SISTEMAS OPERATIVOS II
Ingeniería Informática
Junio 2009

Tiempo 1h 30 minutos
Justificar debidamente TODAS las respuestas.

1	2	3	4	5	6	T
1	1.5	1	1.5	1.5	1.5	8

1. A continuación se muestra una función que establece una pila alternativa para la ejecución de manejadores de señales y tres maneras de llamar a dicha función.

```
int SetPila(int tam, void *addr)
{
  sigset_t s;

  s.ss_sp=addr;
  s.ss_size=tam;
  s.ss_flags=0;

  return (sigaltstack(&s,NULL));
}
```

- Primera manera

```
char pila [MAXPILA];
...
...
main(int argc,char*argv[])
{
......
......
if (SetPila(MAXPILA, (void *) pila)==-1)
 .....
```

- Segunda manera

```
main(int argc,char*argv[])
{char pila [MAXPILA];
......
......
if (SetPila(MAXPILA, (void *) pila)==-1)
 .....
```

- Tercera manera

```
.....
int CrearPila()
{
 char pila [MAXPILA];
 return SetPila(MAXPILA,(void *)pila);
}
...
main(int argc,char*argv[])
{
  ....
  if (CrearPila()==-1)
    .......
```

Supuestas las siguientes condiciones:

- MAXPILA está correctamente declarado
- MAXPILA está dentro de los límites que impone el sistema operativo
- MAXPILA es lo suficientemente grande para que funcione sin problemas el manejador que se quiere ejecutar ahí.

¿Son correctas estas maneras de llamar a *sigaltstack*?. Razónese.

2. Del proceso con *pid* 15095 se muestra la salida obtenida con el comando *pmap* al comenzar su ejecución y despues de que dicho proceso haya hecho algunas cosas.

```
user@abyecto:~$ pmap  15096
15096:   ./a.out
08048000    476K r-x--  /home/user/a.out
080bf000      4K rw---  /home/user/a.out
080c0000   2376K rw---    [ anon ]
b7f21000      8K rw---    [ anon ]
b7f23000      4K r-x--    [ anon ]
bf9fe000     84K rw---    [ stack ]
 total     2952K
```

```
user@abyecto:~$ pmap  15095
15095:   ./a.out
08048000    476K r-x--  /home/user/a.out A
080bf000      4K rw---  /home/user/a.out B
080c0000   2376K rw---    [ anon ] C
150af000   3368K r----  /home/user/etc.tar.gz D
153f9000   3368K r----  /home/user/etc.tar.gz E
15743000 195448K rw---    [ anon ]  F
2172d000 1435308K r----  /home/user/fichero.iso G
790d8000   3368K r----  /home/user/etc.tar.gz H
79422000   1664K r----  /home/user/bin.tar  I
795c2000   9768K rw-s-    [ shmid=0x148014 ] J
79f4c000   9768K rw-s-    [ shmid=0x148014 ] K
7a8d6000   9768K rw-s-    [ shmid=0x14000f ] L
7b260000   9768K rw-s-    [ shmid=0x148014 ] M
7bbea000   9768K rw-s-    [ shmid=0x14000f ] N
7c574000 976572K rw---    [ anon ] O
b7f23000      4K r-x--    [ anon ]  P
bf9fe000     84K rw---    [ stack ] Q
 total  2468696K
```

Contéstese a cada una de las siguientes preguntas indicando SI, NO o NO PUEDE SABERSE A LA VISTA DE LA INFORMACIÓN QUE SE SUMINISTRA con *pmap*

a) ¿Ha hecho el proceso alguna llamada *shmget*? (si la respuesta es SI indíquese el número mínimo de llamadas que ha podido hacer y si es NO o NO PUEDE SABERSE dígase por qué)

b) ¿Ha hecho el proceso alguna llamada *shmat*? (si la respuesta es SI indíquese el número mínimo de llamadas que ha podido hacer y si es NO o NO PUEDE SABERSE dígase por qué)

c) ¿Ha hecho el proceso alguna llamada *mmap*? (si la respuesta es SI indíquese el número mínimo de llamadas que ha podido hacer y si es NO o NO PUEDE SABERSE dígase por qué)

d) ¿Ha hecho el proceso alguna llamada *open*? (si la respuesta es SI indíquese el número mínimo de llamadas que ha podido hacer y si es NO o NO PUEDE SABERSE dígase por qué)

e) ¿Ha hecho el proceso alguna llamada *malloc*? (si la respuesta es SI indíquese el número mínimo de llamadas que ha podido hacer y si es NO o NO PUEDE SABERSE dígase por qué)

f) ¿Hay, en el espacio de direcciones de este proceso, algún *segmento vnode*?

3. En el buffer cache la FREE LIST (lista de bloques libres) y la marca de libre/ocupado que lleva cada buffer parecen contener la misma información. ¿Cuál o cuáles de las siguientes afirmaciones son correctas?

a) Es la misma información, se guarda en cada buffer además de en la lista por seguridad: si la lista se corrompe puede regenerarse a partir de los buffers marcados como libres.

b) No es exactamente la misma información. Todo buffer en la FREE LIST está libre, pero no todo buffer con marca de libre está en la FREE LIST

c) No es exactamente la misma informacón puesto que si dos buffers contienen un mismo bloque de disco y ninguno de ellos está siendo utilizado, en la FREE LIST solo aparecería una vez

d) No es exactamente la misma información. Todo buffer en en la FREE LIST debe tener la marca de libre y **ADEMÁS** no tener la de *delayed write*

e) Ninguna de las enteriores es correcta (explíquese)

a) ¿Qué es una *sleep priority*?

b) ¿Qué es un punto de apropiación?

c) En un sistema System V R4 ¿Cómo varía la prioridad de un proceso de la clase *time sharing*? ¿y la de un proceso de la clase *real time*?

5. La llamada *sighold* es la llamada que existe en System V R3 para que un proceso enmascare una señal. Aqui se presenta una implementación utilizando las llamadas nativas de System V R4

```
int sighold(int sig)
{
  struct sigaction s;

  if (sigaction(sig, NULL,&s)==-1) /**/
    return -1;
  sigaddset (&s.sa_mask,sig);
  return (sigaction(sig,&s,NULL);
}
```

Elegir **CLARAMENTE** entre a) ó b)

a) Es correcta. Dígase entonces si es necesaria o superflua la primera llamada a sigaction (/**/) y por qué

b) Es incorrecta. Propóngase la solucion correcta.

6. a) En un shell ¿Cual es la diferencia (desde el punto de vista de programación) entre crear un proceso en primer plano y otro en segundo plano?

b) ¿Podría hacerse que un proceso que está siendo ejecutado por un shell en primer plano pasase a ser ejecutado por ese mismo shell en segundo plano? (Se refiere al mismo proceso, no a crear uno nuevo que ejecuta el mismo programa).

1 - La tercera manera es incorrecta: Se utiliza como pila la dirección de una variable local de la función (crearPila); cuando se hace return de esta función dicha zona de memoria (que está en la pila) se desasigna y queda libre para ser usada por llamadas a otras funciones (parámetros, variables locales, direcciones de vuelta) ~~poder~~ por lo que la ejecución del manejador podría sobreescribir estas cosas produciendo resultados inesperados.

La segunda manera es correcta, aunque en este caso se trata también de una variable local, esta ~~de~~ solo se desasignaría al hacer return en main() y esto es al terminar el proceso.

La primera manera es correcta: se trata de una variable global que existe siempre.

2- ~~Con~~ En el enunciado se han marcado las regiones del proceso A .. Q para mayor claridad.

a) SI. En el espacio de direcciones del proceso hay 5 regiones de memoria compartida (J, K, L, M, N) que corresponden a dos

zonas distintas: una, de identificador 0x148014, que se ha mapeado 3 veces (J, K, M) y otra, de identificador 0x1400f, que se ha mapeado dos veces (L, N). El número mínimo de llamadas para obtener dos identificadores distintos es 2.

b) SI. Hay 5 mapeos de zonas de memoria compartida por tanto 5 llamadas shmat.

c) SI. Las regiones D, E, G, H, I corresponden a ficheros mapeados en memoria, por tanto 5 llamadas mmap (un fichero mapeado varias veces son mapeos distintos y por tanto varias llamadas mmap)

d) NO PUEDE SABERSE

e) SI. Si nos fijamos en los segmentos anónimos vemos que F y O no estaban al principio de la ejecución, por lo tanto como mínimo ha hecho dos llamadas malloc.

f) SI: A, B, D, E, G, H, I. Un segmento anodo es un segmento asociado a un fichero

3- e) Es la misma información. En cada buffer hace falta para saber si está libre o ocupado sin necesidad de recorrer una lista. La FREELIST

hace falta para mantener el orden LRU en el reemplazo

4- a) sleep priority: prioridad que se asigna a los procesos que vuelven de una espera. Es mayor que las prioridades en modo usuario para que las llamadas al sistema se completen antes

b) Un punto de apropiación es un punto dentro del código del Kernel donde el kernel esta en estado consistente y se comprueba si hay algun proceso RT listo. En caso de haberlo se inicia el cambio de contexto.

c) La de los procesos RT es fija y solo se cambia mediante llamada al sistema

La de un proceso TS se recalcula al abandonar la CPU. si agota el quanto su prioridad disminuye. si abandona la CPU antes de agotar el quanto su prioridad aumenta.

5. Es incorrecta: la mascara que se instala con sigaction solo afecta al manejador, no al proceso

```
int sighold (int sig)
{ sigset_t s;
  sigemptyset (&s);
  sigaddset (&s, sig);
  return sigprocmask (SIG_BLOCK, &s, NULL);
}
```

6- a) En el proceso en primer plano el shell se queda en espera con alguna de las llamadas wait, mientras que en segundo plano el shell continua su ejecución.

```
if ((pid = fork()) == 0) { /* hijo */
    Ejecutar(---);
    exit(0);
}
if (primerplano)
    waitpid(pid, NULL, 0);
```

b) Habría que sacar al shell de la espera lo cual puede hacerse enviandole una señal. Evidentemente debe ser una señal que no haga que el shell termine o que tenga instalado un manejador con el flag SA_RESTART que haría que el shell reiniciase la llamada wait