SISTEMAS OPERATIVOS

Grado en Informática. Curso 2012-2013

Práctica 3: Procesos: memoria y credenciales

CONTINUAR la codificación de un intérprete de comandos (shell) en UNIX. 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.
- 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** (puede utilizarse *valgrind* para detectar errores de memoria)
- 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).
- El shell leerá de su entrada estándar y escribirá en su salida estándar, de manera que podría (quizá en una práctica futura) ser ejecutado un archivo de comandos invocando al shell con su entrada estándar redireccionada a dicho archivo.

El shell llevar una lista de direcciones de las memoria que se asigna con los comandos *malloc, mmap y shared*. Para cada dirección de memoria que se asigne con uno de esos comandos ha de guardar: la dirección de memoria, el tamaño asignado, el instante de asignación, el tipo de asignación (*malloc, mmap o shared*), e información dependeiente del tipo de asignación (clave e id si es con *shared*, nombre y el descriptor del fichero si es con *mmap* . . . ...) La implementación de dicha lista es TOTALMENTE LIBRE (una lista, tres

listas, con arrays, con punteros, con arrays de punteros ...).

Debe comprobarse que los valores almacenados en la lista SON COHERENTES con la salida del comando **pmap** para dicho proceso

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 nos dará una lista de las direcciones de memoria asignadas **con el comando malloc**

free [**tam**] Desasigna (con *free*) en el shell el bloque de tamaño *tam* aisgnado mediante el comando *malloc* y lo elimina de la lista. Si no se especifica tamaño nos dará una lista de las direcciones de memoria asignadas **con el comando malloc**

mmap fichero [**-s**] [**perm**] 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. *perm* representa los permisos del mapeo (en formato *rwx*). Si se especifica *-s* el mapeo ha de ser tipo *shared*; en caso contrario *private*. Además guardará esa dirección, junto con el tamaño, el nombre y el descriptor 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, el instante en que se mapeó y el descriptor de fichero.

munmap [**fich**] Desmapea el fichero *fich* 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, el instante en que se mapeó y el descriptor de fichero.

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 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*, en este caso se usa tamaño 0): **salvo error, este comando siempre realiza un nuevo mapeo: NO DEBE BUSCARSE EN LA LISTA POR CLAVE**. 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. UNA MISMA ZONA DE MEMORIA COMPARTIDA PUEDE ESTAR MAPEADA MAS DENA VEZ

unshared [**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. Si no se especifica *tam* 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.

rmkey **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

mem [**-malloc|-shared|-mmap**]. Muestra la lista de direcciones de memoria asignadas con el comando *malloc*, *shared* o *mmap*. Si no se especifica argumente nos muestra todas. Para cada dirección muestra ademas los detalles: tipo de dirección, instante de asignación, tamaño ...

memdump ***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

```
->memdump 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 [n]. Invoca a la función recursiva *n* veces. La función recursiva recibe un parámetro que indica el número de veces que se tiene que invocar. Además esta función tiene 2 variables: un array automatico de 512 caracteres y un array estático de 512 caracteres. Esta función debe imprimir

- el valor del parámetro que recibe así como la dirección de memoria donde se almacena dicho parámetro.
- la dirección donde se almacena el array automático: el nombre del

array como puntero.

- la dirección donde se almacena el array estático: el nombre del array como puntero.

Un posible código para dicha función podría ser:

```
void recursiva (int n)
{

char automatico[TAMANO];
static char estatico[TAMANO];


printf ("parametro n:%d en %p\n",n,&n);

printf ("array estatico en:%p \n",estatico);

printf ("array automatico en %p\n",automatico);

if (n>0)
   recursiva(n-1);

}
```

uid [-l] [u] Muestra o cambia la credencial efectiva de usuario del proceso a u. Si se especifica -l, u representa un login, en caso contrario es el valor numérico de la credencial. En caso de no poder cambiarla indicará con un mensaje de error el motivo. Si no se especifica ni *-l* ni *u*, *uid* muestra las credenciales de usuario del proceso. Para cada credencial muestra el número, el login asociado y en nombre real del usuario

Información detallada de las llamadas al sistema y las funciones de la librería puede (y debe) obternerse con man (mmap, shmget, shmat, shmdt, shmctl, munmap, malloc, free, setuid, getuid, geteuid, getpwuid. . . .

FORMA DE ENTREGA

Las prácticas se entregarán mediante el repositorio de Subversion bajo el directorio P3 antes de proceder a su defensa. Esta carpeta deberá incluir tanto el código fuente como el fichero Makefile (en caso de usarse) que permite su compilación. Las cabeceras de los ficheros fuente deben incluir un comentario con los nombres de los integrantes del grupo de prácticas y el horario en el que están apuntados.

La práctica será entregada y defendida ante el profesor en el aula de prácticas.

Todos los miembros del grupo deberán estar presentes para la entrega, de forma que el profesor pueda revisar su funcionamiento así como realizar comentarios/cuestiones a los integrantes del grupo o pedir cambios en el código que se puedan considerar pertinentes.

Las prácticas entregadas que posteriormente no se defiendan con solvencia podrán implicar un `No apto` para todos los miembros del grupo

FECHA LIMITE DE ENTREGA 23-XII-2012

AYUDAS

Se adjunta la función para asignar memoria compartida, y la implementación de los comandos *mmap* y *rmkey*. Se supone que las funciones GuardarDireccionShared y GuardarDireccionMmap son las que guardan las direcciones en la lista.

```
void * ObtenerMemoriaShmget (key_t clave, off_t tam)
{
 void * p;
 int aux,id,flags=0777;
 struct shmid_ds s;

 if (tam)
   flags=flags | IPC_CREAT | IPC_EXCL;

 if (clave==IPC_PRIVATE)   /*no nos vale*/
        {errno=EINVAL; return NULL;}

 if ((id=shmget(clave, tam, flags))==-1)
        return (NULL);

 if ((p=shmat(id,NULL,0))==(void*) -1){
        aux=errno;
        if (tam)
                shmctl(id,IPC_RMID,NULL);
        errno=aux;
        return (NULL);
        }
 shmctl (id,IPC_STAT,&s);

 GuardarDireccionShared(p, s.shm_segsz, clave);
 return (p);
}
```

```
void Cmd_Mmap (char *fichero, char *tipomap, char * perm)
{
   void *p;
   struct stat s;
   int df, protection=0,map=MAP_PRIVATE,modo=O_RDONLY;

   if (fichero==NULL)
       { ListarDirecciones(DIRMMAP); return;}
   if (tipomap!=NULL){
          if (!strcmp(tipomap,"-s"))  map=MAP_SHARED;
          if (perm!=NULL && strlen(perm)<4) {
                protection=0;
                if (strchr(perm,'r')!=NULL) protection|=PROT_READ;
                if (strchr(perm,'w')!=NULL) protection|=PROT_WRITE;
                if (strchr(perm,'x')!=NULL) protection|=PROT_EXEC;
                }
   }
   if (protection&PROT_WRITE)  modo=O_RDWR;
   if (stat(fichero,&s)==-1 || (df=open(fichero, modo))==-1)
        { perror ("imposible acceder al fichero"); return; }
   if ((p=mmap (NULL,s.st_size, protection,map,df,0))==MAP_FAILED)
        { perror ("error mmap");close(df);  return; }
   close(df);
   printf ("fichero %s mapeado en %p\n", fichero, p);
   GuardarDireccionMmap (p, s.st_size,fichero,df);
}

void Cmd_rmkey (char * key)
{
  key_t clave;
  int id;

  if (key==NULL || (clave=(key_t) strtoul(key,NULL,10))==IPC_PRIVATE){
        printf ("   rmkey  clave_valida\n");
        return;
        }
  if ((id=shmget(clave,0,0666))==-1){
        perror ("shmget: imposible obtener memoria compartida");
        return;
        }
  if (shmctl(id,IPC_RMID,NULL)==-1)
        perror ("shmctl: imposible eliminar memoria compartida\n");
}
```

Para probar el funcionamiento de las credenciales

1. Comprobar que los directorios HOME de los usuarios tienen permisos `rwx------`
2. Compilar el programa y colocar el ejecutable en `/tmp`
3. Poner al ejecutable permisos `rwsr-xr-x` (chmod 4755 a.out)
4. Entrar como otro usuario (distinto del root) y ejecutar el programa
   - comprobar que credenciales muestra
   - comprobar que cambios permite en las credenciales
   - comprobar a que directorio HOME se puede acceder segun el valor de las credenciales