Puntuación: para cada pregunta • *100% si está bien contestada* • *-50% si no está bien contestada* • *0 si se deja en blanco*	*UO:* *Nombre:*	*Modelo 0*

Cuestiones Verdadero / Falso

IMPORTANTE: Este examen consta de preguntas verdadero/falso agrupadas en grupos de cuatro. Sin embargo **el valor de cada pregunta es independiente de las otras tres**. Debes contestar a cada pregunta independientemente, con VERDADERO o FALSO.

[Arranque del Sistema]

1. El iniciador ROM es el encargado de crear las estructuras de datos del sistema Operativo.
2. El boot localizado en el sector de arranque es proporcionado por el fabricante del hardware.
3. *El iniciador ROM carga el sector de arranque del disco.
4. El iniciador ROM comprueba el sistema de ficheros.

[Activación del Sistema Operativo]

5. El SO se activa siempre que un proceso intenta acceder a una dirección de memoria principal.
6. *El SO se activa siempre que un proceso realiza una llamada al sistema para acceder a un fichero.
7. *El SO se activa siempre que se produce un error de ejecución como “null pointer Exception”.
8. *El SO se activa siempre que el procesador recibe una interrupción externa.

[Activación del Sistema Operativo]

9. * Las llamadas al sistema son funciones de la API contienen una instrucción tipo TRAP.
10. *Cuando se activa el Sistema Operativo el procesador pasa a ejecución en modo núcleo.
11. *Cuando se ejecuta una instrucción TRAP, se salta al sistema operativo, por lo que se guardan los registros del procesador en la pila del sistema.
12. Los dispositivos externos se comunican con el Sistema Operativo a través de llamadas al sistema.

[Interfaz con el Sistema Operativo]

13. Los programas se comunican con todos los servicios del Sistema Operativo a través de las órdenes del Shell.
14. *Las interfaces gráficas que proporcionan los sistemas son programas que usan llamadas al sistema para pedir servicios al Sistema Operativo.
15. Un sistema operativo puede tener varias interfaces finales, pero sólo una interfaz de programación (API).
16. Las interfaces de usuario final, como el intérprete de órdenes o la interfaz gráfica, se ejecutan en supervisor puesto que forman parte del sistema operativo.

[Evolución de los Sistemas Operativos]

17. Linux es un sistema operativo basado en Unix, creado en los 70
18. *La multiprogramación y el tiempo compartido se comenzó a desarrollar en los años 60

19. *Ken Thompson y Dennis Ritchie diseñaron UNIX y lo implementaron en lenguaje C.
20. El desarrollo y expansión del Unix se lleva a cabo en los 70

[Multiprogramación]

21. * La ejecución paralela de instrucciones en el procesador y operaciones en los dispositivos posibilita la multitarea.
22. *Uno de los fundamentos de la multitarea es el reparto del uso del procesador entre varios procesos.
23. Los sistemas Multiusuario están soportados por las técnicas de Multiprogramación y Tiempo Compartido.
24. *En un sistema operativo con Tiempo Compartido, cada interrupción de reloj produce la activación del Sistema Operativo.

[Ciclo de vida de un proceso] Sistema con modelo de 7 estados

25. Los procesos en estado listo pasarán a bloqueados si se produce algún evento para ello.
26. Hay una única cola que contiene todos los procesos bloqueados en espera de cualquier evento.
27. *Cuando un proceso abandona el procesador, el sistema operativo debe modificar información del BCP
28. Un proceso en estado bloqueado suspendido puede pasar a ejecución en cuanto lo carguen en memoria principal.

[Gestión de una interrupción] Supongamos un SO que se ejecuta en el Espacio de Direcciones del Proceso (dentro de los procesos)

29. Cuando se produce una interrupción la dirección de la rutina de gestión de interrupciones se obtiene a partir de la tabla de descriptores de interrupciones (o tabla de vectores de interrupciones).
30. Cuando se produce una interrupción, el hardware automáticamente salva todos los registros en la pila del sistema.
31. *Cuando se produce una interrupción se sustituyen los punteros a la pila del proceso de usuario por los punteros a la pila de control del sistema operativo.
32. *Cuando se produce una interrupción que implica un cambio de proceso el sistema operativo almacena información en el PCB del proceso interrumpido.

[Cambio de contexto]

33. En un sistema operativo que se ejecuta como parte del proceso de usuario interrumpido, cada interrupción produce cambio de contexto
34. En un sistema operativo que se ejecuta como núcleo independiente, cada interrupción produce un cambio de proceso.
35. *En un sistema operativo que se ejecuta como parte de los procesos de usuario, el cambio de contexto sólo se realiza si hay cambio de proceso.
36. *En un sistema operativo que se ejecuta como parte de los procesos de usuario un cambio de contexto implica guardar los valores de los registros del procesador en el PCB del proceso que abandona y cargar los valores de éstos a partir del PCB del proceso que accede al procesador

[Hilos]

37. *Los hilos de un mismo proceso comparten todo el espacio de direcciones asignado al proceso.
38. *Los hilos de un mismo proceso tienen cada uno su propia pila de ejecución.
39. Los hilos de un mismo proceso comparten los valores de los registros del procesador guardados en el BCP.
40. Si se utilizan hilos implementados a nivel de usuario, cada vez que se crea un hilo se produce una activación del sistema operativo.

[Políticas de planificación]

41. *En un sistema operativo que soporta hilos, son los hilos de manera individual los que cambian de estado, no los procesos globalmente.
42. Las políticas de planificación con prioridades dinámicas con envejecimiento pueden producir inanición.
43. *En un sistema que use turno rotatorio (con cuanto de tiempo) el proceso o hilo abandona el procesador siempre que agote el cuanto de tiempo asignado y pasa a estado bloqueado
44. *En Windows se usa una política de prioridades dinámicas en la que se favorece a los procesos con mucha entrada/salida.

[Planificación de procesos]

45. La política de planificación apropiativa (o con requisamiento) permite intercambiar procesos temporalmente a memoria secundaria.
46. *Los sistemas operativos Windows o Linux utilizan prioridades y turno rotatorio (cíclica) al menos.
47. En un sistema con política de planificación con turno rotatorio, los procesos abandonan la CPU en dos únicos casos: cuando se acaba el cuanto de tiempo o bien cuando finalizan
48. En una política de planificación por prioridades con envejecimiento, se intenta primar a los procesos con más e/s que a los procesos con más CPU

[Planificación]

Sea un conjunto de procesos p1, p2, p3, p4 y p5. Todos llegan en el instante 0, pero en el orden indicado. Tienen una duración (en msg) respectivamente de 100, 50, 25, 75, 15. Tienen una prioridad, respectivamente, de 10, 1, 1, 5, 7 (a mayor número, mayor prioridad).

49. Con SJF el proceso p1 tiene un tiempo de espera de 75 msg.
50. Con Turno Rotatorio con cuanto 1 el proceso con mayor tiempo de ejecución es el p4.
51. *Utilizando FCFS (FIFO) el proceso con menor tiempo de retorno (ejecución) es el p1.
52. Con prioridades no apropiativas el proceso con menor tiempo de espera es el p2.

[Planificación en Multiprocesadores]

53. El multiprocesamiento simétrico implica que se dedica un procesador a ejecutar el sistema operativo, y el resto para los procesos de los usuarios.
54. *En un sistema multiprocesador con una cola única de planificación se favorece el equilibrio de carga.

55. *En un sistema multiprocesador con múltiples colas de planificación se favorece la afinidad del procesador.
56. * El uso de cola única en sistemas con mucha carga puede provocar cuello de botella.

[Concurrencia]

57. Si se desea comunicar procesos en la misma máquina resulta más rápido usar tuberías que otros mecanismos.
58. Si se desea comunicar procesos en la misma máquina resulta más rápido usar mensajes que otros mecanismos.
59. *Si se desea comunicar procesos en la misma máquina resulta más rápido usar memoria compartida que otros mecanismos
60. Si se desea comunicar procesos en la misma máquina resulta más rápido usar semáforos que otros mecanismos

[Concurrencia] Sean dos hilos que usan una variable común para almacenar información compartida y necesaria por ambos, de forma que uno de ellos no debe leer la información de la variable hasta que el otro haya escrito en ella el dato adecuado

61. Este problema se clasifica como el problema de “los lectores y escritores”.
62. *Este problema se clasifica como el problema de “el productor - consumidor”.
63. Este problema se clasifica como el problema de “cliente - servidor”.
64. Este problema se clasifica como el problema de “recursos compartidos o problema de los filósofos”.

[Concurrencia]

65. Si se desean sincronizar procesos en máquinas diferentes se usan semáforos.
66. Si sólo existe un procesador, no es posible la concurrencia de procesos
67. *La sección crítica es un fragmento de código que contiene un recurso compartido por varios procesos o hilos y cuyo acceso debe ser controlado.
68. *La exclusión mutua implica la ejecución de manera atómica de la sección crítica de cada proceso o hilo

[Mecanismos de comunicación]

69. Tuberías y memoria compartida son mecanismos de comunicación para procesos situados en máquinas diferentes.
70. *Entre hilos del mismo proceso no es preciso un mecanismo de comunicación gestionado por el sistema operativo.
71. *Semáforos y mutex sirven para sincronizar pero no para comunicar.
72. Si se desean comunicar procesos en máquinas diferentes se usan semáforos

[Semáforos y mutex]

73. *Si un hilo ejecuta un lock(m) sobre un mutex que inicialmente está a 0, el sistema operativo pasa al hilo al estado bloqueado.
74. Si un proceso ejecuta un wait(s) sobre un semáforo que inicialmente está a 1, el sistema operativo pasa al proceso al estado bloqueado.
75. *Un proceso bloqueado en una cola de un semáforo puede ser desbloqueado con una llamada signal de otro proceso que use el mismo semáforo

76. *Un proceso bloqueado en una cola de un semáforo, puede ser desbloqueado por otro proceso que tenga acceso a dicho semáforo.

[Señales]

77. *El estándar POSIX define señales como mecanismo de sincronización pero no ocurre lo mismo con la interfaz API de Windows.
78. *Las señales pueden funcionar de manera síncrona o asíncrona.
79. *Kill es una llamada al sistema que envía una señal a un proceso.
80. Las señales sirven tanto para comunicación como para sincronización.

[Tuberías]

81. *Las tuberías son útiles para dar solución a problemas de productores / consumidores
82. Las tuberías sólo sirven para comunicar información, habrá que incluir también un mecanismo de sincronización adicional.
83. Al escribir en una tubería vacía, el proceso queda bloqueado
84. *Al leer de una tubería vacía, el proceso queda bloqueado

[Políticas de planificación] Sean dos procesos p1 y p2 con duración de 10 msg y prioridades 1 y 5, respectivamente (a mayor número mayor prioridad). Si el p1 llega en el instante 0 y el p2 en el instante 1 (se supone que los procesos no se bloquean en ningún momento).

85. Con Turno Rotatorio con cuanto 5 el tiempo de retorno de p1 es de 15 ms.
86. *Con Turno Rotatorio con cuanto 5 el tiempo de retorno de p2 es de 19ms.
87. *Con prioridades apropiativas el tiempo de espera del proceso 2 es 0.
88. Con prioridades apropiativas el tiempo de espera del proceso 1 es 0.

[Llamadas al Sistema para creación de procesos] Sea el siguiente código con llamadas fork y exec. Sea P el proceso que ejecuta este código

```
int cuenta = 5;
if (fork() == 0)
{

    printf("%d ", cuenta);
}
else
{
       fork();
       cuenta ++;
       printf("%d ", cuenta);


}
exec("fichero ejecutable");
```

89. *A partir del proceso P que ejecuta este código se generan 2 procesos más (sin contar a P)
90. *Una posible salida por pantalla es "656 "
91. *Todos los procesos que se generan son unos hijos de P
92. El fichero ejecutable se ejecutará 2 veces

[Concurrencia] Se desea sincronizar dos procesos de tal modo que, como resultado de su ejecución, uno escriba en un fichero "Hola que tal" y el otro lea este texto y lo muestre por pantalla.

```
fd file;
char buffer[128];

file = fopen("r+", "file.dat");
if (fork() != 0){
       wait(sem1);
       buffer = "hola que tal";
       write(file,buffer);
       signal(sem1);
       signal(sem2);
}
else{
       wait(sem2);
       read(file,buffer);
       printf(buffer);
}
```

93. *El semáforo sem2 debe inicializarse a 0 y sem1 a 1 para obtener el resultado esperado
94. *Si se elimina el semáforo sem1, y se inicializa sem2 a 0, se obtiene el resultado esperado
95. Para que la sincronización sea correcta falta que el proceso hijo finalice con un signal (sem1)
96. *Si se sustituye el fichero por una tubería, y la operación read por una de lectura en la tubería y write por una de escritura en la tubería, no sería necesario ningún semáforo para lograr la sincronización

[Concurrencia] Sean los siguientes procesos que ofrece un banco a sus clientes para que puedan meter y sacar dinero cuando lo deseen. Cada método se ejecuta en hilos diferentes y pueden ser lanzados simultáneamente.

```
void Ingresar(int cantidadIngresar, Cuenta miCuenta){
              wait(s);
              miCuenta = miCuenta + cantidadIngresar;
              signal(s);

       }

void Sacar(int cantidadSacar, Cuenta miCuenta){
              Wait (t)
              miCuenta = miCuenta – cantidadSacar ;
              signal(t)
       }
```

97. *Las operaciones deben hacerse con el mismo semáforo en ambos métodos para que no se produzcan condiciones de carrera.
98. Será necesario cambiar signal(s) en ingresar por signal(t) y signal(t) en sacar por signal(s) para que no se produzcan condiciones de carrera.
99. Si S=1 y t=0 inicialmente, se produce interbloqueo
100. Los semáforos s y t deben inicializarse a 1 para que no se produzcan condiciones de carrera.