Examen de Sistemas Operativos ITIS Fuenlabrada — Mayo 2010

Tiempo total: 3 horas.

Problema: Llamadas al sistema (3,5 puntos)

Implemente en C para Plan 9, de la forma más eficiente posible, un programa `wcout` que reciba como argumentos nombres de ficheros. El programa debe ejecutar dichos ficheros y contar el número de bytes que escribe cada uno por su salida estándar. Si no se le pasan argumentos, el programa debe fallar y notificar el modo de uso como corresponda. Por ejemplo, suponiendo que

```
; lc /tmp/dir
fich1 fich2
; cat /tmp/dir/fich1
#!/bin/rc
echo hola
; cat /tmp/dir/fich2
#!/bin/rc
date
;
```

una posible ejecución de `wcout` sería la siguiente:

```
; wcout /tmp/dir/fich1 /tmp/dir/fich2
/tmp/dir/fich1  5
/tmp/dir/fich2  29
;
```

Esto es así dado que `fich1` escribe 5 bytes en su salida estándar y `fich2` escribe 29 bytes. No se permite almacenar en ficheros auxiliares la salida de los programas ejecutados ni se permite utilizar *wc*(1) para implementar el programa. Se debe usar *pipe*(2) como mecanismo para obtener la salida de cada uno de ellos. **Entrega:** Copie un fichero llamado `wcout-`*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 (3,5 puntos)

Escriba un script de RC llamado `processes.rc` que dado un argumento que indica un nombre de programa, imprima periódicamente por su salida estándar los procesos que están ejecutando dicho programa con el siguiente formato (las letras en cursiva no indican salida del programa):

```
; processes.rc -u jperez acme
USER JPEREZ RUNS ACME, PID: 334
USER JPEREZ RUNS ACME, PID: 434
(espera)
USER JPEREZ RUNS ACME, PID: 334
USER JPEREZ RUNS ACME, PID: 665
(espera)
```

El script puede aceptar como primer argumento el modificador `-u` seguido de un nombre de usuario. Si es ese el caso, únicamente se imprimirán los procesos de dicho usuario. En otro caso, deberá contemplar los procesos de todos los usuarios. El script debe usar un tiempo de *polling* de 5 segundos. Si se usa el script incorrectamente, deberá acabar con el status adecuado y notificar el fallo como corresponda. **Entrega:** Copie un fichero llamado `processes-`*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) ¿Qué contiene un fichero binario ejecutable (*a.out*) ? ¿Para qué sirve cada parte?

B) Suponga un asignador de memoria dinámica para `malloc` que reserva la memoria en base a bloques fijos de 2 Kb. Si ejecutamos un reproductor multimedia cuya estructura principal es una lista enlazada de nodos:

```
typedef struct Song Song;
struct Song{
        char name[128];
        char album[128];
        char artist[128];
        char path[128];
        Song *next;
};
```

¿Tendremos fragmentación? ¿Externa o interna? Suponiendo que sí tiene, ¿Cuánta memoria se estaría perdiendo aproximadamente para una biblioteca musical con 10000 canciones?

C) En un sistema de ficheros de tipo UNIX, ¿Cuál es la diferencia entre un enlace duro y un enlace simbólico? ¿Puede estar *roto* un enlace duro? ¿Por qué?

Solución

wcout-simple.c

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

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

static void
waitforall(void)
{
        Waitmsg *msg;
        int errs = 0;

        while((msg = wait()) != nil){
                if(msg->msg[0] != 0)
                        errs++;
                free(msg);
        }
        if(errs)
                exits("some errors");
        exits(nil);
}


static void
fwcout(char *p)
{
        int     fd[2];
        char    buf[1024];
        long    nr;
        long    tot;
```

```
        if(pipe(fd) < 0)
                sysfatal("pipe: %r");
        switch(fork()){
        case -1:
                sysfatal("fork: %r");
        case 0:
                dup(fd[1], 1);
                close(fd[0]);
                close(fd[1]);
                execl(p, p, nil);
                sysfatal("exec: %s: %r", p);
        }
        close(fd[1]);
        tot = 0;
        for(;;){
                nr = read(fd[0], buf, sizeof(buf));
                if(nr <= 0)
                        break;
                tot += nr;
        }
        close(fd[0]);
        if(nr < 0)
                sysfatal("read: %r");
        print("%s\t%ld\n", p, tot);
        waitforall();
}

void
main(int argc, char *argv[])
{
        int i;

        ARGBEGIN{
        default:
                usage();
        }ARGEND;

        if(argc == 0)
                usage();
        for(i = 0; i < argc; i++){
                switch(fork()){
                case -1:
                        sysfatal("fork: %r");
                case 0:
                        fwcout(argv[i]);
                }
        }
        waitforall();
}
```

—

processes.rc

```
#!/bin/rc

#  The only regexp metacharacter expected in
#  user and program names is dot (.)
```

```
rfork e

pt=5

if(~ $1 '-u'){
        if(! ~ $#* 3){
                echo usage: processes.rc [-u name] program >[1=2]
                exit usage
        }

        uregexp= '{echo $2  | sed 's/\./\\./g'}
        pregexp='{echo $3  | sed 's/\./\\./g'}
}
if not{
        if(! ~ $#* 1){
                echo usage: processes.rc [-u name] program >[1=2]
                exit usage
        }
        uregexp='.+'
        pregexp='{echo $1  | sed 's/\./\\./g'}
}

while(sleep $pt){
        ps |  grep '^'^$uregexp^'[→ ]'  | \
                        grep '[ →]'^$pregexp^'$' | \
                        awk '{print "user " $1 " runs " $7 ", pid: " $2}'  | \
                        tr a-z A-Z
}
```

—

TEORÍA

A) Como indica la página de manual *a.out*(6), un fichero ejecutable de este tipo contiene una cabecera y ciertas secciones. La cabecera, además de tener un *número mágico* para identificar el tipo de fichero, contiene (entre otras cosas) el tamaño del texto (instrucciones) del programa, el tamaño de los datos inicializados (variables globales inicializadas, etc.), el tamaño del BSS para datos no inicializados, el tamaño de la tabla de símbolos, y el punto de entrada para el programa (dirección virtual del comienzo del programa). A continuación, en el fichero se incluye el texto del programa y los datos inicializados. Los datos no inicializados (BSS) no se incluyen, ya que todos se inicializan a cero. La tabla de símbolos, necesaria sólo a efectos de depuración, también se incluye en el fichero (aunque se puede eliminar con el comando `strip` para ahorrar espacio en disco).

B) Si usamos un asignador con bloques fijos, la fragmentación siempre será interna, ya que no quedan huecos entre los trozos de memoria marcados como ocupados. En este caso, una granularidad de 2 Kb, se pierde bastante memoria por la fragmentación. Al tener una lista enlazada, cada nodo de la lista se reservará llamando a `malloc` con el tamaño del nodo, `sizeof(Song)`.

El tamaño de la estructura `Song` en un Intel 32-bit es de 516 bytes. Por tanto, por cada nodo reservado se pierden 1532 bytes. Por tanto, aproximandamente ¾ de la memoria se pierde por fragmentación interna. Para una biblioteca de 10000 canciones, que necesita una lista enlazada de 10000 estructuras de este tipo, se pierden aproximadamente 14 Mb de memoria.

C) En un sistema de ficheros de tipo UNIX, las entradas de directorio relacionan un nombre de fichero con el i-nodo correspondiente. Un enlace duro es un nuevo nombre para un i-nodo. Mientras que haya algún nombre asociado, el i-nodo prevalece. Para eso se usa una cuenta de referencias en el propio i-nodo. Cuando su contador de referencias llega a cero, se libera el i-nodo y sus bloques.

Un enlace simbólico es en realidad un fichero especial distinto (con su correspondiente i-nodo) cuyos datos contienen la ruta de un fichero, el fichero al que *apunta*. Un enlace simbólico se puede romper si se borra el fichero al que apunta. Sin embargo, al menos que el sistema de ficheros sea inconsistente, un enlace duro no se puede romper, ya que es simplemente una referencia al i-nodo.

—