Universidad Nacional Experimental Del Táchira
Departamento De Ingeniería En Informática
Sistemas Operativos Cod. 0425603t
Laboratorio 5 – Gestión de Memoria

Gestión de memoria en Linux

El principal objetivo de este laboratorio es comprender el funcionamiento básico de las memoria, como acceder a las regiones de memoria y compartir variables.

PARTE I. Consulta de las regiones de memoria

En los sistemas Linux se puede consultar la información que maneja el núcleo del Sistema Operativo accediendo al directorio "/proc". Para la realización de esta práctica se consultará dicha información, a través del fichero "maps" asociado a cada proceso, ejecute en un terminal la siguiente línea que mostrará el contenido del archivo comentado anteriormente:

```
$ cat /proc/$$/maps
```

La salida está compuesta por la siguiente información:

- **Dirección**: El inicio y fin de la región en el espacio de memoria del proceso.
- **Permisos**: Describe como se puede acceder a la información almacenada, posee los permisos de: (r) lectura, (w) escritura, (x) ejecución, (p) si es privada o (s) si es compartida. Si se accede a la memoria rompiendo los permisos, ocurre una Violación de Segmento. Se puede utilizar la llamada del sistema **`mprotect`** para cambiar los permisos asignados.
- **Desplazamiento**: Si la región fue mapeada desde un archivo (usando **`mmap),`** este valor es el desplazamiento de donde comieza el segmento de memoria. De lo contrario es solo 0.
- **Dispositivo**: Si la región fue mapeada desde un archivo, se representa en hexadecimal la dirección donde el archivo yace.
- **Inodo**: Si la región fue mapeada desde un archivo, este es el número del mismo.

- **Ruta**: Si la región fue mapeada desde un archivo es el nombre del archivo. Existen estructuras especiales como `[heap]`, `[stack]`, o `[vdso]`. `[vdso]` utilizadas para objetos compartidos.

PARTE II. Uso global de la memoria

En una nueva terminal, ejecute el siguiente comando y analice la salida:

```
$ free -t
```

Luego proceda e ejecutar muchas aplicaciones, abra el navegador y entre a varias páginas, abra editores de texto, juegos, varias terminales con la finalidad de incrementar el uso de memoria RAM. Ahora vuelva a ejecutar el comando anterior y analice que observa.

PARTE III.

Mecanismos de Comunicación Inter-procesos (IPC)

En POSIX podemos crear, manipular y eliminar segmentos de memoria para mapearla al espacio de direcciones de un proceso, para ello podemos utilizar los servicios **shmat:**

```
//Creación y obtención del identificador de un segmento de memoria:
int shmget(key_t clave, int tamano, int opcion)

//Conexión de un segmento al espacio de direcciones del proceso
char *shmat(int shmid, char *adr, int opcion)

int shmdt(char *adr) //Desconexión de un segmento previamente conectado

int shmctl(int shmid, int cmd, struct shmid_ds *buf) //Operaciones de control
```

Ejemplo en lenguaje C para una variable entera compartida en GNU/Linux

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define CLAVE 231235123

int main(){
  int *p, resp = 1, valor;

  //Crea la variable compartida
  int varComp = shmget((key_t) CLAVE, sizeof(int), IPC_CREAT|0666);

  if(varComp == -1)
    return -1;

  while(resp){
    printf("Menu\n");
    printf("1.Leer\n");
    printf("2.Escribir\n");
    printf("0.Salir\n");

    scanf("%d",&resp);

    switch(resp){
      case 1:
          p = (int *) shmat(varComp,NULL,0);
          valor = *p;
          shmdt((char *) p);
          printf("El valor actual es %d\n\n", valor);
          break;
      case 2:
          printf("\n\nIngrese valor\n");
          scanf("%d",&valor);
          p = (int *) shmat(varComp,NULL,0);
          *p = valor;
          shmdt((char *) p);
          break;
    }
  }
}
```

Ejecute 2 instancias del programa, realice acciones de escritura y lectura en ambos y observe como la información es compartida entre ambos procesos.

Ejemplo en lenguaje C++ para una variable entera compartida en Windows

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <tchar.h>
#define LARGO 10

bool validarComparticion(HANDLE manejador){
   if ( manejador == NULL){
      _tprintf(TEXT("Could not create file mapping object (%d).\n"),GetLastError());
      return false;
   }
   return true;
}

int main(){
      int resp = 1;
      LPCTSTR p;
      TCHAR clave[] = TEXT("Local\\MiPagina"), valor[LARGO];
      HANDLE varComp;

      while(resp){
         printf("Menu\n");
         printf("1.Leer\n");
         printf("2.Escribir\n");
         printf("0.Salir\n");
         scanf("%d",&resp);

         switch(resp){
           case 1:
                  varComp = OpenFileMapping( FILE_MAP_ALL_ACCESS, FALSE, clave);
                  if(validarComparticion(varComp)){
                        p = (LPTSTR) MapViewOfFile(varComp, FILE_MAP_READ, 0, 0, sizeof(TCHAR) *
LARGO);
                           printf("El valor actual es %s\n\n", p);
                           UnmapViewOfFile(p);
                   }
                  break;
            case 2:
                  varComp = CreateFileMapping( INVALID_HANDLE_VALUE,    // Especificacion de página
                           NULL,                  // Atributos de seguridad
                           PAGE_READWRITE,         // Permiso de escritura y lectura
                           0,
                           sizeof(int),              // tamaño  máximo
                           clave
                  );
                  if(validarComparticion(varComp)){
                           p = (LPTSTR) MapViewOfFile(varComp, FILE_MAP_ALL_ACCESS, 0, 0, sizeof(int));
                           printf("\n\nIngrese valor\n");
                           scanf("%s",&valor);
                           CopyMemory((PVOID) p, valor, sizeof(TCHAR) * LARGO);
                           UnmapViewOfFile(p);
                  }
                  break;
            }
         }
         CloseHandle(varComp);
         ExitProcess(0);
}
```

PARTE III. Asignación

1. En base al ejemplo anterior, desarrolle un programa de chatear, donde al iniciar el programa pregunta el nombre de la sala a entrar, este nombre se utilizará como **llave para la solicitud de memoria compartida**, de esta forma, si 2 o mas usuarios (procesos) se “conectan” a la misma sala, podrán intercambiar mensajes.

2. Desarrolle un archivo de cabecera llamado **memcompa.h**, con las siguientes funciones:

    - int **agregar_msg**(char *msg);
    - void **modificar_msg**(int clave, char valor[100])
    - char* **consultar_msg**(int clave)
    - void **destruir_msg**(int clave)

3. Debe realizar un único programa con soporte nativo para los sistemas operativos Windows y Linux, para ello, debe utilizar en el código fuente directivas de pre-procesador para utilizar las librerías y funciones correctas del sistema operativo en el cual se compilará el programa.

4. La comunicación se hace compartiendo datos en memoria, no puede utilizar *sockets* de red.

(Valor 18 puntos)