SESSIÓ 3

Dado el siguiente código, y suponiendo que ninguna llamada a sistema devuelve error, ¿qué valor verá el padre en la variable i y qué valor verá el hijo en el punto A marcado con el comentario?

```
i=10;
fork();
/* punto A: valor de i? */
```

Trieu-ne una:
Respostes

- a. El proceso padre verá un 10 y el valor para el hijo está indefinido
- b. Los dos procesos verán un 10
- c. El proceso hijo verá un 10 y el padre no verá la variable
- d. El proceso padre verá un 10 y el hijo no verá la variable

La resposta correcta és: Los dos procesos verán un 10

Dado el siguiente código, suponiendo que ninguna llamada a sistema devolverá error y que lo ejecutamos en una máquina con una sola unidad de cálculo, ¿podemos saber qué proceso ejecutará antes la línea del printf? ¿El padre o el hijo?

```
fork();
printf("soy el primero!")
```

Trieu-ne una:
Respostes

- a. Siempre será el hijo, porque después de la creación el padre cede la cpu al hijo
- b. No lo podemos saber porque depende de la decisión de planificación que tome el sistema
- c. La ejecutarán los dos a la vez
- d. Siempre será el padre, porque después de la creación continua usando la cpu

La resposta correcta és: No lo podemos saber porque depende de la decisión de planificación que tome el sistema

Dado el siguiente código, suponiendo que ninguna llamada a sistema devuelve error, ¿podemos saber qué proceso escribirá antes por pantalla? ¿el padre o el hijo?

```
ret=fork();
if (ret>0){
   waitpid(-1, NULL,0);
   printf("Este mensaje lo escribe %d\n",getpid());
   exit(0);


} else {
   printf("Este mensaje lo escribe %d\n",getpid());
   exit(0);
}wd
```

Trieu-ne una:
Respostes

- a. Siempre escribe primero el hijo porque el padre tienen que ejecutar más instrucciones hasta llegar al printf
- b. No se puede saber, depende de la política de planificación del sistema
- c. Siempre escribe primero el hijo, porque en este código el padre no puede ejecutar el printf hasta que el hijo acaba la ejecución
- d. Siempre escribe primero el padre, porque después de la creación continúa usando la cpu

La resposta correcta és: Siempre escribe primero el hijo, porque en este código el padre no puede ejecutar el printf hasta que el hijo acaba la ejecución

Dado el siguiente código:

```
for (i=0;i<3;i++){
   ret=fork();
   if (ret > 0) {
      execlp("ls","ls",NULL);
}
}
```

Suponiendo que ninguna llamada a sistema devuelve error, ¿cuántos procesos creará?

Trieu-ne una:
Respostes

a. 8

b. 128

c. 1

d. 3

La resposta correcta és: 3

A continuación mostramos el contenido parcial del fichero /proc/21784/status en un momento dado:

```
Name:   prog1
State:  R (running)
Tgid:   21784
Ngid:   0
Pid:    21784
PPid:   21662
TracerPid:  0
Uid:    10014   10014   10014   10014
Gid:    10000   10000   10000   10000
FDSize: 256
(...)
```

¿Qué podemos afirmar del proceso 21784 respecto al momento en el que hemos consultado este fichero?

Trieu-ne una:
Respostes

a. El proceso no estaba consumiendo ni memoria ni cpu en ese momento

b. Con la información mostrada, no podemos saber si estaba consumiendo algo de memoria o cpu en ese momento

c. El proceso estaba consumiento memoria pero no podemos saber si estaba consumiendo cpu

d. El proceso estaba consumiento tanto memoria como cpu

La resposta correcta és: El proceso estaba consumiento tanto memoria como cpu

SESSIÓ 4

Si ejecutamos las siguientes sentencias:

```
struct sigaction trat;
sigset_t mask;
...
trat.sa_handler = func;
sigaction(SIGALRM, &trat, NULL);
```

```
sigemptyset(&mask);
alarm(2);
fork();
sigsuspend(&mask);
```

Cuando pasen 2 segundos después de la llamada a alarm tanto padre como hijo recibirán siempre un SIGALRM y ejecutarán la función func

Trieu-ne una:
Respostes

Vertader

Fals

La resposta correcta és 'Fals'.

Con la siguiente llamada al sistema estamos seleccionando y bloqueando todos los signals, excepto SIGKILL y SIGSTOP que están protegidos por el kernel:

```
sigfillset(&mask);
```

Trieu-ne una:
Respostes

Vertader

Fals

La resposta correcta és 'Fals'.

Considerando que el siguiente código compila y se ejecuta bien:

```
int count = 0;
void func(int sig)
{
count++;
printf("*** Signal %d. Total recibidos %d.\n", sig, count);
kill(getpid(), SIGUSR1);
sleep(10);
printf("*** Fin tratamiento signal %d\n", sig);
}

void main(int argc, char *argv[])
{
struct sigaction trat;
sigset_t mask;

sigemptyset(&mask);
sigaddset(&mask, SIGUSR1);
trat.sa_flags = 0;
trat.sa_mask = mask;
trat.sa_handler = func;
sigaction(SIGALRM, &trat, NULL);
alarm(1);
while (1) ;
}
```

La ejecución finaliza durante el tratamiento del SIGALRM, inmediatamente, después de mostrar el primer mensaje y antes de mostrar el segundo mensaje por pantalla, debido a que se envía un SIGUSR1 a sí mismo

Trieu-ne una:
Respostes

Vertader

Fals

La resposta correcta és 'Fals'.

Dado el siguiente código, suponiendo que ninguna llamada a sistema provoca error, ¿qué podemos asegurar con respecto al SIGALRM que el sistema operativo enviará como consecuencia del alarm(2) que aparece en el código?

```
void funcion(int signum) {
char buf[80];
sprintf(buf, "SIGARLM!\n");
write(1,buf, strlen(buf));
}

main(int argc, char *argv[]){
   struct sigaction trat;
   ...
   trat.sa_handler = funcion;
   sigaction(SIGALRM, &trat, NULL);
   alarm(2);
   fork();
   execlp("./A","A",(char *) NULL);
   waitpid(-1,NULL,0);

}
```

Trieu-ne una o més:
Respostes

- [ ] a. Si el proceso hijo recibe el SIGALRM (debido al alarm(2)) mientras está ejecutando el código del programa "A", si ese código no ha reprogramado el tratamiento del SIGALRM, el proceso hijo morirá
- [x] b. Lo más probable es que el proceso padre reciba el SIGALRM mientras está ejecutando el waitpid y entonces escribirá el mensaje "SIGALRM!\n"
- [ ] c. Podría ocurrir que el proceso hijo recibiera el SIGALRM, debido al alarm(2), antes de ejecutar el execlp y entonces ejecutaría el tratamiento por defecto y moriría
- [ ] d. Si el proceso padre recibe el SIGALRM mientras está ejecutando el código del programa "A", si ese código no ha reprogramado el tratamiento del SIGALRM, el proceso padre morirá
- [ ] e. El proceso padre siempre escribirá el mensaje "SIGALRM!\n" al cabo de 2 segundos de la ejecución del alarm(2)
- [x] f. El alarm(2) que tenemos en este código no provoca que el hijo reciba un SIGALRM
- [ ] g. Si el proceso padre recibe el SIGALRM mientras está ejecutando el programa "A" siempre escribirá el mensaje "SIGALRM!\n"
- [ ] h. Si el proceso hijo recibe un SIGALRM mientras está ejecutando el programa "A" siempre escribirá el mensaje "SIGALRM!\n"

Les respostes correctes són: Si el proceso padre recibe el SIGALRM mientras está ejecutando el código del programa "A", si ese código no ha reprogramado el tratamiento del SIGALRM, el proceso padre morirá, El alarm(2) que tenemos en este código no provoca que el hijo reciba un SIGALRM

Dado el siguiente código, suponiendo que ninguna llamada a sistema provoca error, ¿qué podemos afirmar sobre su ejecución?

```
int pidh = 0;
void func_alrm(int s) {
kill (pidh, SIGKILL);
}
main(int argc, char *argv[]) {
struc sigaction trat;
...
trat.sa_handler = func_alrm;
trat.sa_flags = SA_RESTART;
sigaction(SIGALRM, &trat, NULL);
pidh=fork();
if(pidh == 0) {
while (1) ;
}

alarm(1);
waitpid(-1, NULL, 0);
}
```

Trieu-ne una:
Respostes

a. El proceso padre se bloqueará durante un segundo esperando a su hijo, y pasado ese tiempo acabará aunque su hijo siga en ejecución

b. El proceso hijo nunca acabará la ejecución porque entra en un bucle infinito

c. El proceso padre se bloqueará durante 1 segundo en el waitpid y pasado ese tiempo provocará la finalización del proceso hijo y acabarán los dos

d. El proceso padre se bloqueará durante 1 segundo en la función alarm y a continuación se quedará bloqueado para siempre en el waitpid.

La resposta correcta és: El proceso padre se bloqueará durante 1 segundo en el waitpid y pasado ese tiempo provocará la finalización del proceso hijo y acabarán los dos

SESSIÓ 5

Del programa mem1, tenemos el siguiente trozo de código (modificado). Si el objetivo es reservar 10 zonas de datos de 4096 bytes y guardar las direcciones en un vector p de punteros a chars,

```
#define REGION_SIZE             4096
char *p;

 for (i = 0; i < 10; i++)
  {
    p[i] = malloc (REGION_SIZE);
    sprintf (buff, "\tp: %p, region %d: %p\n", &p, i, p[i]);
    write(1,buff,strlen(buff));
  }
```

¿es correcto el siguiente trozo de codigo?¿Cual de las siguientes afirmaciones es cierta o falsa?

El programa reserva 10 zonas de datos de 4096 bytes pero como p no esta inicializado se liberan automáticamente — Resposta 1: falsa

El programa reserva 10 zonas de datos de 4096 bytes pero no podremo acceder a ellas porque se guardan en un puntero sin inicializar — Resposta 2: cierta

El programa reserva 10 zonas de datos de 4096 bytes pero como p es un puntero sin inicializar es incorrecto — Resposta 3: falsa

La resposta correcta és: El programa reserva 10 zonas de datos de 4096 bytes pero como p no esta inicializado se liberan automáticamente → falsa, El programa reserva 10 zonas de datos de 4096 bytes pero no podremo acceder a ellas porque se guardan en un puntero sin inicializar → falsa, El programa reserva 10 zonas de datos de 4096 bytes pero como p es un puntero sin inicializar es incorrecto → cierta

Del programa mem1, tenemos el siguiente trozo de código

```
#define REGION_SIZE             4096

for (i = 0; i < niterations; i++)
  {
    p = malloc (REGION_SIZE);
    sprintf (buff, "\tp: %p, region %d: %p\n", &p, i, p);
    write(1,buff,strlen(buff));
    free(p);
  }
```

¿Cual de las siguientes afirmaciones es cierta o falsa?

El valor de p será constante ya que en cada iteración pedimos y liberamos la misma cantidad de memoria — Resposta 1: falso

El malloc es incorrecto ya que hay que indicar el tipo de datos de la región de datos — Resposta 2: cierto

El malloc reserva memoria en cada iteración pero el free es incorrecto ya que hemos de indicar el tamaño a liberar

Resposta 3: falso

La resposta correcta és: El valor de p será constante ya que en cada iteración pedimos y liberamos la misma cantidad de memoria → cierto, El malloc es incorrecto ya que hay que indicar el tipo de datos de la región de datos → falso, El malloc reserva memoria en cada iteración pero el free es incorrecto ya que hemos de indicar el tamaño a liberar → falso.

La siguiente línea de código

```
p = malloc(sizeof(int)*100);
```

Trieu-ne una:
Respostes

- a. Reserva en el heap espacio para 100 enteros consecutivos
- b. Siempre incrementa el heap en el tamaño equivalente a 100 enteros
- c. Incrementa el heap en el espacio para 100 enteros pero no es necesario que sean consecutivos ya que cada posicion del vector apuntará a la memoria reservada

La resposta correcta és: Reserva en el heap espacio para 100 enteros consecutivos

Ejecutamos un programa y consultamos su rango de direcciones en el fichero map. El contenido es el siguiente

```
alumne@pcrecanvib5:/proc/5398$ cat maps
562443d3c000-562443d3d000 r-xp 00000000 08:01 282034                     /home/alumne/SO/S5/mem1_previo
562443f3c000-562443f3d000 r--p 00000000 08:01 282034                     /home/alumne/SO/S5/mem1_previo
562443f3d000-562443f3e000 rw-p 00001000 08:01 282034                     /home/alumne/SO/S5/mem1_previo
7effdea5b000-7effdec42000 r-xp 00000000 08:01 393468                     /lib/x86_64-linux-gnu/libc-2.27.so
7effdec42000-7effdee42000 ---p 001e7000 08:01 393468                     /lib/x86_64-linux-gnu/libc-2.27.so
7effdee42000-7effdee46000 r--p 001e7000 08:01 393468                     /lib/x86_64-linux-gnu/libc-2.27.so
7effdee46000-7effdee48000 rw-p 001eb000 08:01 393468                     /lib/x86_64-linux-gnu/libc-2.27.so
7effdee48000-7effdee4c000 rw-p 00000000 00:00 0
7effdee4c000-7effdee73000 r-xp 00000000 08:01 393383                     /lib/x86_64-linux-gnu/ld-2.27.so
7effdf059000-7effdf05b000 rw-p 00000000 00:00 0
7effdf073000-7effdf074000 r--p 00027000 08:01 393383                     /lib/x86_64-linux-gnu/ld-2.27.so
7effdf074000-7effdf075000 rw-p 00028000 08:01 393383                     /lib/x86_64-linux-gnu/ld-2.27.so
7effdf075000-7effdf076000 rw-p 00000000 00:00 0
7ffe8ffd8000-7ffe8fff9000 rw-p 00000000 00:00 0                          [stack]
7ffe8fffa000-7ffe8fffd000 r--p 00000000 00:00 0                          [vvar]
7ffe8fffd000-7ffe8ffff000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

Asocia los rangos de direcciones con regiones del programa

7ffe8ffd8000-7ffe8fff9000 — Resposta 1: stack

562443d3c000-562443d3d000 — Resposta 2: codigo

562443f3d000-562443f3e000 — Resposta 3: data

La resposta correcta és: 7ffe8ffd8000-7ffe8fff9000 → stack, 562443d3c000-562443d3d000 → codigo, 562443f3d000-562443f3e000 → data.

La siguiente línea de código

```
int *p=sbrk(100);
```

Trieu-ne una:
Respostes

- a. Incrementa el límite del heap en el espacio necesario para 100 enteros
- b. Incrementa el límite del heap sólo si es necesario, es decir, si no hay 100 enteros consecutivos libres
- c. Incrementa el límite del heap en 100 bytes
- d. Incrementa el límite del heap sólo si es necesario, es decir, si no hay 100 bytes consecutivos libres

La resposta correcta és: Incrementa el límite del heap en 100 bytes

SESSIÓ 7

El código de los device drivers es independiente de los dispositivos que gestionan

Trieu-ne una:
Respostes

- Vertader
- Fals

La resposta correcta és 'Fals'.

Si utilizamos el siguiente código para escribir en un fichero el número 5:

```
int value = 5;
write(1, &value, sizeof(int));
```

Y el siguiente código para leer el fichero generado

```
char buffer[80]
int ret = read (0, buffer, sizeof(buffer));
```

Cuál o cuáles de las siguientes afirmaciones es cierta

Trieu-ne una o més:
Respostes

- [ ] Después del read, la variable buffer contendrá los bytes que codifican el entero guardado en el fichero
- [x] El read devolverá el número de bytes leídos (sizeof(buffer))
- [ ] Después del read, la variable buffer contendrá el entero leído convertido a string
- [ ] El read devolverá el número de bytes leídos (sizeof(int))

Retroacció

Les respostes correctes són: El read devolverá el número de bytes leídos (sizeof(int)), Después del read, la variable buffer contendrá los bytes que codifican el entero guardado en el fichero

Al ejecutar el siguiente comando desde la shell, ¿cuánto vale argc de cmd1?

```
# cmd1 -l a.out < /etc/issue > /tmp/log.cmd1
```

Resposta: 1

La resposta correcta és: 3

Dado el siguiente código que lee una secuencia de bytes de la entrada estándar utilizando un buffer.

```
int ret;
char buff[256];
ret=read(0, buff, sizeof(buff));
```

¿Cuál de las siguientes llamadas a write sería la más adecuada para escribir los caracteres leídos por la salida estándar?

Trieu-ne una:
Respostes

- write(1, buff, sizeof(buff));
- write(1, buff, 256);
- write(1, buff, strlen(buff));
- write(1, buff, ret);

La resposta correcta és: write(1, buff, ret);

Si suponemos que hemos creado un nuevo dispositivo llamado 'myDevice' utilizando el comando 'mknod', indica cuál de las siguientes opciones permitiría redireccionar la entrada estándar del programa 'cat' a este nuevo dispositivo

Trieu-ne una:
Respostes

cat | myDevice

cat < myDevice

cat > myDevice

No es posible hacerlo, ya que myDevice no es un fichero regular

La resposta correcta és: cat < myDevice

SESSIÓ 8

El siguiente trozo de código:

int v[2];

pipe (v);

Crea una pipe sin nombre, donde el parámetro v corresponde al buffer donde se almacenarán los datos leídos o escritos.

Trieu-ne una:
Respostes

Vertader

Fals

La resposta correcta és 'Fals'.

Un socket y una pipe se diferencian en:

Trieu-ne una o més:
Respostes

a. El socket utiliza dos canales para comunicar procesos, mientras que la pipe utiliza un único canal de lectura y escritura.

b. Para poder utilizar sockets el programa debe tener permisos de root.

c. Un socket permite comunicar procesos en máquinas diferentes pero conectadas a través de una red.

d. Una pipe utiliza dos canales para comunicar procesos, mientras que un socket utiliza un único canal de lectura y escritura.

Les respostes correctes són: Un socket permite comunicar procesos en máquinas diferentes pero conectadas a través de una red., Una pipe utiliza dos canales para comunicar procesos, mientras que un socket utiliza un único canal de lectura y escritura.

Dado el siguiente trozo de código:

```
char buf[SIZE];
while ((n = read(0, buf, SIZE)) > 0)
      write(1, buf, n);
```

Indica cuáles de las siguientes afirmaciones son **CIERTAS**:

Trieu-ne una:
Respostes

a. El proceso que ejecuta el código finalizará el bucle cuando lea el carácter ^D

b. El proceso que ejecuta el código queda bloqueado en la operación read hasta que no logre leer SIZE bytes.

c. El proceso que ejecuta el código quedará bloqueado en la operación write hasta que no logre escribir SIZE bytes

d. El proceso escribirá n bytes por salida estándar y continuará la ejecución del bucle mientras n>0

La resposta correcta és: El proceso que ejecuta el código finalizará el bucle cuando lea el carácter ^D

Dado el siguiente código:

```
main() {
      char buf[64];
      int pd[2];
      pipe(pd);
      if (fork()==0) {
            dup2(pd[0], 0);
            close(pd[0]);
            close(pd[1]);
            execlp("cat","cat",NULL);
      } else {
            close(pd[0]);
            sprintf(buf,"Un saludo\n");
            write(pd[1],buf,strlen(buf));
            close(pd[1]);
            sprintf(buf, "Fin del saludo\n");
            write(1, buf, strlen(buf));
      }
}
```

Suponiendo que lo ejecutamos sin redireccionar ningún canal, cual de las siguientes afirmaciones es **CIERTA**:

Trieu-ne una:
Respostes

- a. Por la salida estándar mostrará únicamente el mensaje "Un saludo"
- b. Por la salida estándar mostrará únicamente el mensaje "Fin del saludo"
- c. Por la salida estándar mostrará los mensajes "Un saludo" y "Fin del saludo"
- d. Por la salida estándar no mostrará ningún mensaje

La resposta correcta és: Por la salida estándar mostrará únicamente el mensaje "Un saludo"

Para el caso de una operación de escritura sobre una pipe sin lectores, indica cuál de las afirmaciones es **CIERTA**:

Trieu-ne una:
Respostes

- a. El programa no compila.
- b. Cuando el proceso solicite la operación de escritura sobre una pipe sin lectores, éste recibirá una señal SIGPIPE.
- c. El proceso que solicita la operación de escritura quedará bloqueado a la espera de lectores.
- d. La operación de escritura retornará un 0 indicando que ha podido escribir 0 bytes.

La resposta correcta és: Cuando el proceso solicite la operación de escritura sobre una pipe sin lectores, éste recibirá una señal SIGPIPE.

SESSIÓ 9

Queremos posicionar el puntero de lectura/escritura de un canal (file descriptor) asociado a un fichero ordinario a la antepenútima posición del fichero. ¿Qué forma de especificar el desplazamiento es la más idónena?

Trieu-ne una:
Respostes

- a. SEEK_CUR
- b. SEEK_SET
- c. SEEK_END
- d. Ninguna de las anteriores

La resposta correcta és: SEEK_END

¿En qué caso(s) la llamada al sistema lseek devolverá error?

Trieu-ne una o més:
Respostes

- a. En un canal (file descriptor) asociado a un fichero ordinario sobre el que sólo tenemos permiso de lectura, al intentar modificar el puntero de lectura/escritura.
- b. En un canal (file descriptor) asociado a un socket, al intentar modificar el puntero de lectura/escritura.

c. En un canal (file descriptor) asociado a un fichero ordinario que NO ha sido abierto en modo O_RDWR, al intentar modificar el puntero de lectura/escritura.

La resposta correcta és: En un canal (file descriptor) asociado a un socket, al intentar modificar el puntero de lectura/escritura.

¿Cuál(es) de las siguientes situación(es) provocará(n) que la llamada al sistema "open" devuelva error?

Trieu-ne una o més:
Respostes

a. Intentar abrir un fichero que ya está abierto por el mismo proceso.

b. Intentar abrir un fichero con el flag O_TRUNC cuando dicho fichero ya está abierto por otro proceso del sistema.

c. Intentar abrir un fichero en modo de escritura cuando el usuario tiene toda su quota de disco ocupada.

d. Intentar abrir un fichero cuando el proceso tiene la tabla de canales llena.

La resposta correcta és: Intentar abrir un fichero cuando el proceso tiene la tabla de canales llena.

Indicad cuál(es) de las siguientes afirmaciones en relación a enlaces simbólicos (soft links) y enlaces físicos (hard links) son CIERTAS.

Trieu-ne una o més:
Respostes

a. Con la orden "ln" sólo podemos crear enlaces simbólicos.

b. No es posible parametrizar la llamada "open" para que cree un enlace simbólico.

c. Al borrar un fichero mediante la orden "rm", el sistema de ficheros NO actualiza los enlaces simbólicos que lo referenciaban.

d. La orden "cp src.txt dst.txt" (donde src.txt es un enlace simbólico correcto y dst.txt no existe) no podrá ejecutarse y mostrará un mensaje de error.

Les respostes correctes són: No es posible parametrizar la llamada "open" para que cree un enlace simbólico., Al borrar un fichero mediante la orden "rm", el sistema de ficheros NO actualiza los enlaces simbólicos que lo referenciaban.

Para ejecutar la órden "mkdir dir"? (podéis asumir que "dir" no existe), ¿en cuántas estructuras i-node es preciso actualizar el campo "número de enlaces" (link count)?

Trieu-ne una:
Respostes

a. 0

b. 1

c. 2

d. 3

La resposta correcta és: 2

PARCIAL TEORIA (T1-T3)

Es posible que un proceso después de llamar a exit se quede en estado zombie para siempre?

Trieu-ne una:

a. NO, el Sistema siempre libera los PCBS de los procesos zombies si su padre no 10 hace
b. Sí, si el padre se queda vivo para siempre y no libera los PCBS de sus hijos muertos
c. No, eso sólo seria posible si el proceso hubiera muerto por signal
d, No, el estado zombie es algo temporal mientras no se recoge la causa de su muerte

La resposta correcta és: Si, si el padre se queda vivo para siempre y no libera los PCBs de sus hijos muertos siempre y no libera los PCBs de sus hijos muertos

Dado el siguiente código, y suponiendo que ninguna llamada a sistema devuelve error, indica cuál de las siguientes opciones representa su comportamiento.

```
main() {
        char msg[80];
        fork();
        execlp("ls","ls",(char *))0);
        sprintf(msg,"Saliendo\n");
        write(1,msg,strlen(msg));
}
```

Trieu-ne una:
a. Veremos dos veces el contenido del directorio actual de trabajo y 2 veces el mensaje "Saliendo \n"
b. Veremos una vez el contenido del directorio actual de trabajo y una vez el mensaje "Saliendo \n"
c. No veremos nada
d. Veremos dos veces el contenido del directorio actual de trabajo

La resposta correcta és: Veremos dos veces el contenido del directorio actual de trabajo

Un sistema operativo realiza diferentes funciones. Elige la que NO procede,
Trieu-ne una:

a. Hace de intermediario entre el usuario y el hardware
b. Programa las operaciones más difíciles.
c. Es el programa que asigna recursos a los demás
d. Ofrece un entorno seguro e eficiente

La resposta correcta és: Programa las operaciones más difíciles.

Dado el siguiente código:

```
main() {
        sigset_t mask;
        sigfillset(&mask) ;
        sigprocmask(SIG_BLOCK, &mask, NULL);
        alarm(1);
        sigsuspend(&mask);
        sprintf(msg, "final\n");
        write(1,msg,strlen(msg));
}
```

Selecciona cuál será su comportamiento si 10 ponemos en ejecuci6n y sabemos que ninguna llamada a sistema devolverá error.
Trieu-ne una:

a. Se quedará para siempre bloqueado en el sigsuspend, para evitarlo deberíamos modificar el parámetro del sigsuspend
b. Se quedará para siempre bloqueado en el sigsuspend, para evitarlo deberíamos quitar la llamada a sigprocmask
c. Acabará al llegar al final del código mostrando por pantalla el mensaje "final\n"
d. Morirá al recibir el SIGALRM que programa alarm(l) y no mostrará el mensaje "final\n" por pantalla

La reposta Correcta és: Se quedará para Siempre bloqueado en el Sigsuspend, para evitarlo deberíamos modificar el parámetro del sigsuspendd.

Dado un sistema linux ¿cuál de las siguientes características Sí son heredadas por los hijos?
Trieu-ne una:

a. El valor del program counter
b. Los temporizadores activos
c. El identificador de proceso padre
d. Los signals pendientes de tratar

La resposta correcta es: El valor del program counter.

Considera la creación de procesos en Linux Cuál de las siguientes características de su padre NO HEREDARÁ el proceso hijo:
Trieu-ne una:

a. Las variables locales
b. El identificador
c. El identificador del usuario propietario
d. Las variables globales e. Los tratamientos asociados a los signals

La resposta correcta es: El identificador.

Dado el siguiente código:

```
void trat_alrm(int s){}
main(){
        sigset_t mask;
        struct sigaction sa;
        sigfillset(&mask);
        sigprocmask(SIG_BLOCK, &mask, NULL);
        sa.sa_handler.trat_alrm;
        sa.sa_flags=0;
        sa.sa_mask=mask;
        sigaction(SIGALRM,&sa,NULL);
        alarm(1);
        sigsuspend(&mask);
        sprintf(msg,"final\n");
        write(1, msg, strlen(msg));
}
```

Selecciona cuál será su comportamiento si lo ponemos en ejecución y sabemos que ninguna llamada a sistema devolverá error.
Trieu-ne una:

a. Morirá al recibir el SIGALRM que programa alarm(1) y no mostrará el mensaje "final" por pantalla
b. Se quedará para siempre bloqueado en el sigsuspend, para evitarlo deberíamos quitar la llamada a sigprocmask
c. Se quedará para siempre bloqueado en el sigsuspend, para evitarlo deberíamos modificar el parámetro del sigsuspend
d. Acabará al llegar al final del código mostrando por pantalla el mensaje "final"

La resposta correcta és: Se quedará para siempre bloqueado en el sigsuspend, para evitarlo deberíamos modificar el parámetro del sigsuspendd. Ofrece un entorno seguro e eficiente

```
main(){
        execlp("./prog","prog",(char*)0);
}
```

Suponiendo que lo ponemos en ejecución desde el directorio donde se encuentra y que tenemos permiso de ejecución sobre el ejecutable. Indica cuál de las siguientes afirmaciones es cierta.
Trieu-ne una:

a. Fallará al ejecutar execlp porque un proceso no puede mutar al mismo programa que está ejecutando
b. El proceso no acabará nunca, se quedará para siempre en estado RUN, ejecutando mutaciones al mismo programa
c. Fallará cuando el sistema se quede sin memoria para crear PCB's nuevos necesarios para las mutaciones
d. El proceso no acabará nunca, irá cambiando de estado y liberando la CPU cuando lo decida el planificador, y ejecutará para siempre mutaciones al mismo programa

La resposta correcta és: El proceso no acabará nunca, irá cambiando de estado y liberando la CPU cuando lo decida el planificador, y ejecutará para siempre mutaciones al mismo programa

¿La llamada a sistema fork es bloqueante?
Trieu-ne una:

a. No, una vez creado el nuevo proceso ambos pasan a competir por la CPU y ambos se encuentran en disposición de ocuparla
b. Si, los dos procesos, padre e hijo, pasan al estado BLOCKED a la espera de que el sistema les asigne la CPU
c. Si, al crear al nuevo proceso el padre pasa al estado BLOCKED hasta que el hijo le cede la CPU
d. Si, el nuevo proceso creado...
La resposta correcta és: No, una vez creado el nuevo proceso ambos pasan a competir por la CPU y ambos se encuentran en disposición de ocuparla

Considera la política de planificación Round Robin. Dado un proceso intensivo en cálculo y otro proceso intensivo en E/S, ¿cuál de los dos tendrá más sobrecarga en su tiempo de ejecución debido a los cambios de contexto?
Trieu-ne una:

a. El intensivo en cálculo, porque será expulsado más veces de la CPU por el sistema para evitar que acapare la CPU
b. El intensivo en E/S, porque se bloqueará con mucha frecuencia y cada bloqueo implica un cambio de contexto J
c. La sobrecarga será la misma ya que los dos tipos de procesos son expulsados de la CPU con la misma frecuencia
La resposta correcta és: El intensivo en E/S, porque se bloqueará con mucha frecuencia y cada bloqueo implica un cambio de contexto.

Indica qué requerimientos desde el punto de vista del programador deben tener las Mamadas a sistema. (Puedes seleccionar más de una).
Trieu-ne una o més:

a. Deben poder evolucionar en el tiempo, añadiendo parámetros o cambiando el tipo de los mismos.
b. No se puede modificar su contexto, deben guardar y restaurar los registros de CPU que vayan a modificar.
c. Deben poder invocarse como cualquier otra función
d. Deben ser muy eficientes, ejecutándose en modo usuario para ir más deprisa.
Les respostes correctes són: Deben poder invocarse como cualquier otra función, No se puede modificar su contexto, deben guardar y restaurar los registros de CPU que vayan a modificar.

En un sistema multiprogramado, ¿cuántos procesos pueden estar simultáneamente en estado RUN?
Trieu-ne una:

a. Sólo 1
b. Como mínimo tantos como CPU's o cores tenga la máquina
c. Como máximo tantos como CPU's o cores tenga la máquina
La resposta correcta és: Como máximo tantos como CPU's o cores tenga la maquina

Empareja las siguientes llamadas a sistema sobre el dedo que pueden tener sobre las alarmas pendientes de un proceso
La resposta correcta és: exec — No tiene ningún efecto sobre las alarmas pendientes, fork — El nuevo proceso inicia la ejecución sin ninn,lno alarma nandiantp aluno, IMP anillar y pstololpcpr loa nieva alarmad. Las variables globales e. Los tratamientos asociados a los signals

Considera la creación de procesos en linux. Y un código que crea un proceso hijo. Antes de la creación, se declaran variables locales y globales que ambos procesos modifica, ¿Qué valor tendrán esas variables al final de la ejecución de cada proceso?
Trieu-ne una:

a. El valor que deje el último proceso que la modifique
b. El valor que deje el padre en la inicialización
c. Depende del orden en el que se intercale la ejecución de ambos procesos
d. Para cada proceso tendrán el valor que ese proceso haya puesto
e. Depende de si la variable es local o global
La resposta correcta és: Para cada proceso tendrán el valor que ese proceso haya

¿Cuándo se ejecuta código del kernel del sistema operativo?
Trieu-ne una:

a. Solo cuando una aplicación invoca una llamada a sistema
b. Cuando el sistema recibe una interrupción, cuando una aplicación provoca una excepción y cuando una aplicación invoca una llamada a sistema
c. Cuando estamos en modo usuario
d. ninguna de las otras opciones es correcta.

La resposta correcta és: Cuando el sistema recibe una interrupción, cuando una aplicación provoca una excepción y cuando una aplicación invoca una llamada a sistema

Suponed que tenemos tres programas que han capturado el signal SIGINT y le han asignado una rutina de usuario que simplemente hace una escritura en pantalla y permite que el programa continúe. Sin hacer ninguna otra llamada a sistema a continuación, cada programa ejecuta una de las siguientes sentencias
programa 1: kill(getpid(I, SIGINT);
programa 2: kill(getpid(I, SIGSTOP);
programa 3: kill(getpid(I, SIGKILL)
¿En qué estado estará el proceso que ejecute cada uno de estos programas como consecuencia de ejecutar estas sentencias? Empareja cada sentencia con el estado.

La resposta correcta és: kilfigetpidOSIGINT) — RUN, kiNgetpidg,SIGKILD ZOMBIE, kill(getpidg,SIGSTOP) BLOCKED.

Un programa es portable entre dos sistemas si ...
Trieu-ne una:

a. el sistema operativo es el mismo, aunque la arquitectura de la cpu sea diferente.
b. el sistema operativo y la arquitectura de la cpu son las mismas
c. solo si la cpu es la misma, independientemente del sistema operativo
d. si el binario está en formato ELF (Extensible Linking Format)

La resposta correcta és: el sistema operativo y la arquitectura de la cpu son las mismas.

Una vez acabada la system cal!, se vuelve a modo usuario mediante una instrucción. En Intel, hay más de una que cumple esa función. Elige la que NO es correcta.
Trieu-ne una:

a. sysexit
b.rett
c. ret
d. iret

La resposta correcta és: ret

Con una política de planificación apropiativa, ¿podría darse que un proceso nunca pase del estado RUN a READY?
Trieu-ne una:

a. SI, podría ocurrir que un proceso siempre se bloqueara o que acabara la ejecución antes de que el sistema decidiera expulsado
b. No, esto solo puede ocurrir si la política de planificación es no apropiativa
c. No, si la política de planificación es apropiativa el sistema puede expulsar a los procesos de RUN aunque todavía puedan avanzar con la ejecución

La resposta correcta és: SI, podría ocurrir que un proceso siempre se bloqueara o que acabara la ejecución antes de que el sistema decidiera