SISTEMAS OPERATIVOS II **Junio 2012**

Material permitido: **Calculadora NO programable** Tiempo: **2 horas**	**Aviso 1**: Todas las respuestas deben estar debidamente **razonadas**. **Aviso 2**: Escriba sus respuestas con una **letra lo más clara posible**. **Aviso 3**: Evite los **tachones**. **Aviso 4**: Notificación de la salida de las calificaciones, solución del examen y fecha de revisión en la página web de la asignatura: **http://www.uned.es/533032/**

ESTE EXAMEN CONSTA DE 5 PREGUNTAS

Preguntas 1 a 4

1. Explique **razonadamente** si las siguientes afirmaciones son verdaderas o falsas:

I) (*1 p*) El núcleo de UNIX realiza la invocación del algoritmo `wakeup()` únicamente dentro de los algoritmos asociados a las llamadas al sistema.

II) (*1 p*) La tabla de regiones es una estructura local a cada proceso que contiene una entrada por cada región asociada al proceso (código, datos, pila de usuario y memoria compartida (si existiese)).

2. (*2 p*) Enumere y explique **brevemente** las tres mecanismos de sincronización presentes en las distribuciones clásicas de UNIX.

3. (*2 p*) Dibuje un diagrama, **adecuadamente rotulado**, que esquematice las principales acciones que realiza el núcleo durante la ejecución del algoritmo `exec()`.

4. Supóngase que el directorio de trabajo actual de un usuario contiene tres ficheros ordinarios (`datos`, `fotos_mul`, `listas.txt`) y tres subdirectorios (`fotos, videos, correos`).

a) (*0.5 p*) ¿Qué orden se debe teclear desde la línea de comandos ($) para conocer la ruta absoluta del directorio de trabajo actual?

b) (*0.5 p*) ¿Qué orden se debe teclear para mover al subdirectorio `fotos` los ficheros `datos` y `fotos`?

c) (*0.5 p*) ¿Qué orden se debe teclear para que el directorio de trabajo actual pase a ser el subdirectorio `fotos`?

Material permitido: **Calculadora NO programable** Tiempo: **2 horas**	**Aviso 1**: Todas las respuestas deben estar debidamente **razonadas**. **Aviso 2**: Escriba sus respuestas con una **letra lo más clara posible**. **Aviso 3**: Evite los **tachones**. **Aviso 4**: Notificación de la salida de las calificaciones, solución del examen y fecha de revisión en la página web de la asignatura: **http://www.uned.es/533032/**

ESTE EXAMEN CONSTA DE 5 PREGUNTAS

Pregunta 5

5. Al compilar el código C del programa que se muestra al final se crea el ejecutable `j12`. Supóngase que al invocar `j12` desde la línea de órdenes del terminal (`$`) se le asocia el *pid* 356. Supóngase además que la asignación de los *pid* de los procesos hijos, si se llegaran a crear, se realiza mediante la expresión *pid_hijo=pid_padre+h*, donde *h*=1, 2, 3,... hace referencia al orden de creación del proceso hijo, es decir, *h*=1 es el primer hijo creado, *h*=2 es el segundo hijo creado, etc. Suponer además que el intérprete de comandos desde donde se lanza `j12` tiene asociado el *pid* 250. Conteste razonadamente a los siguientes apartados:

a) *(1 p)* Explique el significado de las cinco sentencias enumeradas (**[ ]**) del código del ejecutable `j12`.

b) *(1.5 p)* Explique **detalladamente** el funcionamiento de este programa si se invoca desde el interprete de comandos mediante la orden `$ j12 3`

```
[1]  main(int argc, char *argv[])
     {

          int a,b=0,c,d;

          if (argc!=2)
          {
               exit(4);
          }
          else
          {
[2]            a=atoi(argv[1]);
[3]            while (fork()==0 && b!=a)
               {
                    printf("\nMensaje 1[%d]\n", getpid());
                    b=b+1;
               }
          }
[4]       c=wait(&d);
[5]       printf("\nMensaje 2[%d]=%d\n", getpid(),getppid());
     }
```

SOLUCION EXAMEN JUNIO 2012

1. Explique **razonadamente** si las siguientes afirmaciones son verdaderas o falsas:

I) (*1 p*) El núcleo de UNIX realiza la invocación del algoritmo `wakeup()` únicamente dentro de los algoritmos asociados a las llamadas al sistema.

II) (*1 p*) La tabla de regiones es una estructura local a cada proceso que contiene una entrada por cada región asociada al proceso (código, datos, pila de usuario y memoria compartida (si existiese)).

Solución:

I) Típicamente la invocación del algoritmo `wakeup()` se realiza dentro de algún otro algoritmo del núcleo como los asociados a las llamadas al sistema o las rutinas de manipulación de interrupciones. El núcleo también llama al algoritmo `wakeup()` cuando genera una señal para un proceso en estado dormido interrumpible, siempre y cuando el proceso no ignore ni tenga bloqueado dicho tipo de señales. En conclusión la afirmación es **FALSA**.

II) La *tabla de regiones* es una estructura global del núcleo que contiene una entrada por cada región asignada por el núcleo. En conclusión la afirmación es **FALSA**.

2. (*2 p*) Enumere y explique **brevemente** las tres mecanismos de sincronización presentes en las distribuciones clásicas de UNIX.

Solución:

En las distribuciones clásicas de UNIX existían principalmente tres mecanismos de sincronización:

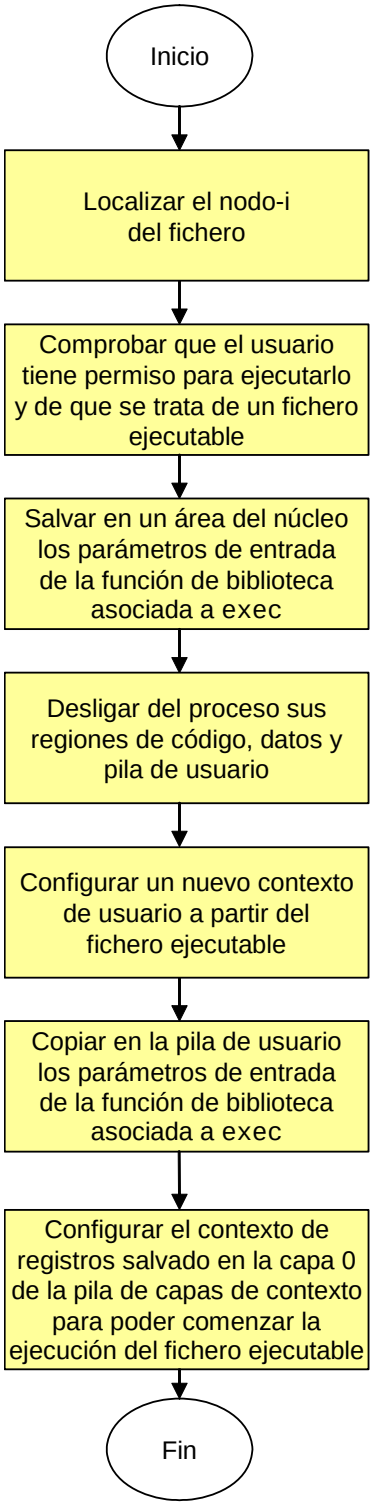
- *Núcleo no expropiable*. Cualquier proceso ejecutándose en modo núcleo continuará ejecutándose, incluso aunque su cuanto haya expirado, hasta que vuelva a modo usuario o entre en el estado dormido en espera de algún recurso que se encuentra ocupado. Esto permite al código del núcleo manipular las estructuras de datos sin necesidad de bloquearlas, sabiendo que ningún otro proceso podrá acceder a ellas hasta que el proceso actual haya terminado de utilizarlas y esté listo para ceder el núcleo en un estado consistente. La no expropiación del núcleo es una herramienta de sincronización bastante útil para un amplio rango de situaciones.

- *Bloqueo de interrupciones*. Aunque el proceso actualmente ejecutándose en modo núcleo no pueda ser expropiado sí que puede ser interrumpido. Las interrupciones son una parte fundamental de la actividad del sistema y normalmente requieren ser atendidas urgentemente. El manipulador de las interrupciones puede manipular las mismas estructuras de datos con las que el proceso actual estaba trabajando, lo que puede producir una corrupción de los datos. Por lo tanto el núcleo debe sincronizar el acceso a los datos que son utilizados tanto por el código normal del núcleo como por el manipulador de interrupciones. UNIX resuelve este problema suministrando un mecanismo para bloquear (enmascarar) las interrupciones mediante la manipulación del *npi*.

- *Uso de los indicadores bloqueado y deseado.* UNIX asocia dos indicadores, *bloqueado* y *deseado*, a cada recurso compartido. Cuando un proceso A desea acceder a un recurso compartido, como un buffer, primero el núcleo comprueba el indicador *bloqueado*. Si no está activado, lo activa y procede a usar el recurso. Si un segundo proceso B intentara acceder al mismo recurso, se encontraría con el indicador *bloqueado* activado y debería entrar en el estado dormido hasta que el recurso quedase disponible. Antes de colocar a dicho proceso en el estado dormido el núcleo activa el indicador *deseado*. Cuando el primer proceso A ha terminado de usar el recurso, el núcleo desactiva el indicador *bloqueado* y comprueba el indicador *deseado*. Si se encuentra activado, eso significará que al menos un proceso se encuentra esperando para usarlo. En ese caso, examina la lista de procesos dormidos y despierta a estos procesos. Cuando uno de ellos sea planificado para ser ejecutado, el núcleo comprobara de nuevo el indicador de *bloqueado* y encontrará que está desactivado, entonces lo activará y procederá a usar el recurso.

3. (*2 p*) Dibuje un diagrama, **adecuadamente rotulado**, que esquematice las principales acciones que realiza el núcleo durante la ejecución del algoritmo `exec()`.

Solución:



4. Supóngase que el directorio de trabajo actual de un usuario contiene tres ficheros ordinarios (`datos`, `fotos_mul`, `listas.txt`) y tres subdirectorios (`fotos`, `videos`, `correos`).

a) (*0.5 p*) ¿Qué orden se debe teclear desde la línea de comandos ($) para conocer la ruta absoluta del directorio de trabajo actual?

b) (*0.5 p*) ¿Qué orden se debe teclear para mover al subdirectorio `fotos` los ficheros `datos` y `fotos`?

c) (*0.5 p*) ¿Qué orden se debe teclear para que el directorio de trabajo actual pase a ser el subdirectorio `fotos`?

Nota: En el apartado *b* del enunciado de esta pregunta existe una pequeña errata fácilmente detectable, donde pone "los ficheros `datos` y `fotos`" debe poner "los ficheros `datos` y `fotos_mul`"

Solución:

a) Para conocer la ruta absoluta del directorio de trabajo actual se debe teclear la orden:

```
$ pwd
```

b) Para mover al subdirectorio `fotos` los ficheros `datos` y `fotos_mul` se debe teclear la orden

```
$ mv datos fotos_mul fotos
```

b) Para que el directorio de trabajo actual pase a ser el subdirectorio `fotos` se debe teclear la orden

```
$ cd fotos
```

5. Al compilar el código C del programa que se muestra al final se crea el ejecutable `j12`. Supóngase que al invocar `j12` desde la línea de órdenes del terminal (`$`) se le asocia el *pid* 356. Supóngase además que la asignación de los *pid* de los procesos hijos, si se llegaran a crear, se realiza mediante la expresión *pid_hijo=pid_padre+h*, donde *h*=1, 2, 3,... hace referencia al orden de creación del proceso hijo, es decir, *h*=1 es el primer hijo creado, *h*=2 es el segundo hijo creado, etc. Suponer además que el intérprete de comandos desde donde se lanza `j12` tiene asociado el *pid* 250. Conteste razonadamente a los siguientes apartados:

a) *(1 p)* Explique el significado de las cinco sentencias enumeradas (**[ ]**) del código del ejecutable `j12`.

b) *(1.5 p)* Explique **detalladamente** el funcionamiento de este programa si se invoca desde el interprete de comandos mediante la orden `$ j12 3`

```
[1]   main(int argc, char *argv[])
      {

            int a,b=0,c,d;

            if (argc!=2)
            {
                  exit(4);
            }
            else
            {
[2]               a=atoi(argv[1]);
[3]               while (fork()==0 && b!=a)
                  {
                        printf("\nMensaje 1[%d]\n", getpid());
                        b=b+1;
                  }
            }
[4]         c=wait(&d);
[5]         printf("\nMensaje 2[%d]=%d\n", getpid(),getppid());
}
```

Solución:

a) El significado de cada una de las sentencias marcadas con **[ ]** de este código es el siguiente:

[1] Definición de la función principal `main` del programa con dos argumentos formales. El primer argumento `argc` es un número entero que contiene el número de argumentos de la línea de comandos. El segundo argumento `*argv[]` es un array de punteros a caracteres. Cada puntero del array apunta a un argumento de la línea de órdenes.

[2] Función de librería `atoi` que convierte a número entero el carácter ASCII apuntado argv[1] que recibe como entrada. El resultado se guarda en la variable `a`

[3] Bucle `while` cuya expresión de evaluación para que se continúe la ejecución del bucle es

que la llamada al sistema `fork` devuelva el valor 0 y que la variable `b` no tome el valor `a`. Recuérdese que `fork` crea un proceso hijo y devuelve 0 para el proceso hijo y el pid del hijo al proceso padre.

[4] Llamada al sistema `wait` para sincronizar la ejecución del proceso A que la invoca con la terminación de un proceso hijo. Posee un único parámetro es la dirección de la variable entera `d` donde se almacenará el código de retorno para el proceso A generado por el algoritmo `exit()`al terminar el hijo. Si la llamada al sistema se ejecuta con éxito devuelve el pid del proceso que ha terminado. En caso contrario, devuelve -1. Dicho valor de salida se almacena en la variable `c`.

[5] Función `printf` para escribir en el dispositivo de salida estándar (típicamente el monitor) el resultado de la llamada al sistema `getpid()`y de la llamada al sistema getppid(). Estas llamadas devuelven, si se ejecutan con éxito, el *pid* del proceso que la invoca y el pid de su proceso padre, respectivamente.

La escritura en el dispositivo de salida estándar se realiza de acuerdo con la siguiente cadena de control

```
"\nMensaje 2[%d]=%d\n"
```

Es decir, en pantalla se mostraría lo siguiente: un salto de línea, el mensaje `Mensaje 2`, la apertura de un corchete, el valor expresado en entero decimal del resultado de `getpid()`, el cierre de un corchete, un signo de igual, el valor expresado en entero decimal del resultado de `getppid()` y un salto de línea.

b) Al escribir la orden `$ j12 3` se comienza a ejecutar el programa `j12.` Supóngase que a la ejecución de dicho programa se le asocia el proceso A, cuyo *pid*=356 según el enunciado.

En primer lugar se comprueba si el número de argumentos con que ha sido invocado `j12` es distinto de dos. Recuerde que el nombre del programa cuenta como argumento. Puesto que `j12 3` tiene dos argumentos la condición del `if` no se cumple y se pasan a ejecutar las sentencias del `else`.

En segundo lugar se invoca a la función de librería `atoi` que convierte el segundo argumento (‘3’) de la invocación de `j12` a número entero y lo almacena en la variable a.

En tercer lugar se ejecuta un bucle while cuya expresión de evaluación para que se continúe la ejecución del bucle es que la llamada al sistema `fork` devuelva el valor 0 y que la variable `b` no tome el valor `a`. Recuérdese que `fork` crea un proceso hijo y devuelve 0 para el proceso hijo y el pid del hijo al proceso padre.

La evaluación de esta condición da como resultado la creación de un hijo B con pid=357, que al cumplir que `fork` le ha devuelto 0 y que b es distinta de 3 ejecuta el contenido del bucle. El proceso padre A no ejecuta el bucle por que para él `fork` no ha devuelto un 0, sino 357, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo B termine.

El proceso B muestra en pantalla el mensaje

```
Mensaje 1[357]
```

e incrementa en una unidad el valor de la variable `b` que ahora toma el valor 1. A continuación evalúa la condición de continuación del bucle que da como resultado la creación de un hijo C con pid=358, que al cumplir que `fork` le ha devuelto 0 y que b es distinto de 3 ejecuta el contenido del bucle. El proceso padre B no ejecuta el bucle por que para él `fork` no ha devuelto un 0, sino 358, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo C termine.

El proceso C muestra en pantalla el mensaje

```
Mensaje 1[358]
```

e incrementa en una unidad el valor de la variable `b` que ahora toma el valor 2. A continuación evalúa la condición de continuación del bucle que da como resultado la creación de un hijo D con pid=359, que al cumplir que `fork` le ha devuelto 0 y que b es distinto de 3 ejecuta el contenido del bucle. El proceso padre C no ejecuta el bucle por que para él `fork` no ha devuelto un 0, sino 359, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo D termine.

El proceso D muestra en pantalla el mensaje

```
Mensaje 1[359]
```

e incrementa en una unidad el valor de la variable `b` que ahora toma el valor 3. A continuación evalúa la condición de continuación del bucle que da como resultado la creación de un hijo E con pid=360, sin embargo como b es igual 3 no ejecuta el contenido del bucle sino que ejecuta la llamada al sistema `wait`, pero como E no tiene hijos no se suspende sino que imprime en pantalla el mensaje

```
Mensaje 2[360]=359
```

y finaliza.

Por su parte el proceso padre D tampoco ejecuta el bucle por que para él `fork` no ha devuelto

un 0, sino 360, y además b=3 por lo que ejecuta una llamada al sistema wait y se queda a la espera de que su proceso hijo E termine. Como éste ya ha finalizado imprime en pantalla el mensaje

```
Mensaje 2[359]=358
```

y finaliza.

Como su hijo D ya ha terminado se despierta al proceso C que imprime en pantalla el mensaje

```
Mensaje 2[358]=357
```

y finaliza.

Como su hijo C ya ha terminado se despierta al proceso B que imprime en pantalla el mensaje

```
Mensaje 2[357]=356
```

y finaliza.

Finalmente como su hijo B ya ha terminado se despierta al proceso A que imprime en pantalla el mensaje

```
Mensaje 2[356]=250
```

y finaliza.