SISTEMAS OPERATIVOS II
Ingeniería Técnica Informática Sistemas
Febrero 2009

Tiempo 1h 30 minutos
Poner Nombre y Apellidos en todas la hojas.
Grapar todas las hojas, incluida esta.
Justificar debidamente TODAS las respuestas.

1	2	3	4	5	6	T
1	1.25	1.5	1.5	1.5	1.25	8

1. ¿Que hace el siguiente trozo de código introducido en un programa al principio de *main()*?(se supone que df es un entero y argv[] en array de punteros que recibe *main* como segundo parámetro)

```
if (argv[1]!=NULL){
  df=open(argv[1],O_RDWR|O_CREAT|O_EXCL);
  if (df==-1){
      perror ("errorcillo");
      exit(0);
    }
  close (1);
  dup (df);
  close (df);
}
```

2. Un alumno se S.O. II tiene que entregar una práctica que tiene un error de memoria y produce un fallo de segmentación. El profesor lo ve y le recuerda que si la práctica termina con *segmentation fault* no puntua. Como el plazo acaba esa tarde y no quiere usar el depurador, tras hablar con unos amigos que hicieron la asignatura otro año, se le ocurre lo siguiente: pone al principio de *main()* una llamada a una función llamada *Inicializa()*. El código de *Inicializa()* puede verse a continuación.

```
void man (int sig)
{
  printf ("Problemas  solucionados\n");
}

void Inicializa(){
{
  struct sigaction s;

  sigemtyset(&s.sa_mask);
  s.sa_handler=man;
  s.sa_flags=0; /*a ver si funciona*/
  sigaction (SIGSEGV,&s,NULL);
}
```

Suponiendo que no se trata de la práctica de señales (y por tanto no hay riesgo de desinstalar el manejador durante la corrección), y suponiendo que el profesor no va a ver el código de *Inicializa()*. ¿Conseguirá el alumno que el profesor no se percate de que su practica produce un fallo de segmentación?

a Si, por supuesto

b Imposible

c Si, pero tiene que cambiar s.sa_flags. (la línea con el comentario de /*a ver si funciona*/)(¿qué hay que cambiar?)

d Si, pero tiene que cambiar otra cosa en *Inicializa()* (¿Cuál?)

3.
- En un sistema UNIX, dígase en qué tres zonas considera el kernel dividida la memoria física
- Diígase en cual (o cuales) de las zonas antes citadas estarían los siguientes elementos.
  - variables locales de un proceso
  - variables globales de un proceso
  - memoria asignada con *malloc()*
  - *u_area* de un proceso
  - estructura *proc* de un proceso
  - estructura *anon_map* de un segmento de un proceso
  - *buffers* de disco
  - tabla de ficheros abriertos del sistema
  - tabla de de descriptores de fichero de usuario
  - código de un manejador de interrupción
  - código del manejador de SIGUSR1 para un proceso
  - código del manejador de SIGSEGV para un proceso
  - máscara de señales ignoradas de un proceso
  - lista de estructuras *seg* que describen el espacio de direcciones de un proceso

4. Un shell mantiene una lista de procesos en segundo plano lanzados desde él e implementada como una lista de estructuras como la que se describe a continuación:

```
struct PROCESO {
   pid_t pid;
   char doscripcion[MAXLINEA];
   int estado;
   int valor_devuelto;
   int prioridad;
   };
typedef struct PROCESO proc_t;
```

Con la siguiente cabecera

```
void ConviertePrimerPlano (proc_t *p)
```

¿Puede hacerse una función que convierta un proceso en segundo plano de dicha lista en primer plano?. En caso afirmativo hágase y en caso negativo justifíquese por qué.

5.
```
void * Smalloc (size_t tam)
{
 int id;
 void * p;
 key_t k=(key_t) getuid();

 if ((id=shmget(k,tam,0_CREAT|0600))==-1)
   return (NULL);

 if ((p=shmat(id,NULL,0))==(void*) -1)
   return (NULL);

 return (p);
}
```

a ¿Qué hace la función *Smalloc()*?

b ¿Qué ocurre si varios procesos del mismo usuario llaman a esta función? ¿y si lo hacen varios procesos de distintos usuarios (todos ellos distintos del root)?

6. En un directorio con permisos rwxrwxrwx se encuentran un fichero *f1* y un ejecutable, *a.out*, cuyo código se muestra a continuación.

```
void main()
{
  int df;

  if ((df=open("f1",0_RDWR))==-1)
     perror ("fallo en open");
  else
      close(df);
  df=open("f2",0_CREAT| 0_RDWR,4755);
  close(df);
}
```

Tanto el ejecutable *a.out* como el fichero *f1* son propiedad del usuario *u1*. Rellénese el siguiente cuadro indicando si se producirá el error de *"fallo en open"* y de quien será el fichero *f2* creado si un usuario *u2* ejecuta *a.out* desde el citado directorio. Se supone que *u1* y *u2* no pertenecen al mismo grupo y ninguno de ellos es el *root*

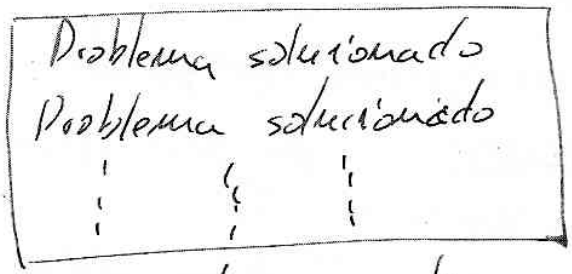
a.out	f1	err open	prop f2
rwxr-xr-x	rwxr-xr-x	SI	u2
rwsr-xr-x	rwxr-xr-x	NO	u1
rwxr-xr-x	rwsr-xr-x	SI	u2
rwsr-xr-x	rwsr-xr-x	NO	u1
rwx———	rwx———	—	—
rws———	rwx———	—	—
rwx———	rws———	—	—
rws———	rws———	—	—
rwxr-xr-x	rwx———	SI	u2
rwsr-xr-x	rwx———	NO	u1
rwxr-xr-x	rws———	SI	u2
rwsr-xr-x	rws———	NO	u1
rwx———	rwxr-xr-x	—	—
rws———	rwxr-xr-x	—	—
rwx———	rwsr-xr-x	—	—
rws———	rwsr-xr-x	—	—

1- En el caso de que reciba un parámetro, considera que es el nombre de un fichero al que redirecciona la salida estándar.

Si el fichero ya existe o no puede abrirlo por cualquier otro motivo (falta de permisos en el directorio ...) el programa da un mensaje de error y termina.

2- b/ Imposible

El manejador NO ARREGLA el fallo de segmentación. Al producirse el error de memoria se produce una excepción ⇒ control al S.O. que envía al proceso SIGSEGV. El proceso al recibir SIGSEGV ejecuta el manejador y luego vuelve a donde estaba: reintenta la instrucción y se vuelve a producir el error ⇒ bucle infinito de llamadas al manejador. El profesor ve



y se da cuenta

Si se cambia la línea s.sa_flags = 0 y se

ponemos unos flags que contengan SA_RESETHAND
en pantalla sale:

Problema solucionado Segmentation fault: core dumped

pues tras la primera llamada al manejador vuelve a su acción por defecto. El profesor también se da cuenta en este caso

3) a) Código del Kernel
Datos de Kernel
Páginas de procesos de usuario.

b) variables locales → páginas procesos usuario
" globales → " " "
memoria asignada malloc → " " "
→ " " "
marca → datos del kernel
estructura proc → " " "
estructura anon_vma → " " "
buffers de disco → " " "
tabla ficheros abiertos del sistema → " " "
tabla descriptores fichero usuario → páginas procesos
código manejador interrupción → código del kernel
código manejador SIGUSR1 → páginas procesos
" " SIGSEGV → " "
mascara señales ignoradas → datos del kernel
lista estructuras seg → " "

El espacio de usuario de un proceso, es decir, todo lo que es código, datos o pila de un proceso está en las páginas de los procesos de usuario. Las variables locales son pila, las globales y lo asignado con malloc son datos y el código de los manejadores es código del proceso. La macro también es espacio de usuario así como la tabla de descriptores del proceso, que está en la macro.

4- La diferencia entre la ejecución en primer y segundo plano es que en primer plano el padre espera a que el hijo termine. Por tanto.

```
void ConvertirPrimerPlano (proct *p)
{
    waitpid(p->pid, NULL, 0);
    QuitarProcesoLista SP (p);
}
```

5- Asigna un bloque de memoria compartida de tamaño tam, creándola en el caso de que no exista. Nos devuelve el puntero a la zona de memoria.

Utiliza como clave el número de ~~usuario~~ usuario, de ahí que si ejecutan esta función

varios procesos del mismo usuario compartirán la zona de memoria entre ellos, y si son de distinto usuario no, ya que se crearía una zona de memoria compartida para cada usuario

6- Dado que u1 y u2 son de distinto grupo, para que u2 pueda ejecutar a.out debe tener permisos de ejecución para resto, es decir rwx r-x r-x ó rwx r-x r-x. Donde a.out no tenga estos permisos u2 no podrá ejecutarlo y no habrá error en open y no se creará f2 (en el cuadro se indica con —).

Dado que solo hay permiso de escritura de f1 para el propietario, y la apertura es en modo RW, para que la apertura no de error el proceso que intenta abrir f1 debe tener credencial efectiva de u1, lo cual solo ocurre cuando los permisos de a.out son rws r-x r-x, en este caso al ser la credencial efectiva la que condiciona el acceso a los ficheros, el fichero f2 creado es de u1. Cuando los permisos del ejecutable son rwx r-x r-x no se cambia la credencial y se produce error en open. El fichero creado en este caso es de u2