SISTEMAS OPERATIVOS II

Tercer curso Ingeniería Informática. Curso 2008-2009

Práctica 1: Procesos en UNIX. Variables de entorno. Memoria.

Comenzar la codificación de un intérprete de comandos (shell) en UNIX, que se irá completando en suscesivas prácticas. Hay que tener en cuenta que

- 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.

En esta primera práctica, el intérprete de comandos, además de algunos comandos de uso general nos permitirá:

- Manipular el entorno del shell, a través del tercer argumento de *main*, la variable de entorno *environ* y las funciones de librerá *getenv* y *putenv*
- asignar y desasignar memoria dentro del shell, asi como inspeccionar sus contenidos
- mapear y desmapear ficheros en memoria así como leer y/o escribir ficheros a/desde la memoria del proceso

- el shell mantendrá la contabilidad (mendiante una lista) de todas las direcciones de memoria obtenidas mediante los comandos *mmap* y *malloc*, de manera que en cualquier momento (mediante los comandos malloc, mmap, o mem) podamos conocer cuales son estas direcciones de memoria. Los comandos *munmap* y *free* permiten, en su caso, eliminar una dirección de memoria de esta contabilidad. Para las direcciones obtenidas con el comando *malloc* se infromará, además de la dirección, del tamaño de la zona; para las obtenidas con *mmap*, además de la dirección debe informarse del nombre del fichero mapeado y del tamaño del mapeo
- el shell implementará el concepto de PATH (no se usará ni *execlp ni execvp* para la ejecución de programas), para lo cual debe mantener una lista (de implementación libre) de directorios donde buscará los ejecutables si no se suministra la ruta completa hasta ellos. El comando *path* manipula esta lista.

Los comandos que incluirá el intérprete de comandos en esta primera práctica son

quit Termina la ejecución del intérprete de comandos.

exit Termina la ejecución del intérprete de comandos.

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

cd [dir] Cambia el directorio actual a *dir*. Si no se le suministra *dir* vuelve al directorio donde estaba el shell cuando comenzó la ejecución.

pwd Muestra el directorio actual. (NO ES LISTAR LOS FICHEROS: ES DECIR EN QUE DIRECTORIO SE ENCUENTRA EL SHELL)

prompt *prompt* Cambia el indicador (prompt) del intérprete de comandos.

getpid Muestra el pid del proceso y tambien el de su proceso padre

environ **[-env] [-main] [-dirs]**] Muestra la lista de todas las variables de entorno accediendo mediante el tercer argumento de *main()* (`-main`) o mediante la variable externa *environ* (`-env`). Para cada variable de entorno muestra, en una misma línea, la variable (nombre=valor), la dirección de memoria donde se almacena la variable (el valor de la variable e[i] como puntero) y la dirección de memoria donde se almacena el puntero (&e[i]). Si se especifica `-dirs` muestra el valor (como punteros) del tercer argumento de main y de *environ* así como las direcciones de memoria donde están almacenados el tercer argumento de main y environ. Ejemplo

```
#entorno -main -dirs
......
.......
.......
ffbefcb8->env[29]=(ffbeff86) PATH=/usr/bin:/usr/local/bin:
ffbefcbc->env[30]=(ffbeffa5) LC_MONETARY=es
ffbefcc0->env[31]=(ffbeffb4) LC_COLLATE=es
ffbefcc4->env[32]=(ffbeffc2) _INIT_NET_STRATEGY=none

20a34->environ=ffbefc44 | ffbefc24->env=ffbefc44  | ffbefc20->argv=ffbefc3c
#
```

getenv **[-main] [-env] [-func] var1, var2 ...** Devuelve los valores de las variables de entorno var1, var2 .. obtenidos mediante el tercer argumento de *main*, *environ* y la función de librería *getenv*. Para cada caso imprime además la dirección de memoria donde está la variable (su valor como puntero) y la dirección donde se almacena el puntero (salvo en el caso de obtenerla con *getenv*). Si no se especifica -env, -main o -func mostrará todos. ejemplo

```
#getenv -main -env TERM
mediante arg main 0xbfd619f6: TERM=xterm (&0xbfd5f8e0)
mediante environ 0xbfd619f6: TERM=xterm (&0xbfd5f8e0)
#getenv TERM
mediante arg main 0xbfd619f6: TERM=xterm (&0xbfd5f8e0)
mediante environ 0xbfd619f6: TERM=xterm (&0xbfd5f8e0)
mediante getenv 0xbfd619fb->xterm
#
```

putenv **-main|-env|-func var valor** Establece el valor de la variable de entorno *var* a *valor* accediendo a ella mediante *environ* (-env), el tercer argumento de main (-main) o la función de librería *putenv*. Si la variable no existe no la creará (informará de ello), salvo que se haya especificado que se haga mediante *puntenv*, entonces será creada.

susenv **-main|-env var nue=val** Sustituye de la variable de entorno *var* por *nue=val* accediendo a ella mediante *environ* (-env) o el tercer argumento de main (-main). Si la variable *var* no existe informará de ello.

delenv -main|-env var . Elimina la variable var del entorno especificado.

fork El intérprete de comandos crea un hijo con fork() y se queda (el propio intérprete de comandos) en espera hasta que dicho hijo (el creado con fork) termine.

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);
}
```

malloc [tamano] asigna en el shell la cantidad de memoria que se le especifica (utilizando la función *malloc*). y nos informa de la direccion de la memoria asignada. Si no se especifica tamaño nos dará una lista de las direcciones de memoria asignadas **con el comando malloc**

free [[-t] dir] libera la memoria asignada a dir (usando la función de librerá *free*). Si la memoria no fue asignada con el comando previo *malloc* nos dará un aviso y no la liberará. Si se especifica -t, *dir* indica un tamaño e intentará desasignar (en caso de ser posible) un bloque de ese tamaño. Si no se especifica *dir* nos dará una lista de las direcciones de memoria asignadas **con el comando malloc**

display dir [cont] Mustra 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 in espacio en blanco) y su valor en hexadecimal. Imprime 25 bytes por línea. (podría producir error)

read file dir Copia el fichero *file* en la dirección de memoria *dir* (podría producir error)

write [-f] file dir cont Copia desde la dirección de memoria *dir cont* bytes en el fichero *file*. -f especifica que se sobreesciba el fichero (si existe) (podría producir error)

stat fich Nos da información del fichero *fich* (nombre, tamaño, permisos, fecha de ultimo acceso)

delete fich Elimina el fichero *fich*

list [-l][dir] Lista los ficheros del directorio *dir*. Si no se especifica dir se entiende que es el directorio actual. -l indica que han de darse tambien los detalles de los ficheros (tamanño, permisos, . . .)

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*

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*

path [op] [dir |file] Manipula y/o muestra la ruta de bśqueda de nuestro intérprete de comandos. Los directorios de esta lista se especificarán sin el caracter '/' al final (salvo el directorio raíz "/"). Se admiten directorios relativos (p.e. "../bin") en esta lista y NO DEBEN convertirse en absolutos. *op* puede ser *-add, -del, -query, -init, -path o -find* Ejemplos

```
#path -add /usr/local/bin /* correctos */
#path -del  /
#path -query .
```

path -add [dir] Añade *dir* a la ruta de búsqueda.

path -del [dir] Elimina *dir* de la ruta de búsqueda.

path -query [dir] Indica si *dir* está en la ruta de búsqueda.

path -init Vacía la ruta de búsqueda.

path -path Añade los directorios de la variable de entorno PATH a la ruta de búsqueda.

path Muestra la ruta de búsqueda.

path -find [file] Si se le indica una trayectoria completa (empezando por ./, por / o por ../) indicará si dicho fichero existe. Si no se especifica trayectoria completa lo busca *file* en la lista de directorios que constituyen la ruta de búsqueda del intérprete de comandos y devuelve la trayectoria completa hasta él. (es TOTALMENTE análogo al comando *which* del sistema, véase el ejemplo a continuación). (NOTA: muy reutilizable para el comando *exec* y la ejecución en primer o segundo plano)
Si no se especifica *dir* o *file* en *-add, -del, -query* o *-find*, se mostrará la ruta de búsqueda.

```
#path -add /usr/openwin/bin
#path -find xterm
/usr/openwin/bin/xterm
#path -find xternp
 xternp: no encontrado
#path -find ./aa.out
 ./aa.out: no encontrado
#path -find ./a.out
 ./a.out
```

exec [LISTAVAR] cmd [args] Ejecuta, sin crear proceso, el programa especificado por cmd con los argumentos especificados por args. Si no se especifica LISTAVAR la ejecución será mediante *execv*. LISTAVAR representa una lista de nombres de variables de entorno, y en caso de especificarse la ejecución será mediante *execve* (los valores se obtendrán mediante *environ*. Si LISTAVAR es "NULLENV" (sin las comillas) la ejecicuón será mediante *execve* pero en un entorno vacio. Si *cmd* es un nombre que comienza por /, ./ o ../ debe entenderse como el nombre absoluto del ejecutable, en otro caso se buscará dicho ejecutable en la lista de directorios con la que el shell implementa el concepto de *path* ejemplo

```
# exec /bin/ls -l
# exec /usr/bin/xterm -e csh
# exec TERM HOME DISPLAY HZ /usr/bin/xterm -e bash -T prueba
# exec ls
# exec ./a.out
# exec NULLENV ./a.out argu_uno
```

- **[LISTAVAR] cmd [args]** Exactamente igual al anterior salvo que ahora la ejecución la hace un proceso hijo del shell (Es decir, el shell crea un proceso y dicho proceso hijo es el que ejecuta lo que se especifica por *[LISTAVAR] cmd [args]*. La ejecución es en primer plano (el shell espera a que el hijo termine).

Información detallada de las llamadas al sistema y las funciones de la librería debe obternerse con man (stat, chdir, getcwd, opendir, malloc, free, mmap, munmap, getenv, putenv, fork, exec . . .)

Salvo que se diga explicitamente, en NINGUN CASO la práctica puede producir error en tiempo de ejecución.

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 3 Abril 2009