SISTEMAS OPERATIVOS II

Tercer curso Ingeniería Técnica Informática de Sistemas. Curso 2011-2012

Práctica 1: Procesos en Unix: Memoria

Comenzar la codificación de un intérprete de comandos (shell) en UNIX que se irá completado en sucesivas prácticas. Nótese que los comandos aquí descritos deben interpretarse de la siguiente manera

- Los argumentos entre corchetes [] son opcionales.
- Los argumentos separados por `|` indican que debe ir uno u otro, pero no ambos simultaneamente.
- El intérprete de comandos debe aceptar y entender la sintaxis aquí propuesta, pero no tiene que forzarla. (por ejemplo, si hay varios argumentos deben aceptarse en el orden especificado, pero puede resultar mas cómodo de programar asumiendo que pueden ir en cualquier orden)

Además deben tenerse en cuenta las siguientes indicaciones

- **En ningún caso debe producir un error de ejecución (segmentation, bus error . . .)**. La práctica que produzca un error en tiempo de ejecución no será puntuada. Excepcionalmente se admitirá un error en tiempo de ejecución en algunos comandos: en estos casos se indicará explicitamente (***)
- No debe dilapidar memoria (ejemplo: variable que se asigna cada vez que se llama a una función y no se libera). **NO SE REFIERE A DECLARAR LOS ARRAYS DE TAMAÑO PEQUEÑO**
- Cuando el shell no pueda ejecutar una acción por algún motivo, **NO DEBE TERMINAR SU EJECUCIÓN**, simplemente lo indicará 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).
- Las direcciones de memoria deben mostrase en **hexadecimal**.
- La información que se nuestra en pantalla no debe incluir en ningun caso líneas en blanco (ni líneas de '*' ni de '=',. . .).
- 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 llevará además una (o varias) lista(s) (**de implementación libre**) de direcciones de memoria, que incluye las direcciones de memoria compartida que tiene en su espacio de direcciones (obtenidas com *shmat*), las direcciones donde tiene mapeado un fichero y las direcciones de memoria que se ha asignado con *malloc* cuando así se le ha requerido con el comando *memoria -malloc*. Además, para cada una de estas direcciones almacenará tambien el tamaño correspondiente, el instante en que se ha creado, si se trata de memoria de malloc, memoria compartida o un fichero mapeado y, en su caso, el nombre del fichero mapeado si procede. Los contenidos de dicha lista deben ser coherentes con la salida del comando *pmap*

En esta práctica el shell tendrá capacidad para crear y acceder a sistemas de ficheros implementados sobre zonas de memoria; el comando *sfmemoria -crear* crea y/o formatea dichos sistema de ficheros. Las opciones *-copiar, -mover y -borrar* del comando *sfmemoria* acceden a los ficheros en dichos sistemas de ficheros..

Comandos a implementar en esta práctica

fin Termina la ejecución del intérprete de comandos.

salir Termina la ejecución del intérprete de comandos.

autores Indica los nombres y los logins de los autores de la práctica.

obtenerpid [-p] Muestra el pid del proceso. Con -p muestra el pid del proceso padre del shell

cd [dir] Cambia el directorio actual del shell a *dir*. Si no se especifica *dir* muestra el directorio actual del shell

memoria [-n] [-malloc|-mmap|-shared|] [arg. . .] Asigna (o desasigna) memoria en el shell proviniente de *malloc*, de mapear on fichero con *mmap* o de maperar una región de memoria compartida (con *shmat*)

- **memoria -malloc *tam*** Asigna en el shell *tam* bytes mediante malloc y nos informa de la dirección donde se ha asignado. Además guardará esa dirección, junto con el tamaño, y el instante de la asignación en una lista (de implementación libre). Si no se especifica tamaño los dará una lista de las direcciones de memoria asignadas **con el comando memoria -malloc**
- **memoria -n -malloc *tam*** Desasigna (con *free*) en el shell el bloque de tamaño *tam* aisgnado mediante *malloc* y lo elimina de la lista. Si no se especifica tamaño los dará una lista de las direcciones de memoria asignadas **con el comando memoria -malloc**
- **memoria -mmap *fichero* [private|shared] [r][w][x]** Mapea en memoria el fichero especificado en toda su longitud a partir del offset 0 y nos informa de la dirección de memoria donde ha sido

mapeado. *private o shared* representa el tipo de mapeo y *r, w o x* los permisos del mapeo. Además guardará esa dirección, junto con el tamaño, el nombre del fichero y el instante del mapeo en una lista (de implementación libre). Utiliza la llamada *mmap*. Si no se especifica *fichero* nos informa de las direcciones de memoria donde hay mapeados ficheros, indicándonos la dirección, el tamaño del mapeo, el fichero que hay mapeado en ella y el instante en que se mapeó.

- **memoria -n -mmap *fichero*** Desmapea el fichero *fichero* mapeado en memoria y elimina dicha dirección de la lista de direcciones. Si no se especifica *fichero* nos informa de las direcciones de memoria donde hay mapeados ficheros, indicándonos la dirección, el tamaño del mapeo, el fichero que hay mapeado en ella y el instante en que se mapeó.
- **memoria -shared *key [tam]*** Obtiene la memoria compartida de clave *key*, la mapea en el espacio de direcciones del proceso y nos informa de la dirección donde se ha mapeado. Además guardará esa dirección, junto con el tamaño y el instante del mapeo en una lista (de implementación libre). Si no se especifica *key* nos informa de las direcciones de memoria donde hay mapeada memoria compartida, indicándonos la dirección, el tamaño del mapeo y el instante en que se mapeó. Si se especifica *tam* la zona debe crearse del tamaño especificado (no debe existir previamente); de no indicarse *tam* se supondrá que la zona de memoria compartida ya existe y simplemente se mapea (con *shmat*).
- **memoria -n -shared *tam*** Desasigna (con *shmdt*) en el shell la zona de memoria compartida con tamaño *tam* y la elimina de la lista de direcciones de memoria compartida.

memoria -n *[dir]* Elimina de la lista de direcciones de memoria del shell la dirección *dir* y la desasigna (con *free* si fue obtenida con malloc, con *munmap* si corresponde a un fichero mapeado o con *detach* si corresponde a una zona de memoria compartida). Si no se especifica *dir* nos mostrará todas las direcciones de la lista, así como sus detalles (tamaño, instante ...)

memoria Muestra la lista direcciones de memoria del proceso en donde hay mapeadas memorias compartidas, se han mapeado ficheros o han sido obtenidas con el comando *memoria -malloc*. Para cada dirección mostrará, ademas, el tamaño del bloque, si se trata de una zona de malloc, memoria compartida o fichero mapeado, el instante en que se ha creado.

esmemoria -read|-write arg...

- **esmemoria -read *file dir*** Lee el fichero *file* en (copia a) la dirección de memoria *dir*. NO DEBE COMPROBARSE QUE LA DIRECCION ES VALIDA. (podría producir error en caso de que *dir* no fuese adecuada)(***)
- **esmemoria -write *file dir cont [-o]*** Copia desde la dirección de memoria *dir*, *cont* bytes en el fichero *file*. NO DEBE COMPROBARSE QUE LA DIRECCION ES VALIDA. (podría producir error en caso de que *dir* no fuese adecuada)(***). Con -o sobreescribe el fichero en caso de que exista

mostrarmem ***dir [cont]*** Muestra los contenidos de *cont* bytes a partir de la posición de memoria *dir*. Si no se especifica *cont* imprime 25 bytes. Para cada byte imprime, en distintas líneas, el caracter asociado (en caso de no ser imprimible imprime un espacio en blanco) y su valor en hexadecimal. Imprime 25 bytes por línea. NO DEBE COMPROBARSE QUE LA DIRECCION ES VALIDA. (podría producir error en caso de que *dir* no fuese adecuada)(***). Ejemplo

```
->memoria -mmap shell.c shared rw
    fichero shell.c mapeado en 0xb8019000
->mostrarmemoria 0xb8019000 300
  #  i  n  c  l  u  d  e     <  u  n  i  s  t  d  .  h  >     #  i  n  c  l
 23 69 6E 63 6C 75 64 65 20 3C 75 6E 69 73 74 64 2E 68 3E 0A 23 69 6E 63 6C
  u  d  e     <  s  t  d  i  o  .  h  >     #  i  n  c  l  u  d  e     <  s
 75 64 65 20 3C 73 74 64 69 6F 2E 68 3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73
  t  r  i  n  g  .  h  >     #  i  n  c  l  u  d  e     <  s  t  d  l  i  b
 74 72 69 6E 67 2E 68 3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73 74 64 6C 69 62
  .  h  >     #  i  n  c  l  u  d  e     <  s  y  s  /  t  y  p  e  s  .  h
 2E 68 3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73 79 73 2F 74 79 70 65 73 2E 68
  >     #  i  n  c  l  u  d  e     <  s  y  s  /  s  t  a  t  .  h  >     #
 3E 0A 23 69 6E 63 6C 75 64 65 20 3C 73 79 73 2F 73 74 61 74 2E 68 3E 0A 23
```

recursiva ***[-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). El parámetro -sN indica cuanto tiempo (en segundos) debe esperar la última llamada antes de terminar. (por 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 512 caracteres, un array estático de 512 caracteres y un puntero a caracter.
Esta función debe hacer lo siguiente

1. asignar memoria para 1024 caracteres al puntero (mediante malloc).
2. imprimir
   - el valor del parámetro que recibe así como la dirección de memoria donde se almacena dicho parámetro.
   - 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 en la ultima recursividad*/
   sleep (delay);
}
```

El siguiente comando se refieren a los sistemas de ficheros creados sobre memoria. **LA IMPLEMENTACION** de dichos sistemas de ficheros (métodos de asignación y o contabilidad) **ES LIBRE**. Se trata de una práctica de memoria, no de sistemas de ficheros.

sfmemoria -crear|-copiar|-mover|-borrar|listar|-eliminarclave [args ...]

– **sfmemoria -crear id_mem [tam]**. Crea un sistema de ficheros en la zona de memoria designada con el identificador *id_mem*, el tamaño total del sistema de ficheros es *tam* (incluyendo las estructuras de control del sistema de ficheros). Si id_mem representa una zona de memoria que ya existe no ha de especificarse *tam* y, en ese caso, *sfmemoria -crear* equivaldrá a formatear es sistema de ficheros. Si se indica *tam* debemos suponer que id_mem es una clave que no está en uso y se creará una nueva zona de memoria compartida.

  * **ACLARACION SOBRE id_mem:** *id_mem* puede representar una dirección de memoria (donde haya mapeada memoria compartida o de las obtenidas con *memoria -malloc*) o una clave. Para distinguir si es una dirección o una clave se procede de la siguiente manera: se supone que es una dirección y se comprueba si está en la lista de direcciones de memoria que tiene el proceso; si está, es una dirección válida y se usa; si no está, se supone que es una clave y se vuelve a obtener un identificador con *shmget* y una dirección con *shmat* y se añade a la lista (evidentemente esto puede suponer tener mapeada la misma zona de memoria compartida en varios sitios). **IMPORTANTE: SI SE SUMINISTRA UNA CLAVE DEBE VOLVER A MAPEARSE LA ZONA DE MEMORIA**.

    Tambien debe tenerse en cuenta que un puntero a caracter no se almacena en la direccion de memoria representada por la cadena por él apuntada. (`char *p="0x004b0000"` no quiere decir que la cadena apuntada por p esté en la dirección de memoria 0x004b0000). Un posible código para obtener una dirección a partir de un identificador pordría ser:

```
void * ObtenerMemoria (char * idmem, int tam)
{
  void * p;
```

```
  key_t cl;

  p=(void *) strtoull(idmem,NULL,16);

  if (EstaEnListaShareds(p) || EstaEnListaMallocs (p))
   return p;
  cl=(key_t) strtoul (idmem,NULL,10);
  return (ObtenerMemoriaLlamandoShmget(cl,tam));
}
```

Nótese que al no especificar el tamaño de la zona suponemos que ya existe y el tamaño es el que ya tenía, en este caso id_mem puede tratarse de una clave o de una dirección, y si es una dirección podría ser de las de memoria compartida o de las obtenidas mediante *memoria -malloc*. Si se especifica el tamaño, id_mem es una clave y se intentrá crear (y maperar) una zona de memoria compartida nueva, dando error si ya existe. Téngase en cuenta que los sistemas de ficheros en memoria compartida deben poder ser accedidos desde otra instancia del shell en ejecución.

– **sfmemoria -copiar id_fich1 id_fich2** Copia el fichero identificado por id_fich1. La copia es id_fich2.

  * **ACLARACION SOBRE id_fich:** La identificación de un fichero se supone que es de la forma `nombre_fichero@id_mem`, donde ***id_mem*** **es un identificador de memoria tal como acaba de describirse en el apartado anterior** y *nombre_fichero* es el nombre del fichero en el sistema de ficheros representado por *id_mem*. Si no se especifica *id_mem* o id_mem es "disco" se entiende que es un fichero de disco.
    Ejemplos p1.c@10; ficherillo.txt@0xbf008000, p4.c@disco prueba.txt...

– **sfmemoria -mover id_fich1 id_fich2** Mueve el fichero identificado por id_fich1 a id_fich2. *id_fich* debe entenderse como en los apartados anteriores

– **sfmemoria -borrar id_fich** Elimina el fichero identificado por id_fich.*id_fich* debe entenderse como en los apartados anteriores.

– **sfmemoria -listar id_mem** Lista el sistema de ficheros identificado por id_mem. Para el sistema de ficheros debe listar: fecha de creación o último formateo, tamaño total del sistema de ficheros, tamaño total para los datos de los ficheros, número de ficheros y espacio disponible en ese instante en ese sistema de ficheros, y para cada fichero nombre, tamaño e instante de creación. ***id_mem*** **es un identificador de memoria tal como acaba de describirse**

en un apartado anterior.

  – **sfmemoria -eliminarclave key** Elimina la zona de memoria compartida de clave *key*. NO HAY QUE DESMAPEAR NADA: es simplememte una llamada a *shmctl(id, IPC_RMID. . .)* con el identificador adecuado

Información detallada de las llamadas al sistema y las funciones de la librería debe obternerse con man (shmget, shmat, mmap, munmap . . .).

FORMA DE ENTREGA

Va a ser utilizado el servicio de recogida de prácticas suministrado por el Centro de Cálculo de esta Facultad y parte del proceso de corrección de las prácticas va a ser automtíco (compilación, listado de practicas entregadas etc) por lo cual deben entregarse **exactamente** como se indica a continuación:

- Se colocará el código fuente de la práctica en el directorio asignado para ello antes de la fecha tope de entrega de la práctica.
- Se entregará UN SOLO fichero fuente por práctica, de nombre pN.c (N el número de práctica). Por ejemplo, para esta práctica será p1.c (en minúsculas).
- Los grupos de prácticas son de **2 (DOS)** alumnos. La práctica SOLO DEBE SER ENTREGADA POR UNO DE LOS MIEMBROS DEL GRUPO
- en el código fuente de la práctica debe figurar como comentario el nombre de los autores **exactamente** en el siguiente formato

  ```
  /*
  AUTOR:apellido11 apellido12, nombre1:login_en_el_que_se_entrega
  AUTOR:apellido21 apellido22, nombre2:login_en_el_que_se_entrega
  */
  ```

  donde:

  1. La palabra autor aparece en mayúsculas.
  2. Los apellidos y el nombre de los autores están totalmente en minúsculas.
  3. apellidoij representa el apellidoj del componente i del grupo de prácticas.
  4. No hay espacios antes y despues de los dos puntos.
  5. El login que aparece es el del que entrega la práctica (aparece el mismo login en las dos líneas).
  6. Los símbolos de comentarios están en líneas distintas.

7. No debe incluirse la letra ñ ni vocales acentuadas en los nombres

FECHA DE ENTREGA Viernes 10 Noviembre de 2011