Respuestas Orientadoras e incompletas (en algunos casos) a las preguntas de revisión de las clases 4,5,6,7. Favor de tomarlas como orientadoras.

Clase 4 – Revisión – Unidad II – Procesos (hasta modelo de 5 estados)

1. Enumere 4 oraciones que definan qué es un proceso.

Programa en ejecución.
Instancia de un programa ejecutado en un computador.
"Espíritu animado" de un programa.
Entidad asignada a una CPU y ejecutada por ésta.
Unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual y un conjunto de recursos del sistema asociados.

2. ¿Qué es la traza de un proceso?

Esta asociado con el comportamiento del proceso, es la secuencia de instrucciones ejecutada por el proceso.

3. ¿Por qué razón el modelo de dos estados no es apropiado para describir lo que realmente ocurre con los procesos?

Porque este modelo asume que todos los procesos en estado "not running" pueden pasar a estado a "running" una vez seleccionados por el dispatcher y en la realidad esto no es así, puede haber procesos que queden en estado "not running" porque estan esperando la finalización de una operación de entrada/salida.

4. ¿A qué nos referimos con “process spawning”?

"Process spawning" es cuando un proceso crea otro proceso, a través de la operación fork(). Esto también se denomina creación de procesos "pesados" en contraposición a la creación de procesos "livianos" (threads/hilos).

5. ¿A qué nos referimos con “swapping”?

Se refiere a mover todo o parte de un proceso que se encuentra actualmente en memoria principal hacia memoria secundaria. El area de swap es un area de intercambio entre la memoria principal y la secundaria.

6. ¿Qué diferencia hay entre un proceso en estado “Blocked” y otro en estado “Blocked/Suspend”?

El lugar de residencia. Ambos procesos estan bloqueados, solo cambia su ubicación, el proceso "blocked" se encuentra ocupando memoria principal, mientras que el proceso "blocked/suspend" se encuentra en memoria secundaria. Este tipo de estados se aplica en SO que utilizan memoria virtual, permitiendo realizar swapping entre los distintos procesos a ejecutar.

7. ¿ A qué nos referimos con “time overrun”?

Para muchos eventos el SO establece un tiempo límite, cuando se excede el tiempo límite de espera de un evento que se supone debiera haber ocurrido y éste no ocurre, se termina el proceso debido a un "time overrun".

8. ¿En dónde se encuentra un proceso suspendido?

En memoria secundaria.

9. Describa los 5 estados posibles de un proceso.

Running: proceso en ejecución.
Ready: listo para ejecutar cuando le den la oportunidad.
Blocked/Waiting: proceso que no puede ejecutarse hasta que algún evento ocurra, por ejemplo, la finzalización de una operación de entrada/salida.
New: proceso recien creado pero aún no incluído dentro del pool de procesos ejecutables.
Exit: un proceso que ha sido sacado (por el SO) del pool de procesos ejecutables

10. ¿Por qué razones puede suspenderse un proceso?

-Para permitir realizar swapping del proceso y liberar memoria principal.
-Debido a que el SO ha detectado algún interbloqueo con otros procesos.
-Por decisión del usuario, para hacer debug del proceso o porque el mismo esta haciendo uso de algún recurso.
-Por temporización del proceso, ejemplo un proceso hace un simple chequeo cada 30 segundos y luego queda en estado suspendido, así sucesivamente hasta que le toca nuevamente hacer el chequeo.
-En una situación de spawning el proceso padre puede decidir suspenderse a la espera de la finalización de la ejecución del proceso hijo (coordinación, ejemplo un shell o intérprete de comandos) o también puede decidir suspender la ejecución del proceso hijo para inspeccionarlo o modificarlo.

Clase 5 – Revisión – Unidad II – Procesos (Estructuras para gestionar procesos – modos de ejecución – modos de ejecución del SO – creación de procesos Unix/Linux)

11. ¿Por qué es necesario para el SO mantener estructuras de control de los procesos?

Para que el SO pueda seguir la traza de los procesos, para poder administrarlos debe mantener el registro de estado de cada proceso y su informacion de contexto, puesto que la ejecución de los procesos es intercalada/interrumpida con el process switcher y los otros procesos en ejecucion. La cpu "se presta" entre los distintos procesos, incluyendo los procesos del propio SO.

12. ¿Qué categorías de información hay en un bloque de control de proceso?

Identificación de Proceso, Info. de estado cpu, Info. Control de Proceso.

13. ¿Qué es la imagen de un proceso?

Fig.3.16 pag.142. Imagen=bloque de control de proceso (PCB)+pila o stack usuario+espacio de direcciones de usuario+pila núcleo+espacio compartido de direcciones

14. ¿Por qué razón se realiza un cambio de proceso (process switching)?

Pag.138. se realizar porque ocurre una interrupción (causa externa al proc. en ejecucion), ocurre un trap (causa interna al proc. en ejecución), se realiza una llamada al SO (system call).

15. ¿Por qué razón se realiza un cambio de modo?

Porque ha ocurrido una interrupción, se cambia de modo usuario a kernel (para ejecutar instrucciones privilegiadas) o viceversa (para ejecutar instruccion no privilegidas, en modo usuario), se debe guardar el contexto de ejecucion del proceso.

16. ¿Qué diferencia hay entre “cambio de proceso” y “cambio de modo”?

No son conceptos equivalentes, un cambio de modo afecta al tipo de instrucciones que se pueden ejecutar, pero esto no implica un cambio de proceso. Se puede hacer un cambio de modo dentro del mismo programa, esto esta en relación con los modos de ejecucion que tiene un SO Fig.3.15(b).

17. ¿Qué diferencia hay entre “trap” e interrupción?

Son conceptos similares, pero no no idénticos, un trap es una forma de interrupción pero que se origina dentro del propio proceso interrumpido, asociado a una condicion de error o excepción. Mientras que una interrupción es algo externo al proceso en ejecución, por ejemplo, puede ser que se haya agotado la rodaja de tiempo asignada para la ejecucion del mismo (time slice) entonces se produce una interrupción de reloj (clock).

18. En un entorno Unix, Describa: proceso swaper, proceso 0, proceso 1 y proceso init.

Pag.144. Proceso swapper es sinónimo de proceso 0, es mas bien una estructura de datos que se carga en memoria cuando el sistema arranca. Luego este proceso ejecuta al proceso 1 o proceso init y éste es el padre de todos los procesos.

19. En un entorno Unix, ¿Qué es un proceso zombie?

Es un proceso que ya no esta en ejecución, ya sea por una terminación normal o anormal o porque

fue abortado; pero aún queda su información registrada en el sistema para que pueda consultarla el padre. Obsérvese el código del shell, cuando el proceso padre hace un fork() (crea un hijo), y luego hace una llamada a la función waitpid() (si el hijo termina antes de la llamada a waitpid() por parte del padre, el hijo pasa a estado zombie) y luego el padre necesita que no "desaparezca su rastro" para recuperar el status de finalización del proceso hijo. Una vez que el padre ha recuperado la información de finalización del proceso hijo a través de waitpid(), el hijo deja de estar en estado zombie y pasa al estado terminado [Tanenbaum, Sistemas Operativos Modernos, 3ra. ed., Pag. 744].

20. En un entorno Unix, ¿Para qué sirve la llamada al sistema (system call) fork()? ¿Cómo funciona?

Pag.148. fork() sirve para crear un proceso, el SO realiza las siguientes funciones:
1. Solicita una nueva entrada en tabla de procesos para el nuevo proceso 2. Asigna ID proceso 3. Copia imagen del proceso padre con excepción de las regiones de memoria compartidas 4. Incrementa en uno el contador de ficheros en posesión del padre porque ahora el hijo también puede acceder a los ficheros 5. El nuevo proceso se pone en estado listo para ejecutar (ready to run) 6. Devuelve el id de proceso hijo al padre, devuelve 0 al proceso hijo.

Luego de esto el SO tiene 3 opciones: 1. continuar con la ejecución de padre, volviendo al modo usuario, luego de la instrucción fork() 2. transferir el control al proceso hijo, continuar después de la instrucción fork() 3. transferir el control a otro proceso, dejando a padre e hijo en estado “ready to run”.

Clase 6 – Revisión – Unidad II – Procesos (Hilos – Relación Hilos vs Procesos)

1. ¿Qué diferencia hay entre “monohilo” y “multihilo”?

Fig.4.1 (izquierda) Monohilo = 1 proceso = 1 hilo, por ejemplo MS-DOS; mientras que multihilo implica que el SO da soporte a n hilos en ejecución en un mismo proceso, por ejemplo: Windows, Solaris, Fig.4.1 (derecha).

2. Defina qué es un hilo o thread.

Un hilo es una unidad de ejecución que se activa, tiene su propio pc y consume cpu, llamado también thread o proceso ligero/liviano. Mientras que un proceso o tarea es aquel quien tiene la propiedad o tutela de los recursos requeridos para la ejecución de dicho proceso o tarea, la cual puede estar formada por un único hilo o múltiples hilos de ejecución.

3. ¿Por qué un cambio de contexto entre procesos es más “pesado o lento” que un cambio de contexto entre hilos?

Porque para cambiar entre procesos se requiere la intervención del SO para que éste guarde el contexto de ejecución de proceso actual, lo descargue, recupere el contexto de ejecución del nuevo proceso a ejecutar y lo ponga en ejecución. Mientras que un hilo o thread comparte gran parte del contexto del proceso que lo contiene, por lo tanto se puede cambiar de un hilo a otro dentro del mismo proceso sin necesidad de intervención del SO (tampoco requiere un cambio de modo) y sólo salvando y restaurando el contexto propio del thread, véase Fig. 4.2. Los hilos comparten el bloque de control de proceso, solo hay que salvar/restaurar el bloque de control de hilo.

4. Volviendo al concepto de “process spawning” (clase inicial de procesos) ¿Qué sería más eficiente? ¿Llamar a la función fork() o crear un hilo para la atención de cada cliente?

Es más eficiente crear un hilo de atención para cada cliente, porque es más rápido crear un hilo que hacer un fork().

5. ¿Qué sucede con los hilos del proceso X cuando el SO suspende al proceso X?

En un ambiente ULT (User Level Thread), el SO no hace gestión a nivel de hilos (es más, no sabe de su existencia), toda la planificación es a nivel de proceso, por lo tanto, cuando se suspende un proceso, todos los hilos de ese proceso se “suspenden”, no podrían estar en ejecución mientras el proceso que los contiene queda suspendido; el entrecomillado se debe a que esto no implica que los

propios hilos se reconozcan en estado de suspendido, obsérvese la Fig. 4.7 un proceso puede quedar bloquedado y sin embargo, sus hilos continúan en estado ready y running. En un ambiente KLT el SO realiza una gestión a nivel hilo, no obstante la pregunta no se ajusta demasiado a esta idea puesto que aquí son hilos en ejecución dentro del kernel y no dentro de un proceso de usuario; algo similar ocurre en un ambiente ULT/KLT si el hilo ULT está asociado a un KLT el hilo ULT es como que hereda la gestión del hilo KLT, pero el proceso de usuario que lo contiene bien podría quedar suspendido, en tal caso, la bibliografía no aclara que sucede con dicho hilo, pero seguramente no podrá estar en ejecución más allá del status propio del hilo (similar a lo comentado en el ambiente ULT). La pregunta es más aplicable a un ambiente ULT que a los restantes.

6. ¿En un ambiente ULT (user level thread), quién está a cargo de la planificación de los hilos?

El SO no está a cargo de su planificación, ni conoce de su existencia. La planificación está a cargo del programador, quien por lo general, utilizará una librería o biblioteca de software para gestionar los hilos dentro del programa, un ejemplo de ello es la biblioteca pthread (posix threads).

7. ¿En un ambiente KLT (kernel level thread), quién está a cargo de la planificación de los hilos?

El SO.

8. Existen 4 relaciones posibles entre hilos y procesos (1:1, M:1, 1:M, M:N) ¿Cuál de todas ellas está relacionada con los SO Distribuidos?

1:M en donde un hilo puede migrar de un entorno de proceso a otro, permitiendo que los hilos puedan moverse entre distintos sistemas, ejemplo: SO distribuidos como RA (clouds), Esmerald.

9. ¿Por qué se dice que ULT (user level thread) no saca provecho de la multiprogramación?

Porque muchas llamadas al sistema (system calls) son bloqueantes, cuando un hilo hace una llamada bloqueante, todo el proceso queda bloqueado, porque la gestión en un ambiente ULT es a nivel de proceso, no de hilo, en un momento dado sólo puede haber un único hilo en ejecución (si la gestión es a nivel de proceso, el SO no puede asignar el mismo proceso a más de una CPU al mismo tiempo); en este caso, ULT no está sacando provecho de la multiprogramación.

10. ¿Cuáles son las soluciones posibles al problema planteado en la pregunta anterior? ¿Qué entiende por “jacketing”?

Se plantea la posibilidad de escribir un proceso por cada thread, pero ello no es solución, puesto que perdemos la principal ventaja de los threads. La otra posibilidad es utilizar una técnica llamada “jacketing” la cual hace un envoltorio (wrap) de una llamada bloqueante para “transformarla” en una llamada no bloqueante: por ejemplo, en el caso de una operación de I/O el hilo verifica primero si el dispositivo está ocupado (esto es una llamada a nivel de la aplicación, no del sistema), si el dispositivo está ocupado, seguramente la llamada de I/O al SO provocaría el bloqueo del proceso, entonces no realiza la llamada al SO sino que suspende (a nivel de la aplicación del usuario) el thread y pasa el control a otro thread para tener la oportunidad de preguntar más tarde por el estado del dispositivo (si le da que está desocupado, ahí sí hace la llamada al SO, lo cual no implicaría un bloqueo muy largo).

Clase 7 – Revisión – Unidad II – Procesos (SMP – Micronúcleo)

1. Según la taxonomía de Flynn ¿Cuáles son los 4 tipos de sistemas que existen?

SISD monoprocesador; mientras que los sistemas que soportan procesamiento en paralelo serían: MISD, SIMD, MIMD. De éstos, el que más no interesa es el MIMD, puesto que de allí derivan los clusters y los SMP.

2. ¿Un SMP es un MIMD con memoria compartida (fuertemente acoplada) en el cual el núcleo se ejecuta en CPU determinado?

No. Un SMP es un MIMD con memoria compartida en el cual el núcleo se ejecuta en cualquier CPU.

3. ¿En qué se diferencia un SMP de un Cluster?

En cuanto a la memoria. Un SMP tiene una memoria fuertemente acoplada, es compartida por n CPU's que se encuentran sobre el mismo bus, en la misma motherboard, en un mismo computador Fig. 4.9, Pag.175. Mientras que un Cluster tiene una memoria débilmente acoplada o distribuida, puesto que se trata de n computadores distintos c-u con una o más CPU's y con sus propias memorias.

4. ¿Por lo general, el problema de la coherencia de cachés debe ser resuelto por el SO?

No, es resuelto por hardware.

5. ¿En un SMP si falla una CPU entonces falla todo el sistema?

No, porque el kernel puede ejecutarse en cualquier cpu, el sistema debería detectar la falla y reajustarse para ejecutarse en una cpu menos, debe ser tolerante a este tipo de fallos.

6. ¿Por qué se dice que un núcleo o kernel SMP debe ser "reentrante"?

Porque implica que un mismo código pueda estar ejecutándose al mismo tiempo en más de una cpu con un conjunto de datos distintos; debe ser un código lo suficientemente depurado y probado para evitar interbloqueo u operaciones inválidas.

7. ¿Qué tipo de algoritmos se debe utilizar en cuanto a sincronización en un SMP?

El problema es que puede haber n CPU's accediendo al mismo tiempo a un único conjunto de recursos, entre ellos la memoria principal, por ejemplo. Esto implica que se deben utilizar algoritmos tales como exclusión mutua y cerrojos o candados.

8. ¿Micronúcleo implica SO pequeño?

NO. Es sólo una cuestión de diseño, se pueden hacer SO pequeños y grandes (en cuanto a la funcionalidad que éstos ofrezcan) utilizando este tipo de diseños.

9. ¿Monolítico implica SO grande?

NO. Idem anterior, por lo general, los SO monolíticos son los SO más primitivos (por ende, los más pequeños), pero ello no implica (aunque no parece una buena idea) que no pueda desarrollarse un SO relativamente grande bajo este diseño; seguramente tendría serios problemas de mantenimiento y para asegurar su fiabilidad.

10. ¿SO por capas es igual que monolítico?

NO. La principal diferencia entre uno y otro es su organización interna. Un SO por capas es un SO estructurado, es un diseño más maduro. No obstante, un SO por capas puede compilarse en un único bloque de código que corra todo en un mismo espacio de direcciones al estilo de un SO monolítico, pero no obstante ello, su estructura interna permitiría un mejor mantenimiento. También podría implementarse esta idea de SO por capas en una arquitectura no monolítica, usando un micronúcleo en donde existan distintas capas de servicios c-u con su objetivo, su responsabilidad y su interfase bien definidas. O también podrían ser una serie de módulos a ser cargados en tiempo de ejecución y cada uno de ellos con su espacio de direcciones, comunicados entre sí a través de mensajes como es el caso de Linux.

11. ¿El Micronúcleo corre en modo kernel o núcleo, al igual que el resto del SO?

No. El micronúcleo corre en modo kernel, pero el resto del SO corre en modo usuario.

12. ¿La implementación de IPC (inter process communication) está bajo la responsabilidad del micronúcleo o es algo externo a él?

IPC esta bajo la responsabilidad del micronúcleo, pero cabe aclarar que sólo en cuanto a su forma más elemental de comunicación entre procesos, como es el caso del pasaje de mensajes (con cabecera: remitente, receptor; cuerpo: datos, punteros,etc.; puerto: cola de mensajes por proceso).

13. ¿Qué relación hay entre la arquitectura client-server y el micronúcleo?

Se puede pensar a un micronúcleo como un conjunto de procesos servidores que dan servicio a un conjunto de procesos clientes; en este sentido, hay una relación directa entre ambos, sería el caso de una arquitectura client-server dentro de un mismo servidor. Los procesos servidores validan los mensajes recibidos de los clientes y los direccionan internamente hacia/desde el hardware y los

distintos componentes del SO.

14. ¿Por qué razón se dice que hay un potencial problema de rendimiento en la implementación de un SO con micronúcleo? ¿Cuál sería una posible solución?

Porque el envío, recepción y decodificación de mensajes entre procesos es más lento que hacer una llamada al SO (system call) como podría hacerse en un SO bajo una arquitectura no micronúcleo. Una posible solución está en la reducción del tamaño del micronúcleo, cuanto más pequeño, menos tardarán el traslado y decodificación de mensajes, logrando un rendimiento similar y hasta incluso mejor que en un SO por capas tradicional.

15. Explique por qué un SO con micronúcleo facilita la implementación de Sistemas Operativos Distribuidos

Porque si cada mensaje que se envía dentro del SO incluye un ID de servicio solicitado y este ID de servicio es único dentro del sistema (para todas las cpu's existentes en todas las máquinas que posea el SO), se puede obtener una imagen única del servicio a nivel de micronúcleo, facilitando la implementación distribuida del SO.

16. Acorde con la implementación de hilos en las nuevas versiones de Linux, ¿Qué ventajas obtengo cuando los hilos de usuarios están asociados con hilos del núcleo y todos comparten el mismo id de grupo de proceso?

La asignación de recursos se hace por grupo de procesos por lo tanto, si un hilo a nivel de usuario se asocia con un hilo a nivel de kernel y éste tiene asociado n recursos, ambos tendrían acceso a los mismos recursos, podrían compartirlos, evitando un cambio de contexto cuando el planificador cambia entre procesos del mismo grupo.

17. En Unix/Linux, ¿Qué es un proceso zombie?

Idem 5.19.