**1)Responda verdadero o falso y justifique**

1. Para que se necesite la comunicación entre procesos es necesario un sistemas de multiprocesamiento ya que se requiere al menos dos o más procesos ejecutando concurrentemente.

Falso. El multiprocesamiento es el uso de dos o más procesadores para la ejecución de múltiples procesos. El multiprocesamiento puede dar como resultado el paralelismo: la ejecución AL MISMO TIEMPO de dos o más procesos. Pero la concurrencia puede existir (y existe) en sistemas monoprocesador con multiprogramación. Si nuestro sistema monoprocesador implementa algoritmos de planificación de corto plazo no apropiativos, entonces nuestros procesos podrán ser ejecutados de forma concurrente, es decir, su ejecución se superpondrá en el tiempo con la de otros procesos.

La comunicación se necesita siempre que haya concurrencia entre procesos y mientras que estos procesos interactúen entre sí. Puedo tener, dentro de un mismo sistema, dos procesos concurrentes que establezcan una comunicación con motivos de sincronización o de paso de mensajes.

1. Uno de los mayores beneficios de utilizar varios hilos en lugar de varios procesos para un determinado software, es que comparativamente los threads producen menos overhead ya que no producen context-switch a diferencia de los procesos que sí lo requieren.

Verdadero en caso de que los threads sean ULT (user level threads). Al ser ULT, el sistema operativo no tiene idea de la existencia de los mismos, y es el proceso el que realiza el switching entre threads. Este switch es de menor impacto que un context switch realizado por el S.O, ya que el proceso solo tiene que asegurarse de guardar el TCB (thread control block), que es una estructura de datos más chica que el PCB y por ende más rápida de guardar. Además, no es necesario un cambio de modo (pasar de modo usuario a modo kernel para realizar el cambio de contexto).  
  
Falso en caso de que sean KLT (kernel level threads). Se produciría mucho overhead ya que en este caso el S.O sí sabe de la existencia de los hilos y los trata a cada uno como si fueran un proceso.

1. En un determinado momento, en un sistema con monoprocesamiento la cola de listos se encuentra vacía y el S.O. ejecutando un rutina de atención a una interrupción. Podemos afirmar que en ese preciso instante el procesador se encuentra ocioso ya que no hay procesos usuarios a ejecutar.

Falso. Que sea monoprocesamiento no quiere decir que no se aplique multiprogramación, ni que el algoritmo de planificación a corto plazo sea apropiativo. Si el sistema no admite multiprogramación, y si el algoritmo a corto plazo es apropiativo, entonces el procesador estaría ocioso. De otra forma, el proceso nulo se pondría en ejecución para evitar el ocio del procesador.

1. Hace pocos días en un determinado datacenter se generan backups y se actualiza la B.D. en el horario de las 10:00am. A partir de ese momento, los usuarios comenzaron a quejarse por una caída en la performance del sistema durante su horario laboral de 9:00am a 18:00. Un especialista afirma que a pesar de la poca capacidad de procesamiento que se posee, el rendimiento podría mejorar ampliamente cambiando las prioridades de los procesos que se encuentran en la cola de listos.   
   Si compartís la afirmación, explique cómo realizaría determinada modificación. En caso contrario, indique en cuál de los 4 planificadores vistos realizaría la modificación y por qué.

Le daría más prioridad a los procesos de usuarios. El problema sería caer en la inanición del proceso de backup. A las 10AM, habrá una cantidad interesante de procesos de los empleados. El sistema le daría más prioridad a los mismos, mejorando claramente la respuesta que ellos reciben. El problema es que el proceso de backup quedaría relegado y probablemente su progreso sea muy lento.  
  
Un round robin con prioridades serviría. O si no, implementar una planificación con múltiples colas, en base al historial de ejecución de los procesos, que priorice procesos cortos y demás.

1. Debido a la baja probabilidad de ocurrencia, algunos S.O. como Linux y Windows utilizan el sencillo algoritmo del avestruz para reponerse de los Deadlocks.  
     
   Falso. El algoritmo del avestruz es no hacer nada para prevenir o recuperarse de un deadlock. Windows no implementa esta solución, ya que, si un proceso empieza a tardar en responder, el mismo S.O. le indica lo mismo al usuario, ofreciéndole terminar al proceso como una solución. Por lo que Windows lleva cierto reconocimiento del estado de los procesos y de su situación actual. Detecta el deadlock.
2. Los programas objetos tiene resueltas las referencias a las bibliotecas (library) del S.O. en donde se compiló.

Falso. Es el trabajo del link-editor el hecho de enlazar las bibliotecas necesarias que necesita el programa. El programa objeto tiene la información de los recursos que necesita, incluidas las bibliotecas, pero no las tiene enlazadas.

2)Reponder

● Explicar brevemente los métodos de multiprocesamiento simétrico y asimétrico y la diferencia entre los estados: Bloqueados, Bloqueados/Suspendidos y Suspendidos/Listos

<https://pc-solucion.es/2018/04/18/diferencias-entre-multiprocesamiento-simetrico-y-asimetrico/>

**Estados**

Bloqueado: el proceso se encuentra bloqueado (es decir, sin ejecutarse), a la espera de un evento, por ejemplo, que un dispositivo de E/S termine de realizar la acción pedida.

Bloqueado/Suspendido: el proceso se encuentra bloqueado a la espera de un evento, pero debido a varios factores (el evento que espera es algo lento que tarda mucho, el proceso es de baja prioridad, se necesita espacio en memoria central para proceso de más prioridad o menor tiempo de ejecución), el proceso es movido desde la memoria central al almacenamiento secundario. A esto se lo llama swapping.

Suspendido/Listo: el evento que el proceso estaba esperando sucedió y ya no tiene que esperarlo. Su ejecución puede continuar. Pero todavía sigue en el almacenamiento secundario, a la espera de ser swappeado a memoria central. Se implementa una cola de suspendidos/listos ya que el swapping es algo costoso, y el overhead que causaría pasarlo directamente a memoria central una vez que sucede el evento, sería mucho.