SOLUCION EXAMEN JUNIO 2013

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

I) (*1 p*) El superbloque de un sistema de archivos s5fs contiene, entre otras informaciones, la lista de nodos-i.

II) (*1 p*) La rutina de tratamiento de las interrupciones del reloj de un sistema UNIX se encarga, entre otras acciones, de invocar a los callouts.

Solución:

I) En un sistema de archivos *s5fs* la lista de nodos-i se implementa fuera del superbloque. En conclusión la afirmación es **FALSA**.

II) La rutina de tratamiento de la interrupción del reloj no invoca directamente a los callouts. En cada tic, la rutina comprueba si se debe realizar algún callout. Si es así, activa un indicador para indicar que el manipulador de los callouts debe ser ejecutado. El sistema comprueba este indicador cuando retorna a su *npi* base, si está activado, invoca al manipulador de los callouts, el cual invocará al callout que sea necesario. Por lo tanto, un callout se ejecutará tan pronto como sea posible, pero sólo cuando todas las interrupciones que estaban pendientes hayan sido atendidas. En conclusión la afirmación es **FALSA**.

2. Conteste **razonadamente** a las siguientes cuestiones relativas a un sistema UNIX SVR3:
a) (*0.5 p*) ¿Qué se entiende por región de un proceso?
b) (*1.5 p*) ¿Qué es y qué información contiene la tabla de regiones?

Solución:

a) Una *región de un proceso* es un área de direcciones contiguas de memoria virtual.

b) La *tabla de regiones* es una estructura global del núcleo que contiene una entrada por cada región asignada por el núcleo. Cada entrada de esta tabla contiene la información necesaria para describir una región, como por ejemplo:

- Un puntero al nodo-i del fichero cuyo contenido fue cargado dentro de la región.
- El tipo de región (código, datos, pila de usuario o memoria compartida).
- El tamaño de la región.
- La localización de la región en memoria principal, típicamente un puntero a una *tabla de páginas*.
- El estado de la región, que puede ser una combinación de: bloqueada, bajo demanda, en proceso de ser cargada en memoria y válida (cargada en memoria).
- El contador de referencias, que indica el número de procesos que están referenciando a una región.

3. (*2 p*) Enumere las acciones que realiza el manipulador de fallos de protección en un sistema UNIX SVR3 supuesto que el bit *copiar al escribir* asociado a la página que provoca el fallo está desactivado.

Solución:

Para tratar un fallo de protección el núcleo invoca al manipulador de fallos de protección, que necesita como argumento de entrada la dirección virtual que al ser accedida ha provocado el fallo de protección. Esta dirección es suministrada al núcleo por la MMU. El núcleo al ejecutar el manipulador en primer lugar busca la región, la entrada de la tabla de páginas, la entrada de la tabla de descriptores de bloques de disco y la entrada de la tabla de datos de los marcos de página (*edmp1*) asociadas a dicha dirección. En segundo lugar bloquea la región para que el ladrón de páginas no pueda seleccionar la página para ser intercambiada mientras el manipulador está trabajando sobre ella. A continuación comprueba si el fallo de protección se ha producido porque se ha intentado acceder a una página válida cuyos bits de protección no permiten acceder a la página. En dicho caso envía una señal SIGBUS al proceso que provocó el fallo, desbloquea la región y finaliza. Cuando la señal sea tratada provocará la finalización del proceso.

Las últimas acciones que realiza el manipulador de fallos de protección antes de finalizar su ejecución son: activar los bits de *protección* y el bit *modificada*, recalcular la prioridad del proceso y desbloquear la región que había bloqueado al comienzo de su ejecución.

4. Conteste **razonadamente** a las siguientes cuestiones relativas a la siguiente máscara de modo simbólica:

`-r-S-ws--T`

a) (*1 p*) Explique el significado de esta máscara de modo simbólica.
b) (*0.5 p*) Determinar la máscara de modo expresada en binario equivalente.

Solución:

a) La información contenida en esta máscara de modo simbólica es la siguiente: se trata de un archivo regular. El propietario del archivo solo puede leer el archivo. Los usuarios pertenecientes al grupo propietario del archivo pueden escribir y ejecutar el archivo. El resto de usuarios no pueden acceder al fichero. Además los bits `S_ISUID`, `S_ISGID` y `S_ISVTX` están activados.

b) De acuerdo con el apartado a) la máscara de modo binaria de este archivo es

111 100 011 000

Luego la máscara octal es 7430.

5. Al compilar el código C del programa que se muestra al final se crea el ejecutable `j13`. Supóngase que al invocar `j13` desde la línea de órdenes del terminal (`$`) se le asocia el *pid* 1035. 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 `j13` tiene asociado el *pid* 1001. Conteste razonadamente a los siguientes apartados:

a) *(1 p)* Explique el significado de las sentencias enumeradas (**[ ]**) del código del ejecutable `j13`.

b) *(1.5 p)* Explique **detalladamente** el funcionamiento de este programa si se invoca desde el intérprete de comandos mediante la orden `$ j13 4`

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

  int a,b=0,c,d;

  if (argc!=2)
  {
[1]           exit(2);
  }
  else
  {
        a=atoi(argv[1]);
[2]     while (b!=a && fork()==0)
        {
              b=b+1;
              printf("\nMensaje 1[%d]=%d\n", getpid(),getppid());
[3]           execv("bin/date",0);
        }
  }
[4]  c=wait(&d);
  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] Llamada al sistema `exit` para terminar la ejecución del proceso. Recibe como parámetro de entrada el código de retorno para el proceso padre.

[2] Bucle `while` cuya expresión de evaluación para que se continúe la ejecución del bucle consta de dos. La primera condición es que `b` sea distinto de `a`. La segunda condición es que la llamada al sistema `fork` devuelva el valor 0. Recuérdese que `fork` crea un proceso hijo y devuelve 0 para el proceso hijo y el *pid* del hijo al proceso padre.

[3] Llamada al sistema `execv` para invocar desde el proceso al programa `bin/date`.

[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`.

b) Al escribir la orden `$ j13 4` se comienza a ejecutar el programa `j13.` Supóngase que a la ejecución de dicho programa se le asocia el proceso A, cuyo *pid*=1035 según el enunciado.

En primer lugar se comprueba si el número de argumentos con que ha sido invocado `j13` es distinto de dos. Recuerde que el nombre del programa cuenta como argumento. Puesto que `j12 4` 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 (‘4’) de la invocación de `j13` 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 variable `b` no tome el valor `a` y la llamada al sistema `fork` devuelva el valor 0. 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=1036, que al cumplir que b es distinta de 4 y que `fork` le ha devuelto 0 ejecuta el contenido del bucle. El proceso padre A no ejecuta el bucle por que para él `fork` no ha devuelto un 0, sino 1035, 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 incrementa en una unidad el valor de la variable `b` que ahora toma el valor 1, muestra en pantalla el mensaje

```
Mensaje 1[1036]=1035
```

e invoca a la llamada `execv` para ejecutar el programa `bin/date`. Nótese que la ruta pasada como parámetro es una ruta relativa por lo que el resultado de esta llamada al sistema depende del directorio de trabajo actual del usuario. Se pueden presentar dos casos:

CASO 1: El directorio de trabajo es el directorio raíz `/` o contiene un subdirectorio `bin` con una copia del programa `date`.

En este caso la llamada al sistema `execv` se ejecuta con éxito y se ejecuta el programa `date` que muestra en la salida estándar la fecha y la hora. Una vez terminada la ejecución de este programa el proceso B finaliza ya que sus regiones de código, datos y pila durante el tratamiento de `execv` han sido sustituidas por las del programa invocado.

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

```
Mensaje 2[1035]=1001
```

y finaliza.

CASO 2: El directorio de trabajo no es el directorio raíz `/` o no contiene un subdirectorio `bin` con una copia del programa `date`.

En este caso la llamada al sistema `execv` no puede localizar el programa `date` y devuelve un error. A continuación el proceso B evalúa la condición de continuación del bucle, la primera condición se cumple ya que b=1. Por su parte la evaluación de la segunda condición produce la creación de un proceso hijo C con pid=1037. El proceso padre B no ejecuta el bucle porque para él `fork` no ha devuelto un 0, sino 1037, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo C termine.

Por su parte el proceso C, al cumplir las dos condiciones de ejecución del bucle, incrementa en una unidad el valor de la variable `b` que ahora toma el valor 2, muestra en pantalla el mensaje

```
Mensaje 1[1037]=1036
```

e invoca a la llamada `execv` para ejecutar el programa `bin/date`. Al igual que antes la llamada al sistema `execv` no puede localizar el programa `date` y devuelve un error. A continuación el proceso C evalúa la condición de continuación del bucle, la primera condición se cumple ya que b=2. Por su parte la evaluación de la segunda condición produce la creación de un proceso hijo D con pid=1038. El proceso padre C no ejecuta el bucle porque para él `fork` no ha devuelto un 0, sino 1038, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo D termine.

Por su parte el proceso D, al cumplir las dos condiciones de ejecución del bucle, incrementa en una unidad el valor de la variable `b` que ahora toma el valor 3, muestra en pantalla el mensaje

```
Mensaje 1[1038]=1037
```

e invoca a la llamada `execv` para ejecutar el programa `bin/date`. Al igual que antes la llamada al sistema `execv` no puede localizar el programa `date` y devuelve un error. A continuación el proceso D evalúa la condición de continuación del bucle, la primera condición se cumple ya que b=3. Por su parte la evaluación de la segunda condición produce la creación de un proceso hijo E con pid=1039. El proceso padre D no ejecuta el bucle porque para él `fork` no ha devuelto un 0, sino 1039, por lo que ejecuta una llamada al sistema `wait` y se queda a la espera de que su proceso hijo E termine.

Por su parte el proceso E, al cumplir las dos condiciones de ejecución del bucle, incrementa en una unidad el valor de la variable b que ahora toma el valor 4, muestra en pantalla el mensaje

```
Mensaje 1[1039]=1038
```

e invoca a la llamada `execv` para ejecutar el programa `bin/date`. Al igual que antes la llamada al sistema `execv` no puede localizar el programa `date` y devuelve un error. A continuación el proceso E evalúa la condición de continuación del bucle, la primera condición no se cumple ya que b=4, por ello 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[1039]=1038
```

y finaliza.

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

```
Mensaje 2[1038]=1037
```

y finaliza.

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

```
Mensaje 2[1037]=1036
```

y finaliza.

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

```
Mensaje 2[1036]=1035
```

y finaliza.

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

```
Mensaje 2[1035]=1001
```

y finaliza.