Hoja de Ejercicios 4. Memoria Virtual

Ejercicios Prácticos

Los ejercicios marcados con el icono 💻 son prácticos y deben realizarse en el laboratorio. Los ejercicios prácticos de esta hoja requieren el entorno de usuario básico (shell) y de desarrollo (compilador, editores y depurador). Además algunos ejercicios necesitan acceso de superusuario que está disponible en las máquinas virtuales de la asignatura.

💻 ***Ejercicio 1.*** Compila el siguiente programa:

```
#include <stdio.h>
#include <unistd.h>

int num = 22;
int mul;

const char *msg = "El resultado es:\n";

int main(void) {
    static int factor = 2;

    mul = num * factor;
    printf("%s%i\n", msg, mul);

    sleep(600);

    return 0;
}
```

Consulta los segmentos definidos en el ejecutable generado con el comando `readelf -l` y completa la siguiente tabla.

- Los segmentos de la tabla corresponden a las secciones de tipo LOAD. La primera de ellas PHDR es la 00, INTERP la 01 y así sucesivamente.
- Algunas secciones del programa pueden estar asignadas al mismo segmento del ejecutable.

Segmento	Código C correspondiente	Offset	Dir.virtual	Flags
.text				
.rodata				
.bss				
.data				

Ejecutar ahora el programa y obtener los segmentos de memoria virtual accediendo al fichero `maps` del directorio del proceso en `/proc`. **Nota**: Se puede identificar la correspondencia de los

segmentos del ejecutable y los segmentos de memoria virtual comparando el camp offset y los flags:

Segmento	**Direcciones virtuales**	**Offset** del fichero	**Flags**	**Tipo** map/anonimo	**Ruta** del fichero
`.text`					
`.rodata`					
`.bss`					
`.data`					
`[heap]`					
`[stack]`					

Nota: Alternativamente, el programa pmap(1) puede utilizarse para consultar la información sobre los segmentos de memoria (misma información que `proc/<pid>/maps`)

Cuestiones:

- En qué segmento(s) de memoria virtual está la cadena `"El resultado es:\n"`. Los contenido del segmento de memoria virtual se puede acceder en el directorio del proceso `/proc/<pid proceso>/map_files/<rango del segmento>`. Comprobar que la respuesta es correcta con el comando `strings(1)`.
- En la salida del comando readelf se muestra el inicio del programa (entry point). ¿En qué segmento del fichero está ubicado? ¿Y en qué segmento de memoria virtual?
- Dibuja esquemáticamente (como en las transparencias de teoría) el espacio de direcciones del proceso con las regiones correspondientes.
- Investiga cuál es el contenido y propósito de las áreas de memoria marcadas como [vdso] y [vvar].

Ejercicio 2. Considera el siguiente programa y completa la tabla.

```
#define CONSTANT 10
int num1 = CONSTANT;
int num2;

int main(int argc, char *argv[]) {
    int *i = malloc((sizeof(int));

    num2 = argc;

    for (*i=0; *i<CONSTANT; *i=*i+1) {
        fprintf(stdout, "%s: %d, argc: %d\n", string, num1--, num2);
    }

    return 0;
}
```

Símbolo	Espacio en ejecutable (Sí/No)	Segmento
CONSTANT		
i		
num1		
num2		

💻 ***Ejercicio 3.*** Escribe un programa que cree una región de memoria (`mmap(2)`) con las siguientes características:

- Tamaño 1024 bytes
- Acceso privado
- Modo lectura y escritura

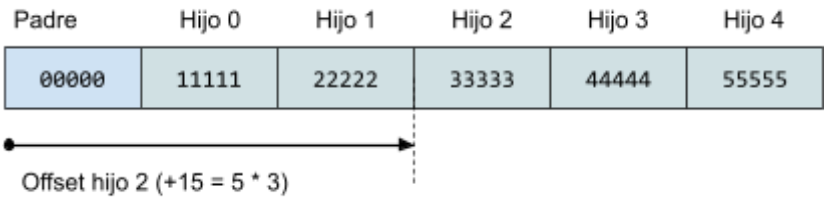
Una vez creada la región de memoria el proceso la inicializará con '\0' (usar un bucle o la llamada memset(3)), mostrará la dirección del segmento (modificador de formato `%p`) y el `PID` del proceso. Finalmente el proceso hará un sleep de 600s. Ejemplo de ejecución:

```
$ ./mimap
PID:2716 Dirección del segmento: 0x7fe4c56ff000
```

Usando el PID consultar el fichero maps e identificar el segmento creado en el espacio de memoria del proceso:

Dirección Inicial	Dirección FInal	Offset	Flags

💻 ***Ejercicio 4.*** Re-escribir el ejercicio 11 de la Hoja 3 proyectando el fichero de salida (`output.txt`) en la memoria virtual:



El esquema es el siguiente:

- **Preparación del segmento de memoria.** Cuando se proyecta un fichero debe tener al menos el tamaño de la región que se quiere crear, y se debe abrir con el mismo modo que se quiere crear la región (lectura, escritura,...). Las acciones que realizará el proceso padre son:
  - Crear el fichero de salida con la llamada `open(2)` y las opciones `O_TRUNC` y lectura/escritura.
  - Para fijar el tamaño de la región usaremos la llamada al sistema `ftruncate(2)` (fichero *sparse*)
  - Proyectará en la memoria virtual del proceso con (`mmap(2)`). Nota: convertir el

puntero retornado por map() a char *
    - Cerrar el descriptor (`close(2)`), esto no afecta al segmento de memoria proyectado.

- **Inicialización del segmento de memoria.** Se crearán los hijos que usarán su identificador como base del desplazamiento a la zona de memoria y fijará en cada posición el caracter correspondiente. *Truco*: se puede usar el operador + en variables de tipo char para movernos por la tabla de caracteres, por ejemplo: `(char) '5' = (char) '0' + (int) 5`

- **Finalización.** El proceso padre esperará la finalización de todos los hijos y eliminará la región con las llamadas msync(2) y munmap(2).

Ejercicio 5. Compara la implementación del Ejercicio 4 y la realizada en el Ejercicio 11 de la Hoja 3:

- Respecto a E/S, describe qué acciones realiza el sistema. ¿Hay alguna diferencia entre ambas alternativas (e.g. número de escrituras en disco…)?. Nota: considera todos los componentes del VFS y memoria virtual.

- Respecto al SO, ¿cuál de las dos alternativas es preferible? ¿Y desde el punto de vista del programador?

Ejercicio 6. Considera un sistema operativo con una memoria virtual paginada de un nivel y tamaño de página de 512 palabras. El espacio de direcciones virtuales tiene 512 páginas y la memoria física tiene 10 marcos de página.

- Describe la estructura del espacio de direcciones virtuales y física, y de la tabla de páginas que usaría el sistema operativo. Indica el tamaño de todos los campos.

- Si el contenido de la memoria física es el que se muestra a continuación, determina el contenido de la tabla de páginas del proceso. ¿Qué dirección física corresponde con las direcciones virtuales 0x13FF y 0x1403?

0x0000	
0x0200	
0x0400	
0x0600	Página virtual 34
0x0800	Página virtual 9
0x0a00	
0x0c00	
0x0e00	Página virtual 65
0x1000	
0x1200	Página virtual 10

Contenido de la memoria

- ¿Qué ocurre si un proceso intenta acceder a la posición 0x80E8?

- Supongamos que el marco de página en las direcciones 0x0800-0x09ff se quiere compartir con otro proceso. ¿Debería asignarse al mismo segmento virtual?

💻 ***Ejercicio 7.*** Considera el siguiente programa:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int factor    = atoi(argv[1]);
    int elementos = factor * 1024;

    printf("Inicializando región de memoria\n");

    int * ptr = (int *) malloc(elementos * sizeof(int));

    for (size_t i = 0; i < elementos; i++){
        ptr[i] = 1;
    }

    sleep(600);

    return 0;
}
```

Realiza dos ejecuciones con el comando strace para identificar cómo se reserva la memoria del array `ptr` y consulta el mapa de memoria del proceso en `/proc` para completar la siguiente tabla. **Nota:** lee la sección NOTES de la página de manual de `malloc(3)`.

- **Caso A.** `strace ./ejercicio7 1`
- **Caso B.** `strace ./ejercicio7 1024`

Caso	Dirección Inicial	Dirección FInal	Flags	Segmento	Mecanismo Memoria Dinámica
A					
B					

Ejercicio 8. Un sistema tiene un uso medio de la CPU del 15% (usuario) y 3% (sistema), el área de swap está ocupada al 92%. La salida del comando ps muestra 50 procesos en estado “D” (espera no interrumpible) . ¿Cuál de estas acciones aumentará más la utilización de la CPU?:

- Ampliar la memoria principal
- Ejecutar más programas para aumentar el grado de multiprogramación
- Aumentar el área de swap
- Añadir más CPU’s

Ejercicio 9. En un sistema con memoria virtual paginada indicar las acciones son realizadas por el sistema operativo (especificando a qué estructuras de datos accede y como las modifica) en los siguiente casos:

- un proceso intenta escribir en una página de solo lectura
- un proceso intenta acceder a una dirección correspondiente a una página que no está en memoria.

Ejercicio 10. Considera el siguiente código e indique razonadamente si cada una de las afirmaciones posteriores son ciertas o falsas. Asuma que todos los segmentos de memoria del proceso asociado (texto, datos, pila,...) ocupa una página de 4 Kb.

```
int main() {
    int i;
    int M[128];
    int x,y;

    x = M[0];
    y = 0;

    for (i=1; i < 200; i++) {
        y = x + M[i];
        M[i] = y;
    }
    return 0;
}
```

- El proceso provocará una excepción por violación de segmento (SIGSEV) al ejecutarlo porque accede a elementos de M no reservados.
- La ejecución del proceso corromperá los datos de la pila.
- La ejecución del proceso puede corromper el segmento de código del programa.
- Si se modifica el bucle para que recorra 4096 posiciones del array M se producirá una excepción.