SISTEMAS OPERATIVOS II
Tercer curso Ingeniería Técnica de Sistemas.
Curso 2009-200

Práctica 4: Sistema de ficheros en UNIX: Redirección.

Continuar la codificación de un intérprete de comandos (shell) en UNIX. Al igual que en la práctica anterior

- Los argumentos entre corchetes [] son opcionales.
- Los argumentos separados por `|` indican que debe ir uno u otro, pero no ambos simultaneamente.
- No debe dilapidar memoria (ejemplo: variable que se asigna cada vez que se llama a una función y no se libera).
- Cuando el shell no pueda ejecutar una acción por algún motivo, debe indicarlo con un mensaje como el que se obtiene con `sys_errlist[errno]` o con *perror()* (por ejemplo, si no puede cambiar de directorio debe indicar por qué).
- En ningún caso debe producir un error de ejecución (segmentation, bus error ...), salvo que se diga explicitamente
- Las direcciones de memoria deben mostrase en **hexadecimal**.
- La información que se nuestra en pantalla no debe incluir en ningun caso líneas en blanco.
- El shell leerá de su entrada estándar y escribirá en su salida estándar, de manera que podría ser ejecutado un archivo de comandos invocando al shell con su entrada estándar redireccionada a dicho archivo.

El shell ha de llevar una lista de los ficheros que mapea en memoria con el comando *mmap* (similar a la que se hizo para el comando *malloc* y las memorias compartidas). Para cada fichero mapeado guardará, además de la dirección, el tamaño del mapeo y el nombre del fichero.

Añadir al intérprete de comandos de las prácticas anteriores las siguientes funciones

Redireccion Modificar los comandos *exec, background, pri, background-pri, exec-pri* y la ejecución en primer plano de manera que se permita la redirección de la entrada estándar, la salida estándar y/o el error estándar

- **[exec|background|pri|exec-pri|background-pri] com... <fich** Ejecuta (sin crear proceso, creando proceso en primer o segundo plano, con o sin cambio de prioridad, tanto especificándole el entorno como sin hacerlo, según corresponda) el comando *com* (eje-

cutable con parámetros) con la entrada estándar redireccionada al fichero *fich*.

- **[exec|background|pri|exec-pri|background-pri] com. . . >fich** Ejecuta (sin crear proceso, creando proceso en primer o segundo plano, con o sin cambio de prioridad, tanto especificándole el entorno como sin hacerlo, según corresponda) el comando *com* (ejecutable con parámetros) con la salida estándar redireccionada al fichero *fich*.

- **[exec|background|pri|exec-pri|background-pri] com. . . #fich** Ejecuta (sin crear proceso, creando proceso en primer o segundo plano, con o sin cambio de prioridad, tanto especificándole el entorno como sin hacerlo, según corresponda) el comando *com* (ejecutable con parámetros) con el error estándar redireccionado al fichero *fich*.

- Los comandos de redirección >, <, # deben ser compatibles entre sí y con la ejecución en segundo plano, la ejecución sin crear proceso, con la ejecución con las prioridades cambiadas, y con la especificación de un entorno. Ejemplos:

  ```
  #background-pri 15 a.out arg1 arg2 ** TERM HOME NUEVA=uno <f1 >f2 #f3
  ```

  ejecuta en segundo plano *a.out arg1 arg2* en con entorno que solo contiene las variables TERM HOME y NUEVA, con la entrada, la salida y el error estándar redireccionados, además establece su prioridad a 15.

pipe **com1 % com2** Ejecuta *com1* (ejecutable con parámetros) redireccionando su salida estándar a la entrada estándar de *com2* (ejecutable con parámetros). Tanto *com1* como *com2* representan ejecutables con parámetros.

```
#pipe ls -lR . . / % wc -l -c
```

ejecuta ls -lR . . / y redirecciona la salida a la entrada estándar de un proceso que ejecuta wc -l -c

mmap **[fichero]** mapea en memoria el fichero especificado. Se mapeará el fichero en toda su longitud a partir del *offset* 0. Devuelve la dirección de memoria donde lo ha mapeado. Utiliza la llamada *mmap*. Si no se especifica fichero nos informa de las direcciones de memoria donde hay mapeados ficheros, indicándonos la dirección, el tamanño del mapeo y el fichero que hay mapeado en ella

munmap **[[-f] dir]** desmapea la dirección de memoria *dir* del fichero que haya mapeado en ella. Si en esa posición de memoria no hay nada mapeado con el comando *mmap* nos dará un aviso y no la desmapeará. Si se especifica -f, *dir* representa el numbre del fichero que queremos desmapear. Si no se especifica *dir* nos informa de las direcciones de memoria donde hay mapeados ficheros, indicándonos la dirección, el tamanño del mapeo y el fichero que hay mapeado en ella

mem [-v] Muestra las siguientes direcciones de memoria:

- ficheros mapeados en memoria
- zonas de memoria asignadas con el comando *malloc*
- zonas donde hay mapeada una región de memoria compartida

con la opción [-v] muestra las direcciones de memoria de

- (al menos 3) variables globales
- (al menos 3) variables locales de *main*
- *main* y (al menos 3) funciones del programa llamadas desde *main*

rec [-a|-n|-d] [-sN] n Invoca a la función recursiva n veces, la opción -a, -n o -d indica si la memoria asignada con malloc en dicha función debe liberarse (a)ntes de la siguiente llamada recursiva, (d)espués o (n)o liberarse. Si no se indica una opción se supone que la memoria se libera despues de la llamada (-d). En parámetro -sN indica cuanto tiempo (en segundos) debe esperar la última llamada antes de terminar. (ejemplo -s10 indicaría que la ultima iteracción de la función recursiva debería hacer una espera (mediante *sleep*) de 10 segundos. La función recursiva recibe tres parámetros: uno que indica el número de veces que se tiene que invocar, otro que indica cuando liberar la memoria asignada con malloc y un tercero que indica cuanto tiene que esperar la última iteración Además esta función tiene 3 variables, un array automatico de 1024 caracteres, un array estático de 1024 caracteres y un puntero a caracter.
Esta función debe hacer lo siguiente

1. asignar memoria (mediante malloc) al puntero para 1024 caracteres.
2. imprimir
   - el valor del parámetro que recibe así como la dirección de memoria donde se almacena.
   - el valor del puntero así como la dirección de memoria donde se almacena.
   - la dirección de los dos arrays (el nombre del array como puntero).
3. si se ha especificado la opción -a liberar la memoria asignada al puntero.
4. invocarse a si misma con (n-1) como parámetro (si n>0).

5. si es la última iteración (n=0) y se ha especificado `-sN` esperar N segundos (por medio de *sleep*).
6. si se ha especificado la opción -d liberar la memoria asignada al puntero.

Un posible código para la función recursiva (sin las definiciones de constantes) podría ser:

```
void recursiva (int n, int liberar, int delay)
{
char automatico[TAMANO];
static char estatico[TAMANO];
void * puntero;

puntero=(void *) malloc (TAMANO);
printf ("parametro n:%d en %p\n",n,&n);
printf ("valor puntero:%p en direccion: %p\n", puntero,&puntero);
printf ("array estatico en:%p \n",estatico);
printf ("array automatico en %p\n",automatico);
if (liberar==ANTES)
   free (puntero);
if (n>0)
   recursiva(n-1,liberar,delay);
if (liberar==DESPUES)
   free (puntero);
if (n==0 && delay)        /* espera para la ultima recursividad*/
   sleep (delay);
}
```

FORMA DE ENTREGA

Como en las prácticas anteriores

FECHA DE ENTREGA Viernes 5 Febrero 2010