Examen de Sistemas Operativos ITIS Fuenlabrada — Mayo 2011

Problema: Llamadas al sistema (3 puntos)

Implemente en C para Plan 9 un programa `hlines` que a partir de una URL que se le pasa como argumento, escriba por su salida estándar el número de líneas que tiene la página correspondiente. Para descargar la URL deberá ejecutar el comando *hget(1)*, que escribe por su salida estandar el contenido de la página indicada por la URL que se le pasa como argumento. Por ejemplo, podríamos ejecutar este comando para almacenar la página web correspondiente en un fichero:

```
; hget http://google.es > pagina
;
```

Un ejemplo de ejecución del programa `hlines` podría ser:

```
; hlines http://lsub.org
145
;
```

El programa no puede ejecutar `rc` ni puede usar ficheros temporales, y debe notificar los errores de ejecución como corresponda y acabar con el estatus apropiado.

Entrega: Copie un fichero llamado `hlines-`*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: Shell Script (3 puntos)

Escriba un script de RC llamado `hcomment.rc` que reciba un número indeterminado de argumentos que especifiquen URLs. El script debe imprimir por su salida estándar la lista de las máquinas (hosts) de las páginas especificadas que comienzan con un comentario HTML. Las URLs (los argumentos) pueden contener mayúsculas y minúsculas, siempre comiezan con `http://`, y los nombres de los dominios y hosts sólo pueden contener carácteres alfanuméricos. Se supone que las páginas están siempre bien formadas.

El script debe escribir por su salida estándar la lista de hosts en minúsculas, ordenada alfabéticamente, y sin duplicados. Por ejemplo, una ejecución podría ser:

```
; hcomment.rc HTTP://AA.ES/INFO.HTML http://www.bb.com http://www.cc.org/a.html
aa.es
www.bb.com
www.cc.org
;
```

Los comentarios de HTML tienen la siguiente sintaxis:

```
<!-- Esto es un comentario...
y así acaba (en otra línea o en la misma!) -->
```

La página podría tener líneas vacías o líneas sólo con espacios/tabuladores antes del comentario, que deberían ser ignoradas. Dicho de otra forma, si el fichero empieza con varias líneas vacías seguidas del comentario, el host sí tiene que aparecer en la lista impresa por la salida estándar.

Entrega: Copie un fichero llamado `hcomment-`*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 (4 puntos)

Responda en un folio de examen las siguientes cuestiones:

A) Suponiendo este código:

```
int a;

void
main(int argc, char*argv[])
{
        a = 3;
        if(fork() == 0)
                a = 4;
        print("value: %d address: %p\n", a, &a);
        exits(nil);
}
```

Responda a estas preguntas razonando la respuesta:

1.- ¿Qué imprime el programa?

2.- ¿Siempre se imprime eso en el mismo orden?

3.- ¿Las direcciones de memoria impresas son físicas o virtuales?

4.- ¿Las direcciones virtuales son las mismas?

5.- ¿Las direcciones físicas son las mismas?

6.- ¿Se usará el mismo marco de página para la variable `a` en los procesos involucrados?

B) Tenemos un disco duro con sectores de 512 bytes, formateado con un sistema de ficheros de tipo UNIX con tamaño de bloque lógico de 4KiB. ¿Cuánto espacio de disco queda comprometido al ejecutar el siguiente comando?

```
; echo hola > /tmp/afile
```

Puede expresar el tamaño de un i-nodo como `sizeof(Inodo)` y puede hacer las suposiciones que quiera respecto al directorio `/tmp`.

C) Los usuarios de un sistema para estaciones de trabajo multimedia experimentan una mala respuesta en los programas interactivos. Uno de los ingenieros propone cambiar el planificador (scheduler) del sistema operativo, que implementa una política Round-Robin, aumentado al doble el cuanto actual. Otro ingeniero propone cambiar la política del planificador a FCFS (FIFO). ¿Apoyaría alguna de las propuestas? Justifique la respuesta.

Solución

hlines.c

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

enum{
        Bsize = 8*1024,
};

int
countlines(int fd)
{
        int eols = 0;
        char buf[Bsize];
        int i;
        long nr;

        while((nr = read(fd, buf, Bsize)) > 0)
                for(i = 0; i < nr; i++)
                        if(buf[i] == '\n')
                                eols++;
        if(nr < 0)
                sysfatal("can't read: %r");
        return eols;
}

int
mkhget(char *url)
{
        int p[2];

        if(pipe(p) < 0)
                sysfatal("can't create pipe: %r");
        switch(fork()){
        case -1:
                sysfatal("can't fork: %r");
        case 0:
                close(p[1]);
                dup(p[0], 1);
                close(p[0]);
                execl("/bin/hget", "hget", url, nil);
                sysfatal("can't exec: %r");
        }
        close(p[0]);
        return p[1];
}

void
main(int argc, char *argv[])
{
        char *s = nil;
        Waitmsg *m;
        int fd;
        int l;
```

```
        if(argc != 2){
                fprint(2, "usage: hlines urls");
                exits("usage");
        }
        fd = mkhget(argv[1]);
        l = countlines(fd);
        close(fd);
        m = wait();
        if(m == nil)
                sysfatal("no child");
        if(m->msg[0] != 0){
                s = "hget had problems";
                fprint(2,"%s\n", s);
        }else
                print("%d\n", l);
        free(m);
        exits(s);
}
```

—

hcomments.rc

```
#!/bin/rc

# testing
# ! hcomments.rc  http://lsub.org/who/esoriano/index.html HTTP://GOOGLE.ES http://yahoo.es

rfork e

if(~ $#* 0){
        echo usage: hcomments.rc url  [url ...] >[1=2]
        exit usage
}

{for(i){
        url=‘{echo $i | tr A-Z a-z}
        if(hget $url | grep -v ’^[ →]*$’ | sed 1q | grep ’^<!--’  > /dev/null ){
                echo $url | sed ’s/http:\/\/([a-zA-Z0-9.]+).*/\1/’
        }
}}| sort -u
```

—

A)

1.- El programa crea un proceso nuevo. Ambos procesos imprimen el mensaje con el valor de la variable y su dirección de memoria. La llamada `fork()` crea un proceso hijo que no comparte memoria con el padre. Como `fork()` retorna 0 en el proceso hijo y el PID del hijo en el padre, la variable `a` en el hijo es 4 y la variable `a` en el proceso padre es 3.

2.- No. Los procesos son concurrentes, y no se sabe el orden en el que se van a planificar, ni con un único procesador ni con varios. Los mensajes se pueden ver en cualquier orden.

3.- Las direcciones impresas son virtuales. Los programas manejan direcciones virtuales. El hardware (MMU) se encarga de traducirlas a direcciones físicas en base a la tabla de páginas del proceso.

4.- Al imprimir la dirección de memoria de la variable, veremos que es la misma para los dos procesos. Los dos procesos están ejecutando el mismo binario, por lo que su código es el mismo.

Además, el valor de la variable difiere, como hemos dicho anteriormente, y la dirección es la misma. Esto sólo tiene sentido si es una dirección de memoria virtual.

5.- No pueden serlo, ya que ambos procesos tienen que tener su propio segmento de datos, de BSS y pila. Incluso si el sistema usa alguna optimización como *Copy-on-Write*, las direcciones físicas serán distintas, ya que cada proceso pasará a tener su propia copia del segmento de datos cuando se realiza la primera escritura. La traducción entre memoria virtual y memoria física se hace en base a la tabla de páginas de cada proceso, y la traducción no será la misma.

6.- No, dada la respuesta anterior. Cada proceso tiene sus propios marcos de página para las páginas de sus segmentos de datos, BSS, y pila.

B) Suponiendo que el fichero `/tmp/afile` no existe antes de ejecutar el comando, se necesitaría añadir una entrada de directorio en los bloques de datos del directorio `/tmp`. Suponiendo que hay espacio suficiente en el bloque de datos del directorio, no haría falta comprometer más espacio en lo que respecta a `/tmp`. La nueva entrada de directorio relacionará el nombre del fichero con su i-nodo.

Habrá que comprometer espacio para el i-nodo del fichero. Suponiendo una implementación simple con un vector de i-nodos, se comprometerá una entrada libre en dicho vector. Una entrada en dicha tabla serán `sizeof(Inode)` bytes.

El i-nodo contiene los metadatos del fichero y las referencias a los bloques que contienen los datos del fichero. En este caso, el fichero tiene 5 bytes, `''hola\n''`, pero el bloque lógico del sistema de ficheros es de 4 KiB. Por tanto, se comprometerá un bloque lógico como primer *bloque directo* del fichero, del que únicamente se están usando 5 bytes (fragmentación interna).

En total, podemos considerar que se han comprometido `sizeof(Inode)`+ 4096 bytes de almacenamiento secundario.

C) Round-Robin es expulsiva, de tal forma que se a un proceso se le expulsa del procesador cuando ha consumido todo su cuanto (esto es, no se ha bloqueado antes de consumir su cuanto).

Un cuanto grande penaliza a las aplicaciones interactivas, ya que tendrán que esperar más en la cola de procesos listos para ejecutar, y por lo tanto se introduce más latencia en la respuesta al usuario. Por tanto, no deberíamos apoyar la primera propuesta. Se podría proponer la reducción del tiempo de cuanto, pero hay que tener en cuenta que un cuanto demasiado pequeño implica desperdiciar más tiempo de CPU en cambios de contexto y puede que tampoco fuese apropiado en este caso en concreto.

La segunda propuesta tampoco es apropiada, ya que la política FCFS (FIFO) no es expulsiva, por lo que el proceso que ejecuta no abandona el procesador hasta que no acaba su ráfaga de CPU. En realidad, se puede ver como si, en el caso anterior, el cuanto fuese infinito. Esta política no es apropiada para aplicaciones interactivas, que se verían gravemente afectadas.

Se podría proponer un esquema de planificación basado en colas de prioridades, en el que se asignaran prioridades altas a los procesos interactivos.

—