pipe a otro proceso, no hay ninguna seccion critica en los datos del mensaje ya que no hay ninguna variable.

2. Verdadero, porque solo puede continuar si el emisor envia un mensaje, es decir, lo espera y de esta manera se sincroniza.

3. ?? G -168 del pdf

4. Falso, el paso de mensajes no solamente se puede implementar con memoria compartida, tambien se puede hacer a traves de pipes o mediante cola de mensajes.

(NI IDEA SI ESTA BIEN)

2. Se tienen 3 procesos: P1, P2 y P3, con tiempos de ejecución: 85, 45 y 118 ms, respectivamente. Si actúa el planificador a largo plazo según el algoritmo SJF (Short Job First) se obtiene que: (Justifique su respuesta)

1. Los procesos se encuentran en la lista de preparados en el orden de llegada: P1, P2 y P3.
2. Los procesos se encuentran en la lista de preparados en el orden: P2, P1 y P3.
3. Los procesos se ejecutan en el orden de llegada: P2, P1 y P3.
4. Los procesos se ejecutan según la prioridad que posean los procesos.

El planificador de largo plazo es el encargada de seleccionar que procesos son admitidos en el sistema para ser procesados y puestos en la cola de listo, por lo tanto, si la politica de orden de selección es SJF, seleccionara al proceso mas corto de entre todos los procesos restantes (es decir, que aun no fueron seleccionados). Luego, se obtiene que los procesos llegan a la cola de listos en el orden P2, P1 y P3, de esta manera (1) es falso y (2) es verdadero. El (3) tambien es falso ya que el orden de ejecucion dependera de la politica de selección de procesos que tenga el planificador de corto plazo, por la misma razon, (4) es falso ya que no necesariamente la politica es ejecucion por prioridades.

(NI IDEA SI ESTA BIEN)

1. Si tiene un sistema operativo que está diseñado para una sola CPU y desea adaptarlo a un sistema de múltiples CPU, describa los tipos de cambios / adiciones al software o hardware que necesitaría realizar en relación con el primitivas de sincronización (bloqueos, etc...) y por qué. Liste tres.

????? https://www.scs.stanford.edu/10wi-cs140/exams/cs140.sum06.midterm.sol.pdf

2. ¿Cuándo entra un proceso en estado zombie? Justifique su respuesta

1. Cuando muere su padre y él no ha terminado todavía.
2. Cuando muere su padre sin haber hecho wait por él.
3. Cuando él muere y su padre no ha hecho wait por él.
4. Cuando él muere y su padre no ha terminado todavía.

(3) Un proceso entra en estado zombie cuando muere y su padre todavia no ha hecho wait por el ya que aunque el proceso hijo haya terminado, la entrada del mismo sigue en la tabla de procesos esperando para ser liberada por el padre y que este lea el estado de salida del hijo.

(G – 122)

1. Brinde un ejemplo para distinguir entre un proceso bloqueado y un proceso interbloqueado.

Proceso bloqueado: Un proceso P1 esta bloqueado cuando esta esperando obtener un recurso que esta bloqueado por otro proceso P2. Cuando P2 libera al recurso, P1 accede a dicho recurso y puede continuar su ejecucion.
Ejemplo: P1 necesita escribir un archivo que esta leyendo P2 y lo tiene en modo exclusivo, una vez que P2 termina, P1 accede al archivo.

Proceso intebloqueado: Un set de proceso esta bloqueado esperando obtener un recurso, el cual solo puede ser liberado por otro proceso del mismo set.
Ejemplo: Un proceso P1 tiene acceso a un recurso R1 y necesita acceder a un recurso R2, mientras que otro proceso P2 tiene acceso al recurso R2 y quiere obtener el recurso R1, entonces, ambos procesos estan interbloqueados ya que estan esperando que un recurso sea liberado por otro proceso del mismo set, se genera una espera circular.

2. ¿Cuál de estas transiciones de estados de un proceso jamás se produce en un sistema normal? Justifique su respuesta
    1. de *bloqueado* a *listo*.
    2. de *listo* a *bloqueado*.
    3. de *ejecutando* a *listo*.
    4. de *ejecutando* a *bloqueado*.

En un sistema normal un proceso jamas pasa de listo a bloqueado ya que si esta en la cola de listos quiere decir que es un nuevo proceso que llego recientemente y esta esperando para realizar su primer uso del procesador o es un proceso que estaba ejecutando y por alguna razon el planificador decidio seleccionar a otro proceso para que use el tiemo del CPU (por ej, se termino su quantum). En cambio, si esta bloqueado es porque esta esperando a que un evento externo suceda (por ej, una E/S), pero para que pase a ese estado tuvo que haber estado utilizando el procesador hasta llegar al punto en que no podia continuar debido al a necesidad de que ocurra dicho evento.
Entonces, la unica forma de que un proceso pase al estado bloqueado es que este utilizando el procesador (osea, su estado es ejecutando) y si un proceso esta en estado listo, este no esta utilizando el procesador.

1.- ¿Qué significa espera ocupada (busy waiting)? ¿Qué tipos de espera ocupada hay en un sistema operativo? ¿Se puede evitar por completo? Explique su respuesta.

Espera ocupada significa que un proceso esta esperando que suceda un evento mientras esta utilizando el procesador, es decir, esta esperando mantiendo al procesador ocupado. Mas especificamente, en la espera ocupada un proceso esa en un loop esperando que se cumpla una condicion (por ej: espera que una variable tenga el valor deseado), lo cual claramente genera un gasto de tiempo al CPU.

La espera ocupada se puede evitar a traves del bloqueo de los procesos, esto es, un proceso llama al syste call Sleep y queda bloqueado hasta que otro proceso lo despierte con el system call Wakeup.

(T – 124,128)
La pregunta es igual al ejercicio 3 de esta pagina de una parte del libro de soluciones de alguna edicion del libro de Silberschatz:
https://www.os-book.com/OSE2/practice-exer-dir/5-web.pdf

2.- La siguiente solución del problema Productor/Consumidor es correcta. Detalle su respuesta.

semaphore mutex = 0, empty = 0;

void Producer() {	void Consumer() {
down(mutex);	down(mutex);
produce();	down(empty);
up(empty);	consume();
up(mutex);	up(mutex);
}	}

La solucion es incorrecta, se debe agregar un semaforo que controle la cantidad de espacios llenos en el buffer.
El productor deberia hacer down(empty) antes de producir y luego de producir hacer up(lleno).
El consumidor deberia hacer down(lleno) antes de consumir y luego de consumir hacer up(empty).
La utilizacion del semaforo mutex es correcta ya que para acceder al buffer (consumir o producir) primero se debe adquirir el acceso exclusivo del mismo y liberarlo despues de realizar su tarea.
Respecto a las inicializaciones tambien son incorrectas, mutex debe arrancar en 1 ya que inicialmente nadie esta utilizando el buffer, empty debe arrancar en N siendo N el tamaño del buffer ya que arranca vacio y, por esa misma razon, el semaforo lleno debe arrancar en 0.

1.- Suponga que tiene un sistema en el que hay un ciclo en un grafo de espera del sistema. ¿Es esta una señal definitiva de un interbloqueo? Justifica tu respuesta.

No, un ciclo en una grafo de espera del sistema no implica que hay un interbloqueo, depende de la cantidad de instancias de cada recurso.
Si cada recurso tiene una sola instancia entonces si hay interbloqueo ya que cada proceso del set (de procesos interbloqueados) esta esperando obtener un recurso, el cual solo puede ser liberado por otro proceso del set
Si no ocurre que cada recurso tenga una sola instancia, entonces no se puede asegurar que hay interbloqueo ya que un proceso que no forma parte del ciclo podria liberar una instancia que es necesitada por un proceso del ciclo, de esta manera, el ciclo finalizaria.

(G – 324,325,326)

2.- Considere el siguiente código para testear locks y variables de condición. Comienza cuando el hilo "principal" llama a ThreadTest (). Realice una traza de ejecución de este programa hasta que imprima el mensaje "Pare aquí". Considere que la planificación es FCFS y las colas se administran utilizando FIFO.

```
Lock *l;
Condition *cv;

void A(int arg) {
  l->Acquire();
  cv->Signal();
  Thread->Yield();
  cv->Wait();
  l->Release();
}
void B(int arg) {
  l->Acquire();
  cv->Wait();
  Thread->Yield();
  cv->Signal();
  l->Release();
}

void ThreadTest() {
  Thread *t;
  l = new Lock("lock");
  cv = new Condition("cv");
  createThread("A");
  createThread("B");
  Thread->Yield();
  Thread->Yield();
  printf("Pare aquí\n");
}
```

Thread->yield obliga al hilo que realiza la llamada a renunciar al uso del procesador y a esperar en la cola de listos.

1. Escribir la secuencia de cambios de contexto que ocurrieron hasta este punto,
2. Enumerar las colas en las que se encuentran los hilos en este punto, y su orden relativo si hay más de un hilo en una cola
3. Indicar el estado de cada uno de los hilos.

1. Traza en la que se pueden ver los cambios de contexto

Instruccion	Hilo	
Thread *t	Principal	
l = new Lock()	Principal	
cv = new Condition()	Principal	
createThread(A)	Principal	
createThread(B)	Principal	
Thread → yield()	Principal	
Cambio de contexto		
l → acquire()	A	
cv → signal()	A	
Thread → yield()	A	
Cambio de contexto		
l → acquire()	B	→ no lo consigue al lock y se suspende
Cambio de contexto		
Thread → yield()	Principal	
Cambio de contexto		
cv → wait()	A	→ hace wait de la cv y se suspende
Cambio de contexto		
printf(Pare aquí)	Principal	

2.
El hilo A esta en la cola de espera de la cv.
El hilo B esta en la cola de espera del lock.

3.
Los hilos A y B estan en estado waiting.
El hilo principal esta en estado terminated?

(NI IDEA SI ESTA BIEN)

1.- Considere un proceso compuesto por dos hilos, *p* y *q*, que están compuestos por un conjunto de sentencias atómicas (A, B, C, D, y E). El proceso crea los dos hilos y comienza la ejecución concurrente. Muestre todos los posibles entrelazados de ejecución, considerando que los hilos contienen las siguientes sentencias.

```
void p()        void q()
{ A;            { D;
  B;              E;
  C;            }
}
```

???

==========

Parcial 2019

Ejercicio 1

1. Cuáles son los motivos principales para utilizar sistemas operativos multitarea.
2. Cuáles son las causas que provocan la ejecución de código del kernel del sistema operativo. Brinde ejemplo para cada una.

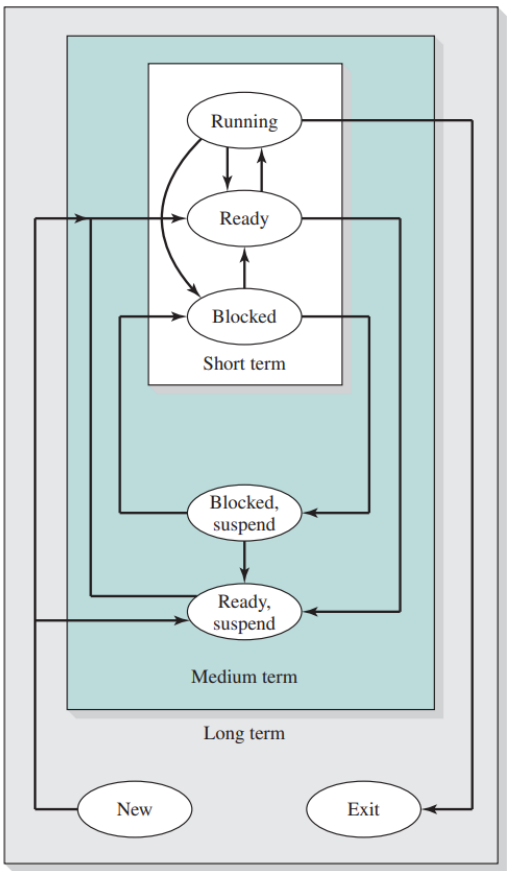
1. Los sistemas operativos multitarea permiten ejecutar varios programas en simultaneo generando una mayor eficiciencia a la hora de realizar tareas ya que no se tiene que realizar de a una a la vez, sino que se las realiza en conjunto. Como consecuencia, tambien permite que muchos usuarios utilicen el sistema a la vez ya que se reparte el tiempo del procesador entre las tareas de todos los usuarios.

2. Las causas que provocan la ejecucion ejecucion del codigo del kernel son:
- Interrupcion: ocurrio un evento externo al proceso que se esta ejecutando y debe ser atendido, por ejemplo, se completo una operación de E/S.
- Trap: ocurrio un error dentro del proceso que se esta ejecutando, por ejemplo, hubo un intento de acceso a un archivo no permitido.
- System call: un programa solicito algun servicio del kernel a traves de la interfaz provista (la llamada al sistema), por ejemplo, un proceso quiere crear un proceso hijo con fork().

(NI IDEA SI ESTA BIEN)

Ejercicio 2

1. Enumere y defina los tipos de planificadores.
2.- Realice el diagrama de estados de procesos y explique cada una de las transiciones y estados. De acuerdo al diagrama propuesto, explique cuándo se ejecutan los distintos tipos de planificadores.



Planificador de largo plazo: Determina que programas son admitidos en el sistema para ser procesados.
Planificador de medio plazo: Determina cuando un programa es llevado parcial o completamente a memoria principal para que este habilitada su ejecucion.
Planificador de corto plazo: Determina que proceso en estado listo sera el siguiente en ser ejecutado por el CPU.

(S – 429,430,454)
Imagen en: S – 428

Ejercicio 3 – Indique la respuesta correcta para cada uno de los incisos.

1. Un sistema operativo multiprogramado puede prescindir del despachador
   a) Si la política de planificación del procesador es apropiativa.
   b) En algunas (pero no en todas) las políticas de planificación apropiativas.
   c) Nunca en políticas de planificación no apropiativas.
   d) Ninguna es correcta

(d) Ninguna es correcta.
El despachador es el planificador de corto plazo, por lo tanto, un sistema operativo multiprogramado no puede prescindir del despachador ya que debe poder seleccionar que proceso sera el siguiente en acceder al procesador.

d) Ninguna es correcta

2. En un sistema operativo multiprogramado que se ejecuta en una arquitectura multiprocesador con una cola de procesos listos por cada procesador: Justifique en una oración.
   a) Las únicas políticas de planificación del procesador posibles son las de colas multinivel.
   b) Habrá tantas colas de procesos bloqueados como procesadores.
   c) Se equilibra la carga entre todos los procesadores.
   d) Ninguna es correcta

a) Falso, no necesariamente tiene que utilizarse colas multinivel.

b) Falso, las cantidad de colas de proceso bloqueados dependera de la cantidad de recursos a los que quieran acceder los procesos.

c) Falso, si bien el balanceo de carga es algo ideal, esto debe ser implementado y el enunciado no aclara nada al respecto.

d) Verdadero.

(NI IDEA SI ESTA BIEN)

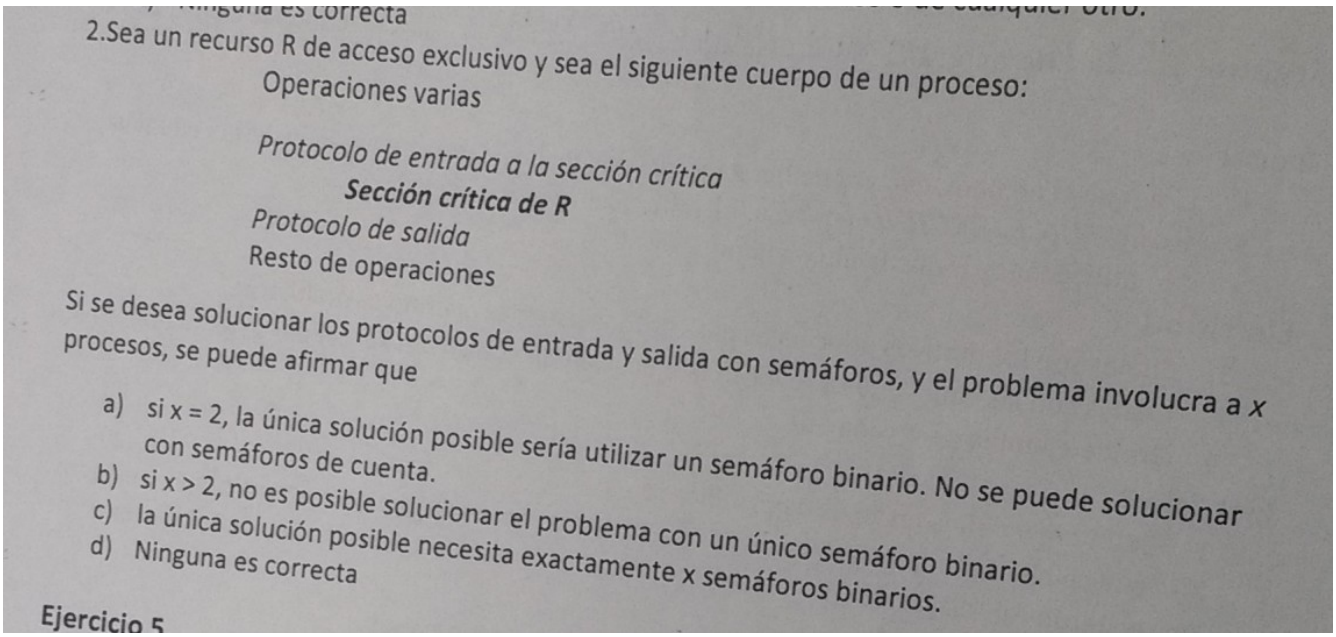
Ejercicio 4 - Indique la respuesta correcta para cada uno de los incisos y justifique su respuesta

1. Para que se cumpla la condición de progreso en la solución al problema de la exclusión mutua,
   a) un proceso no puede abandonar el sistema dejando bloqueado el acceso a la sección crítica
   b) un proceso en su sección crítica, no puede impedir que otro proceso ejecute el resto de sus operaciones que no son la sección crítica
   c) un proceso en la sección crítica de un recurso, debe impedir que cualquier otro proceso pueda ejecutar cualquier sección crítica, del mismo recurso o de cualquier otro.
   d) Ninguna es correcta

2. Sea un

?????
Diria que la (a) o la (d) pero ni idea

Definicion de progreso (G – 260)



Ninguna es correcta

2.Sea un recurso R de acceso exclusivo y sea el siguiente cuerpo de un proceso:

Operaciones varias

Protocolo de entrada a la sección crítica

Sección crítica de R

Protocolo de salida

Resto de operaciones

Si se desea solucionar los protocolos de entrada y salida con semáforos, y el problema involucra a *x* procesos, se puede afirmar que

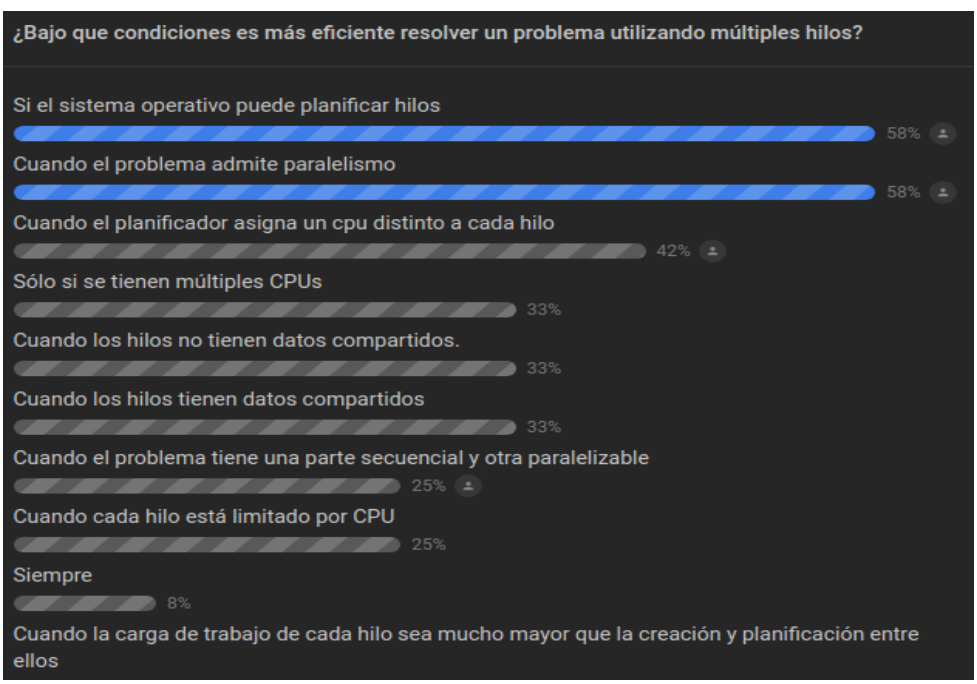
a) si x = 2, la única solución posible sería utilizar un semáforo binario. No se puede solucionar con semáforos de cuenta.
b) si x > 2, no es posible solucionar el problema con un único semáforo binario.
c) la única solución posible necesita exactamente x semáforos binarios.
d) Ninguna es correcta

Ejercicio 5

(d) Ninguna es correcta.
No importa la cantidad de procesos que esten involucrados se debe utilizar un semaforo binario ya que solo uno proceso puede entrar en la seccion critica a la vez, por lo tanto, el acceso solo puede estar habilitado (nadie esta usando la seccion critica) o deshabilado (alguien esta usando la seccion critica)

==========

Ejemplos de pregunta de parcial (preguntadas en alguna clase del 2021):



```
(Indicar opcion correcta) Un proceso interbloqueado:
a) Simpre esta en espera ocupada.
b) Nunca ejecuta.
c) Puede estar suspendido.
```

a) Falso, ya que el semaforo no necesariamente esa implementado con espera ocupada, puede que el proceso sea suspendido.
b) Falso, si el semaforo esta implementado con espera ocupada entonces si esta ejecutando mientras espera.
c) Verdadero, si PUEDE en el caso que el semaforo este implementado sin espera ocupada.

(respondido en la clase)

==========

Parcial random

1.- Enumere ventajas y desventajas de kernels monolítico, micro-kernel y modulados.

Un kernel monolitico no tiene estructura, ubica todas las funcionalidades del kernel en un solo archivo binaria estatico qu se ejecuta en un solo espacio de memoria. El kernel provee el sistema de archivos, la planificacion del del procesador, el manejo de memoria y otras funcionalidades a traves de system calls, es decir, hay muchas funcionalidades dentro de un solo espacio de memoria.
Como ventaja, tiene buena performance y la comunicación dentro del kernel es rapida, ya que se encuentra todo dentro de el. Sin embargo, tiene como desventaja que es dificil de implementar y extender.

Un microkernel estructura al sistema de manera que todo componente no escencial sea removido del kernel y se lo implmenete como un programa de usuario. Su principal funcion es proveer comunicación entre el programa y otros servicios que tambien estan ejecutandose en el espacio de usuario.
Como ventaja, extender el kernel es sencillo a comparacion del monotilico, ademas, provee mayor seguridad y fiabilidad ya que la mayoria de los serivicios corren como procesos usuarios en vez de kernel, si un servicio falla, el resto queda sin alterar.
Como desventaja, tiene una mala performance ya que cada vez que dos servicios a nivel usuario se quieren comunicar, los mensajes se deben copiar entre los servicios, que estan en espacios de memoria separados. Ademas, el SO puede tener que cambiar de un proceso al otro para intercambiar los mensajes. Todo esto genera peor performance.

Un kernel modular tiene un conjunto de componentes centrales y puede agregar diferentes servicios a traves de modulos. Todos los modulos pueden comunicarse entre si.
Como ventaja es parecido al microkernel ya que el modulo primario solo tiene funciones centrales, pero es mas eficiente ya que los modulos no necesitan pasaje de mensajes para

comunicarse. A su vez, es mas facil de implementar y extender que un kernel monolitico ya que solo tiene las funcionalidades necesarias en el modulo central.

(G – 82 hasta 86)

2.- Describa dos operaciones del Sistema Operativo para proteger los recursos.

???
Una operación del sistema operativo para proteger los recursos es el cambio de modo entre usuario y kernel. En el modo usuario, las posibilidades de accion estan restringidas, por ejemplo, acceso a ciertas instrucciones para el manejo de la memoria, acceso a tablas importantes como el PCB, acceso a los registros, etc.

Y la segunda operación no se.

(NI IDEA SI ESTA BIEN)

1.- Describa cómo está compuesto el PCB.

Cuando un proceso esta en el sistema, necesita guardarse cierta informacion del mismo para que el sistema operativo pueda manejarlo, por ello, cada proceso tiene su PCB, este esta compuesto por los siguientes elementos del proceso: el pid, el estado, el program counter, punteros de memoria, datos de contexto (que estan en los registros del procesador mientras se ejecuta), informacion del estado de E/S e informacion adicional (por ejemplo: tiempo de uso del CPU, limites de tiempo, etc).

2.- Dado el siguiente segmento de código:

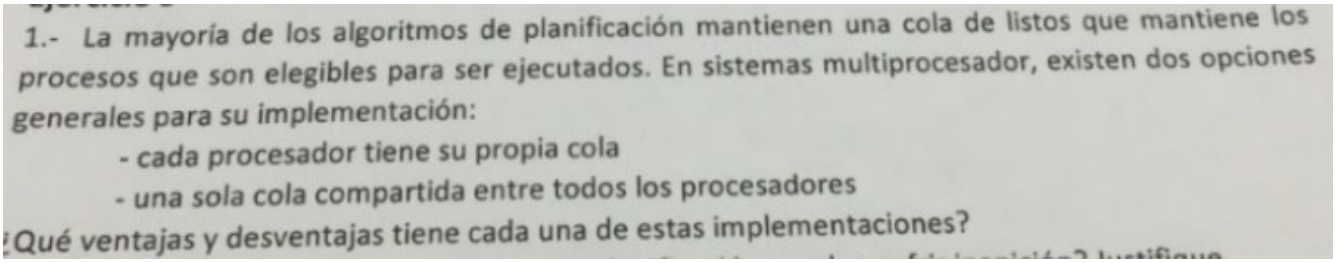
```
pid_t pid;

pid = fork();
if (pid == 0) {
   fork();
   createThread(....);
}
fork();
```

a) ¿Cuántos procesos son creados? Explique cómo llega a la solución.
b) ¿Cuántos hilos son creados? Explique cómo llega a la solución.

a) Se crean 5 procesos, primero el padre (proceso 1), luego, en el fork() anterior al if se crea al hijo (proceso 2), este hijo entra al if y en el fork del if se crea al hijo del hijo (proceso 3). Finalmente, luego del bloque del if, tanto el padre como el hijo originales realizan un fork() creando cada uno otro proceso (proceso 4 y proceso 5).

b) Se crean 2 hilos, el proceso 2 (de la explicacion del inciso a) realiza el fork() creando al proceso 3, por lo tanto, ambos continuaran su ejecucion en la sentencia siguiente la cual es la de creacion de hilo y cada uno creara un hilo.

(NI IDEA SI ESTAN BIEN)


1.- La mayoría de los algoritmos de planificación mantienen una cola de listos que mantiene los procesos que son elegibles para ser ejecutados. En sistemas multiprocesador, existen dos opciones generales para su implementación:

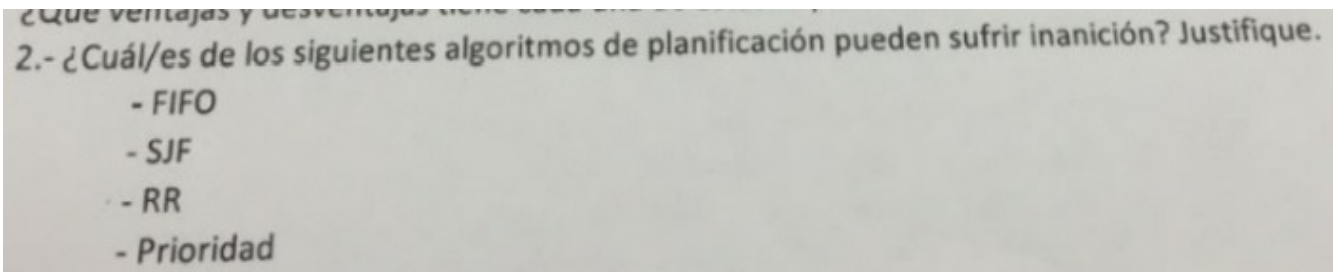
- cada procesador tiene su propia cola
- una sola cola compartida entre todos los procesadores

¿Qué ventajas y desventajas tiene cada una de estas implementaciones?

Si hay una sola cola compartida, la desventaja es que podria suceder una condicion de carrera, por lo tanto se deberia acceder a un mutex para poder acceder a la cola de forma exclusiva y evitar la condicion de carrera. Esto genera una menor performance ya que involucra que mientras un procesador esta obteniendo su proximo proceso, el resto deba esperar generando un cuello de botella.
Como ventaja, no surgen problemas respecto al balance de la carga entre los distintos procesadores ya que al tener todos una misma cola, en cuanto algun procesador este libre, otro proceso accedera al mismo.

Si cada procesador tiene su propia cola, como ventaja se eliminan las infeciencias del acceso exclusivo a una sola cola ya que cada procesador tiene su propia cola privada.
Como desventaja, puede suceder que mientras algun procesador esta muy sobrecargado, otro este inactivo, por lo tanto, se debe implementar alguna forma de balanceo de carga.
Ademas, otra situacion que ocurre es la afinidad del procesador que tienen cada proceso, ya que al cambiarlo se debe vaciar su memoria cache del procesador que dejo y repoblarla con nueva informacion, esto conlleva una desmejora en la performance.

(G – 220 hasta 225)


2.- ¿Cuál/es de los siguientes algoritmos de planificación pueden sufrir inanición? Justifique.

- FIFO
- SJF
- RR
- Prioridad

El algoritmo de planificacion por prioridades puede sufrir inanicion ya que si constantemente llegan procesos de alta prioridad, estos ganaran el acceso al procesador mientras que aquellos de menor prioridad deberan estar continuamente esperando para poder utilizarlo.
Tambien puede pasar con SJF ya que si constantemente llegan procesos con menor tiempo de proxima rafaga, los que tengan mayor rafaga deberan esperan continuamente.