Examen de Sistemas Operativos ITIS Fuenlabrada — Junio 2009

Tiempo total: 3 horas.

Problema: Llamadas al sistema (5 puntos)

Escriba en C para Plan 9 un programa llamado `catsha1` cuyo propósito es crear un resumen hash SHA-1 del resultado de la concatenación de todos los ficheros acabados en `.txt` de un directorio. El orden de la concatenación debe ser el orden en el que se encuentran en el directorio (no hay que ordenar los ficheros de ninguna forma). El programa `catsha1` sólo admite un argumento para indicar el directorio sobre el que se quiere trabajar. Si no se le pasa ningún argumento, debe actuar sobre el directorio de trabajo actual. La salida del programa tiene que ser simplemente los 20 bytes expresados en hexadecimal del resumen SHA-1 generado.

El resumen SHA-1 debe calcularse ejecutando el comando *sha1sum*(1), que debe ejecutarse sin ningún parámetro. Este comando lee su entrada estándar hasta que se acaba, para luego calcular el resumen SHA-1 de los datos leídos y escribirlo por su salida estándar.

A continuación se muestra un ejemplo:

```
term% ls -n /tmp/
el flag -n de ls lista los ficheros sin ordenar
/tmp/a.txt
/tmp/afile
/tmp/z.txt
/tmp/c.txt
term% cat a.txt z.txt c.txt | sha1sum
312382290f4f71e7fb7f00449fb529fce3b8ec95
term% catsha1 /tmp
312382290f4f71e7fb7f00449fb529fce3b8ec95
```

Entrega: Copie un fichero llamado `catsha1-`*login*`.c`, **donde** *login* **es tu nombre de usuario,** en el directorio `/usr/elf/examen`. **ATENCIÓN: Sólo se puede efectuar la entrega una vez.**

Problema: Script de Shell (2 puntos)

Escriba un script de Shell RC llamado `hashexecs.rc` que reciba un directorio como argumento. El script debe crear un fichero llamado `hashes` con una línea para cada fichero de ese directorio que tenga permisos de ejecución para alguien (el dueño, grupo, u otros). Las líneas del fichero deben contener el resumen SHA-1 del fichero ejecutable, el dueño del fichero, y su ruta. Estos datos deben estar separados con un tabulador, de la siguiente forma:

```
sha-1   dueño   ruta
```

Si el fichero `hashes` existe, el script debe notificar el error y salir como corresponde.

A continuación se muestra un ejemplo:

```
term% hashexecs.rc $home/bin/rc
term% cat hashes
b48aa8720d02784d33b7046144f469341531148f        glenda  /usr/glenda/bin/rc/Acme
55c9828055f53817b394dc250aa4079a3dc25bb7        glenda  /usr/glenda/bin/rc/Contruner
87dbbe7f985fe47bcd42e8cf3ef5e9f6aa273312        glenda  /usr/glenda/bin/rc/P
6986e6e3f9eb69eaa2a98fe648f2b50dd420a094        glenda  /usr/glenda/bin/rc/W
a779d837b19da3090643ed42b805facc6638f598        glenda  /usr/glenda/bin/rc/cpfromhost
8128b40f66c935790d9e6c81114dd3633079df79        glenda  /usr/glenda/bin/rc/cphost
```

Entrega: Copie un fichero llamado `hashexecs-`*login*`.rc`, **donde** *login* **es tu nombre de usuario,** en el directorio `/usr/elf/examen`. **ATENCIÓN: Sólo se puede efectuar la entrega una vez.**

Problema: Teoría (3 puntos)

Responda en un folio de examen las siguientes cuestiones:

A) Supongamos un programa llamado `prog` que imprime `hola` por su salida estándar y `adios` por su salida de errores, en ese orden. ¿Cuál es la diferencia entre estas dos ejecuciones? Razone la respuesta.

```
term%  prog > /tmp/f1 >[2] /tmp/f1
term%  prog > /tmp/f1 >[2=1]
```

B) Suponga una máquina con un esquema de paginación simple (un nivel), páginas de 4KB y direcciones de 32 bits. Si un proceso está ejecutando el binario `/bin/ls` instalado en el laboratorio donde se está realizando este examen, y dicho proceso tiene la siguiente tabla de páginas:

página	**marco**
2	*32*
3	*12*
5	*33*
1	*11*
5	*42*
4	*13*
...	...

Las referencias en esta tabla están expresadas en decimal y la numeración de marcos y páginas empieza en 0.

Indique la dirección de memoria física en la que se encuentra el texto de la función `main`. Razone la respuesta indicando los pasos que ha dado y los comandos que ha ejecutado para calcular dicha dirección.

Pista: el comando nm (1) permite ver la tabla de símbolos de un binario.

C) Explique, paso a paso y brevemente, qué ha pasado con este proceso y qué significa cada una de las partes de la siguiente línea:

```
8.out 1435: suicide: sys: trap: fault read addr=0x0 pc=0x000033d3
```

D) En un sistema de ficheros de tipo UNIX con *i-nodos*. ¿Cuántos accesos a disco supone esta llamada al sistema? Razone la respuesta explicando brevemente cada paso en la operación.

```
open("/home/esoriano/doc/readme", OREAD);
```

Omita detalles relacionados con la comprobación de permisos, y suponga que el fichero existe, que el *i-nodo* del directorio raíz está en memoria principal, y que en todos los casos la entrada buscada se encuentra en el primer bloque de datos.

Solución

catsha1.c

```
#include <u.h>
#include <libc.h>

enum{
        Bufsize = 1024,
};

static void
usage(void)
{
        fprint(2, "usage: %s [dirname]\n", argv0);
        sysfatal("usage");
}


static void
dumpfile(char *name, int outfd)
{
        int fd;
        char buf[Bufsize];
        long nr;

        fd = open(name, OREAD);
        if(fd < 0)
                sysfatal("can't open file %s: %r", name);
        while((nr = read(fd, buf, Bufsize)) > 0)
                if(write(outfd, buf, nr) != nr)
                        sysfatal("can't write: %r");
        if(nr < 0)
                sysfatal("can't read from %s: %r", name);
        close(fd);
}


static int
isfiletxt(Dir *d)
{
        char *p;

        if(d->mode & DMDIR)
                return 0;
        p = strrchr(d->name, '.');
        if(p == nil)
                return 0;
        if(strcmp(p, ".txt") == 0)
                return 1;
        return 0;
}
```

```
static void
processdir(char *n, int outfd)
{
	int fd;
	Dir *d;
	char *fname;
	long nd;
	long i;

	fd = open(n, OREAD);
	if(fd < 0)
		sysfatal("can't open dir: %r");
	nd = dirreadall(fd, &d);
	if(nd < 0)
		sysfatal("can't read dir: %r");
	close(fd);
	for(i=0; i < nd; i++){
		if(isfiletxt(&d[i])){
			fname = smprint("%s/%s", n, d[i].name);
			dumpfile(fname, outfd);
			free(fname);
		}
	}
	free(d);
}


void
main(int argc, char *argv[])
{
	char *dirname = ".";
	int p[2];
	Waitmsg *msg;

	ARGBEGIN{
	default:
		usage();
	}ARGEND;

	if(argc > 1)
		usage();
	if(argc == 1)
		dirname = argv[0];
```

```
        if(pipe(p) < 0)
                sysfatal("can't make pipe: %r");
        switch(fork()){
        case -1:
                sysfatal("can't fork: %r");
        case 0:
                close(p[1]);
                dup(p[0], 0);
                close(p[0]);
                execl("/bin/sha1sum", "sha1sum", nil);
                sysfatal("can't exec /bin/sha1sum: %r");
        }
        close(p[0]);
        processdir(dirname, p[1]);
        close(p[1]);
        msg = wait();
        if(msg == nil)
                sysfatal("bug: can't happen!");
        if(msg->msg[0] != '\0'){
                free(msg);
                exits("sha1sum failed");
        }
        free(msg);
        exits(nil);
}
```

—

hashexecs.rc

```
#!/bin/rc

rfork e

if(! ~ $#* 1){
        echo 'usage: hashexecs.rc dir' >[1=2]
        exit usage
}

if(test -f hashes){
        echo 'file hashes exists' >[1=2]
        exit 'file exists'
}

files=`{ls -l $1 | grep '^[^ ]*x' | awk '{print $10}'}

for(i in $files){
        u=`{ls -l $i | awk '{print $4}'}
        h=`{cat $i | sha1sum}
        echo $h^'→'^$u^'→'^$i  >> hashes
}
exit ''
```

—

TEORÍA

A) La primera línea de comandos redirige la salida estándar al fichero `f1`, y después hace lo mismo con el descriptor de fichero 2 (la salida de errores estándar). En ambos casos, se abre el fichero en dicho

descriptor, truncándolo si existía, y ambos descriptores apuntan a una estructura distinta, cada una con su `offset`.

La segunda línea es diferente. Se abre el fichero `f1` en el descriptor 1, y después se duplica dicho descriptor en el descriptor 2. Ambos descriptores apuntan a la misma estructura, por tanto *comparten* su `offset`.

Por lo tanto, el contenido de `f1` al ejecutar la primera línea será `adios`, y el contenido después de ejecutar la segunda línea será `holaadios`. Para comprobarlo, se puede usar el siguiente programa:

```
#include<u.h>
#include<libc.h>

void
main(int, char**)
{
        print("hola");
        fprint(2, "adios");
        exits(0);
}
```

B) Ejecutando el comando `nm` sobre el fichero binario `/bin/ls` podemos ver la tabla de símbolos, y por tanto, determinar la dirección de memoria virtual en la que se encontrará el texto de `main`.

```
term% nm -n /bin/ls
    1020 T main
    12c7 T ls
    1478 T output
    15ed T dowidths
    17c7 T fileflag
    ...
```

La salida del comando nos indica que la dirección es la 0x1020 (hexadecimal). Con la calculadora `bc` podemos calcular que en decimal es 4128:

```
term% bc
ibase=16
1020
4128
```

Esta dirección cae en la segunda página, la página 1, ya que una página de 4KB tiene 4096 Bytes. Por tanto, el desplazamiento dentro del marco es 32. Si miramos la tabla de páginas, vemos que dicha página está alojada en el marco de página 11. Por tanto:

```
term% bc
(4096*11)+32
45088
```

Por tanto, la dirección física será 45088 en decimal, 0xb020 en hexadecimal.

C) El mensaje de error nos indica que el proceso tenía el PID 1435, y estaba ejecutando un binario llamado `8.out`. Nos indica también que el proceso ha abortado su ejecución porque ha recibido una nota puesta por el sistema. Dicha nota fue causada por leer una dirección de memoria inválida, en este caso la dirección 0x0. También nos indica el valor del contador de programa en el que se intentó acceder a dicha dirección.

La primera página de memoria virtual es una página inválida. Cuando se intenta acceder a dicha dirección, la MMU genera un `trap` de fallo de página. El sistema comprueba que dicha página no es válida, y manda una nota al proceso con este texto:

```
suicide: sys: trap: fault read addr=0x0 pc=0x000033d3
```

Cuando el proceso vuelve a ejecutar, intenta manejar dicha nota sin éxito, abortando su ejecución e imprimiendo el mensaje por su salida estándar de errores.

D) Los pasos son:

- Se **accede a disco** para leer el primer bloque de datos del *i-nodo* del raíz (que se referencia en el primer bloque de indirección directo, como dice el enunciado). Se busca el la entrada con nombre `home` entre las entradas de ese directorio. Se encuentra y se obtiene el número de su *i-nodo*.
- Se **accede a disco** para conseguir el *i-nodo* de `/home`.
- Se **accede a disco** para leer el primer bloque de datos de `/home` y conseguir sus entradas de directorio. Se busca la entrada `esoriano` y se obtiene su número de *i-nodo*.
- Se **accede a disco** para conseguir el *i-nodo* de `/home/esoriano`.
- Se **accede a disco** para leer el primer bloque de datos de `/home/esoriano` y conseguir sus entradas de directorio. Se busca la entrada `doc` y se obtiene su número de *i-nodo*.
- Se **accede a disco** para conseguir el *i-nodo* de `/home/esoriano/doc`.
- Se **accede a disco** para leer el primer bloque de datos de `/home/esoriano/doc` y conseguir sus entradas de directorio. Se busca la entrada `readme` y se obtiene su número de *i-nodo*.
- Se **accede a disco** para conseguir el *i-nodo* de `/home/esoriano/doc/readme`, se busca la primera entrada libre en la tabla de descriptores de fichero del proceso, y se guarda una referencia a dicho *i-nodo*.

  Por tanto, son necesarios 8 accesos a disco.

—