SISTEMAS OPERATIVOS II **Septiembre 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**: 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*) La operación de *asignar una región* consiste en asociar a una región una entrada de la tabla de regiones por proceso.

II) (*1 p*) En el UNIX SVR3 la ejecución del manipulador de fallos de protección siempre produce el envío por parte del núcleo de una señal SIGBUS al proceso que provocó el fallo.

2. (*2 p*) Explique **razonadamente** los problemas de seguridad y funcionalidad que posee el sistema de ficheros *s5fs*.

3. (*2 p*) Enumere cuatro de las principales tareas que realiza la rutina de tratamiento de la interrupción del reloj en UNIX.

4. (*1.5 p*) Supóngase una cola de cinco mensajes en UNIX cuyos tipos son {3, 1, 4, 1, 5} y que el mensaje situado más a la izquierda es el primero de la cola, es decir, el mensaje más antiguo. Si un proceso solicita extraer un mensaje con una llamada al sistema `msgrcv` con argumento `msgtipo=x`, determinar el tipo de mensaje que extrae el núcleo en los siguientes casos: a) `x=1`. b) `x=0`. c) `x=-3`.

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**: 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. Cuando se compila el siguiente código C se crea un ejecutable que se llama `s12`. Conteste **razonadamente** a los siguientes apartados:

a) (*1 p*) Explicar el significado de las cinco sentencias enumeradas (**[ ]**) de este código.

b) (*1.5 p*) Explicar el funcionamiento de este progr ama si se invoca desde la línea de órdenes (`$`) de las siguientes formas:

1) `$ s12 fich1.c fich2.c`
2) `$ s12 prueba.txt`

Datos: Los ficheros `fich1.c fich2.c prueba.txt` ya están creados y el usuario posee los permisos adecuados.

```
      #include <fcntl.h>
[1]   main(int argc, char *argv[])
     {
          int var1,var2;
          char var3[50];
[2]       if (argc==1||argc>2)
          {
               printf("\nMensaje DRF\n");
[3]            exit(0);
          }
          else
          {
               var1=open(argv[1],O_RDONLY);
               if (var1==-1)
               {
                    printf("\nMensaje PLO\n");
[4]                 if (fork()==0)
                    {
                         printf("\nMensaje SRT\n");
                         for(;;);
                    }
               }
               else
               {
[5]                 if (read(var1,var3,50)==-1) printf("\nMensaje ERT\n");
                    close(var1);
                    exit(1);
               }
          }
     }
```

SOLUCIÓN EXAMEN SEPTIEMBRE 2012

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

I) (*1 p*) La operación de *asignar una región* consiste en asociar a una región una entrada de la tabla de regiones por proceso.

II) (*1 p*) En el UNIX SVR3 la ejecución del manipulador de fallos de protección siempre produce el envío por parte del núcleo de una señal SIGBUS al proceso que provocó el fallo.

Solución:

I) La operación *asignar una región* consiste en eliminar la primera entrada disponible de la lista enlazada de regiones libres y situarla en la lista de regiones activas. En conclusión la afirmación es **FALSA**.

II) En el UNIX SVR3 un *fallo de protección* se produce cuando un proceso intenta acceder a una página válida cuyos bits de protección no permiten acceder a la página (por ejemplo si un proceso intenta escribir en su región de código). Asimismo un proceso puede incurrir en un fallo de protección cuando intenta escribir una página cuyo bit *copiar al escribir* esté activado.

La ejecución del manipulador de fallos de protección **únicamente** produce el envío por parte del núcleo de una señal SIGBUS al proceso que provocó el fallo si se da la primera causa, es decir, si se intenta acceder a una página válida cuyos bits de protección no permiten acceder a la página. En conclusión la afirmación es **FALSA**.

2. (*2 p*) Explique **razonadamente** los problemas de seguridad y funcionalidad que posee el sistema de ficheros *s5fs*.

Solución:

El principal problema de seguridad de un sistema de archivos s5fs proviene del superbloque que contiene información vital sobre el sistema de ficheros como por ejemplo la lista de bloques libres y el tamaño de la lista de nodos-i libres. Cada sistema de ficheros contiene una única copia de su superbloque. Si la copia está corrupta, todo el sistema de ficheros no podrá ser utilizado.

Los problemas de funcionalidad que presenta un sistema de ficheros s5fs son los siguientes: en un sistema s5fs el tamaño de los nombres de los ficheros está restringido a 14 caracteres. Además el número de nodos-i (y en consecuencia de archivos) está limitado a 65535. Estas limitaciones quizás no importaban mucho hace algunos años, pero para un sistema operativo viable comercialmente y potente, tales restricciones resultan inaceptables.

3. (*2 p*) Enumere cuatro de las principales tareas que realiza la rutina de tratamiento de la interrupción del reloj en UNIX.

Solución:

Entre las principales tareas que realiza la rutina de tratamiento de la interrupción del reloj en UNIX se encuentran las siguientes:

- Rearmar el reloj hardware si fuera necesario.
- Adaptar las estadísticas de uso de la CPU para el proceso actual, es decir, usando la CPU.
- Enviar una señal SIGXCPU al proceso actual si éste ha excedido su cuota de uso de la CPU, es decir, su cuanto.
- Adaptar el reloj de la hora del día y otros relojes relacionados.
- Comprobación de los callouts.
- Despertar a los procesos del sistema, como por ejemplo el intercambiador o el ladrón de páginas, cuando sea necesario.
- Comprobación de las alarmas.

4. (*1.5 p*) Supóngase una cola de cinco mensajes en UNIX cuyos tipos son {3, 1, 4, 1, 5} y que el mensaje situado más a la izquierda es el primero de la cola, es decir, el mensaje más antiguo. Si un proceso solicita extraer un mensaje con una llamada al sistema `msgrcv` con argumento `msgtipo=x`, determinar el tipo de mensaje que extrae el núcleo en los siguientes casos: a) `x=1`. b) `x=0`. c) `x=-3`.

Solución:

a) Como `msgtipo=x=1>0` se extrae el primer mensaje (el situado más a la izquierda) del tipo 1 que haya en la cola.

b) Como `msgtipo=x=0` se extrae el primer mensaje (el situado más a la izquierda) que haya en la cola independientemente de su tipo. Corresponde al mensaje más antiguo. Luego en este caso se extrae el mensaje de tipo 3.

c) Como `msgtipo=x=-3<0` se extrae el primer mensaje (el situado más a la izquierda) que cumpla que su tipo es menor o igual al valor absoluto de x y a la vez sea el más pequeño de los que hay. En este caso se extrae el primer mensaje que cumpla la siguiente condición:

$$\min\{3,1,4,1,5\} \leq |-3|$$

Luego se extrae el primer mensaje (el situado más a la izquierda) de tipo 1.

5. Cuando se compila el siguiente código C se crea un ejecutable que se llama `s12`. Conteste **razonadamente** a los siguientes apartados:

a) (*1 p*) Explicar el significado de las cinco sentencias enumeradas (**[ ]**) de este código.

b) (*1.5 p*) Explicar el funcionamiento de este programa si se invoca desde la línea de órdenes (`$`) de las siguientes formas:

1) `$ s12 fich1.c fich2.c`
2) `$ s12 prueba.txt`

Datos: Los ficheros `fich1.c fich2.c prueba.txt` ya están creados y el usuario posee los permisos adecuados.

```
        #include <fcntl.h>
[1]     main(int argc, char *argv[])
        {
          int var1,var2;
          char var3[50];
[2]       if (argc==1||argc>2)
          {
              printf("\nMensaje DRF\n");
[3]           exit(0);
          }
          else
          {
              var1=open(argv[1],O_RDONLY);
              if (var1==-1)
              {
                    printf("\nMensaje PLO\n");
[4]                 if (fork()==0)
                    {
                          printf("\nMensaje SRT\n");
                          for(;;);
                    }
              }
              else
              {
[5]                 if (read(var1,var3,50)==-1)
                    {
                          printf("\nMensaje ERT\n");
                    }
                          close(var1);
                          exit(1);
                    }
              }
          }
      }
```

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] Sentencia condicional de tipo `if` que comprueba si se cumple al menos una de las siguientes condiciones que `argc==1` o que `argc>2`.

[3] Llamada al sistema `exit` para finalizar la ejecución del proceso. Posee el parámetro de entrada 5 que será devuelto al proceso padre del proceso invocador, a dicho parámetro se le suele denominar código de retorno para el proceso padre.

[4] Llamada al sistema `fork` para crear un proceso hijo. Se comprueba dentro de una sentencia condicional `if` si la llamada devuelve un valor 0, es decir, si se está ejecutando el proceso hijo.

[5] Sentencia condicional de tipo `if` donde se comprueba si la llamada al sistema `read` para leer 50 bytes del archivo con descriptor `var1` y escribirlos en la posición de memoria `var2` ha devuelto el valor -1, es decir, se ha producido algún error durante la ejecución de la llamada al sistema. En dicho caso se ejecuta la función `printf` que muestra en la salida estándar, por defecto la pantalla, el texto `Mensaje ERT`.

b1) Al escribir la orden `s12 fich1.c fich2.c` se comienza a ejecutar el programa `s12.` Supóngase que a la ejecución de dicho programa se le asocia el proceso A. En primer lugar se comprueba si el número de argumentos con que ha sido invocado `s12` es igual a 1 o mayor de 2. Puesto que `s12 fich1.c fich2.c` tiene tres argumentos (recuérdese que el nombre del programa cuenta como argumento) la condición del `if` se cumple, por lo que se muestra en la pantalla el texto `Mensaje DRF` y se invoca a la llamada al sistema `exit` para terminar la ejecución del proceso.

b2) Al escribir la orden `s12 prueba.txt` se comienza a ejecutar el programa `s12.` Supóngase como antes que a la ejecución de dicho programa se le asocia el proceso A. En primer lugar se comprueba si el número de argumentos con que ha sido invocado `s12` es igual a 1 o mayor de 2. En este caso el número de argumentos es igual a 2 con lo que la condición no se cumple.

A continuación se invoca a la llamada al sistema `open` para abrir el archivo `prueba.txt` con permisos de solo lectura. Si la llamada se ejecuta con éxito guarda en la variable `var1` el descriptor del archivo. En caso de error devuelve el valor -1.

Posteriormente se comprueba si se ha producido algún error, de acuerdo con el enunciado el archivo `prueba.txt` ya está creado y el usuario posee los permisos adecuados por lo que la llamada al sistema `open` se ejecutaría sin errores. En dicho caso, a continuación se invoca a la llamada al sistema `read` para leer 50 bytes del archivo y escribirlos en la posición de memoria `var2`. Además se comprueba si se ha producido algún error durante la ejecución de `read`. En caso afirmativo se mostraría en la pantalla el texto `Mensaje ERT`.

Si la llamada `read` se ejecuta correctamente entonces a continuación se invoca primero a la llamada al sistema `close` para cerrar el archivo y luego a la llamada al sistema `exit` para terminar la ejecución del proceso.