

Proyecto: Limitador de memoria para procesos en Linux

1. Resumen Ejecutivo

Este proyecto tiene como objetivo el desarrollo de un limitador de memoria que se encargará de mantener los recursos de la computadora más bajos, en casos que se necesiten correr procesos que si la tienen, utilizaran toda la memoria posible del sistema. Este comportamiento no es siempre el adecuado. Por lo que desarrollarán un limitador que permita modificar la cantidad máxima de HEAP que un proceso puede pedir a través de la función malloc (malloc -> mmap). Deben implementar un sistema de memoria que guarde registro de qué procesos están limitados en una lista. Esta lista de momento no será persistente, y debe residir solamente en memoria (en espacio de kernel).

2. Objetivos del Aprendizaje

2.1 Objetivo General

- Desarrollar un limitador de memoria que cuente con las operaciones CRUD de procesos en una lista enlazada simple dentro de memoria en espacio de kernel.

2.2 Objetivos Específicos

- Comprender los principios de asignación de memoria, memoria alojada y límite de recursos.
 - Implementar técnicas de control de asignación de memoria a funciones como malloc, que internamente hace uso de mmap
 - Entender el funcionamiento de malloc y como un proceso en espacio de usuario solicita dinámicamente más memoria.
 - Adquirir experiencia práctica en la manipulación y modificación del kernel de Linux para la gestión de memoria.
-

3. Enunciado del Proyecto

3.1 Descripción del problema a resolver

Hay procesos que acaparan muchísima memoria si la encuentran disponible en el sistema, dejando a otros procesos sin memoria. Por lo que deben implementar un sistema de limitación de recursos que les permita poner un límite a la cantidad de memoria máxima que los procesos pueden solicitar dinámicamente. Esto lo deben hacer a través de un control en memoria del kernel, el cual monitoree con ayuda de una estructura de lista dinámica que procesos deben ser limitados y cual es el valor máximo de memoria para cada uno de ellos.

3.2 Alcance del proyecto

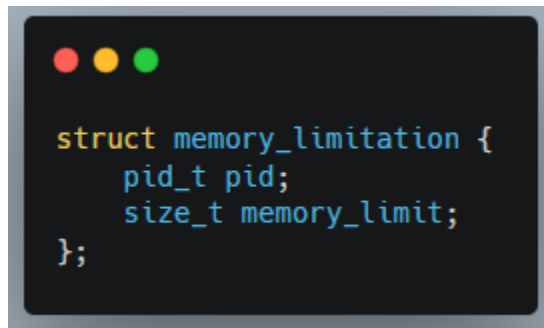
- Implementación de un algoritmo de limitación asignación de memoria.
- Configuración del entorno de desarrollo en Linux y compilación de un kernel modificado, con integración de los algoritmos y técnicas de asignación de memoria.

3.3 Requerimientos técnicos

- Implementación en lenguaje C.
- Conocimiento de llamadas al sistema y estructuras internas del kernel de Linux.
- Configuración de un entorno de desarrollo en Linux y re-compilación del kernel para verificar los cambios implementados.
- Continuación del kernel modificado del proyecto 2, Linux 6.8.0-49-usac1

3.4 Entregables

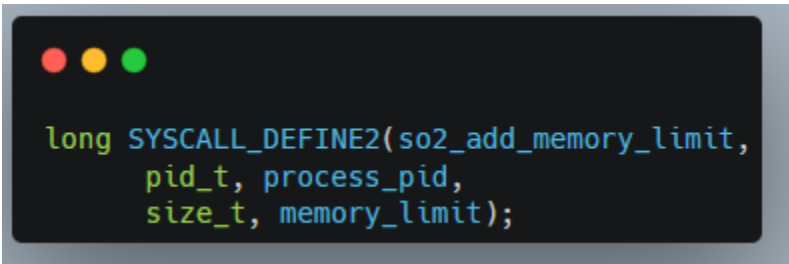
- **Syscall 1: Limitar un proceso**
 1. Se debe poder agregar entradas de procesos nuevos a ser limitados
 2. Debe estar limitada al uso de **usuarios *sudoers***
 3. Cada entrada a la lista debe hacer uso del siguiente struct



```
struct memory_limitation {  
    pid_t pid;  
    size_t memory_limit;  
};
```

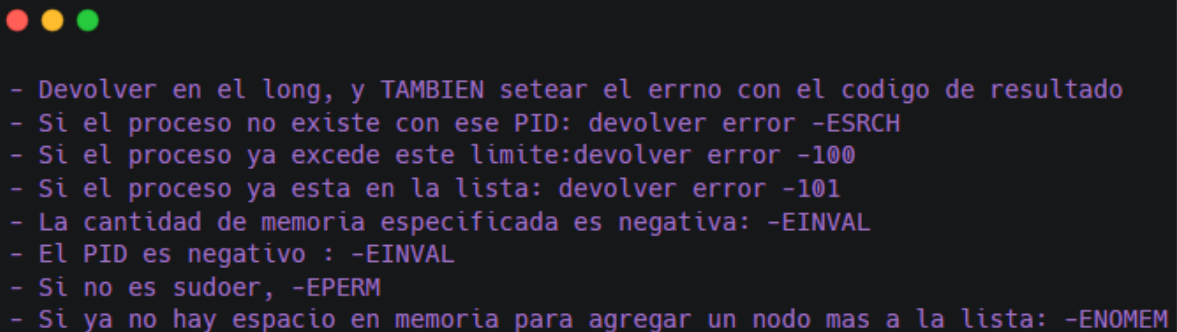
4. Debe tener el **ID de syscall 557**

5. Debe contar con la siguiente **definición de función**



```
long SYSCALL_DEFINE2(so2_add_memory_limit,  
    pid_t, process_pid,  
    size_t, memory_limit);
```

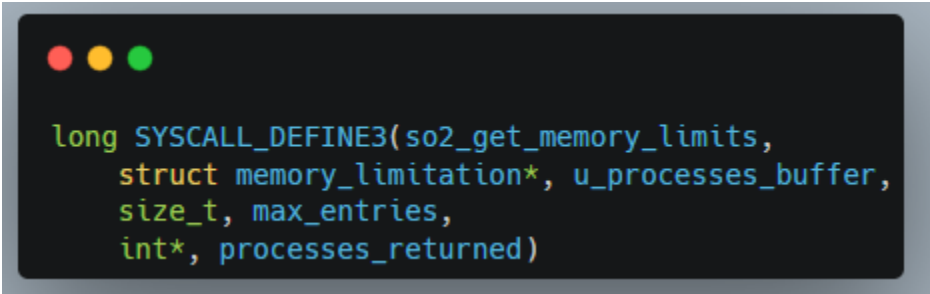
6. Debe tener el siguiente **manejo de errores**



```
- Devolver en el long, y TAMBIEN setear el errno con el codigo de resultado  
- Si el proceso no existe con ese PID: devolver error -ESRCH  
- Si el proceso ya excede este limite:devolver error -100  
- Si el proceso ya esta en la lista: devolver error -101  
- La cantidad de memoria especificada es negativa: -EINVAL  
- El PID es negativo : -EINVAL  
- Si no es sudoer, -EPERM  
- Si ya no hay espacio en memoria para agregar un nodo mas a la lista: -ENOMEM
```

- **Syscall 2: Obtener una lista de procesos limitados**

1. Se debe poder obtener la lista de procesos que han sido limitados
2. Se debe pasar el puntero a un struct donde el kernel escribirá los procesos
3. Se debe indicar el tamaño maximo de nuestro buffer
4. Se debe pasar un puntero para que el kernel nos indique cuántos procesos realmente escribió en nuestro buffer
5. Dicho lo anterior, la **definición de función** debe tener la siguiente forma



```
long SYSCALL_DEFINE3(so2_get_memory_limits,  
    struct memory_limitation*, u_processes_buffer,  
    size_t, max_entries,  
    int*, processes_returned)
```

6. Debe tener el **ID de syscall 558**

7. Se debe **manejar los errores** de la siguiente manera

```
- Devolver en el long, y TAMBIEN setear el errno con el codigo de resultado
- Tienen que llenar el struct con las entradas correspondientes en espacio de usuario
- Tiene que setear processes_returned a la cantidad de procesos que se escribieron
- Validar el puntero de user-space: -EINVAL
- Si el max es <=0: -EINVAL
- Si el buffer es mas pequeño, solo usar el limite del buffer (no se devuelve error)
```

8. Puede ser utilizada por cualquier usuario

- **Syscall 3: Actualizar el límite de un proceso**

1. Se debe poder actualizar entradas de procesos previamente limitados
2. Debe estar limitada al uso de **usuarios *sudoers***
3. Debe tener el **ID de syscall 559**
4. Debe contar con la siguiente **definición de función**

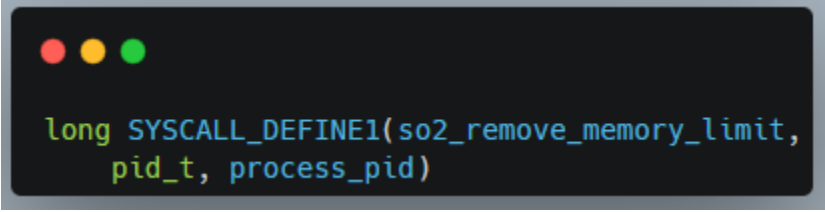
```
long SYSCALL_DEFINE2(so2_update_memory_limit,
                    pid_t, process_pid,
                    size_t, memory_limit)
```

5. Debe manejar errores de la siguiente forma

```
- Devolver en el long, y TAMBIEN setear el errno con el codigo de resultado
- Si el proceso no existe con ese PID: devolver error -ESRCH
- Si el proceso ya excede este limite:devolver error -100
- Si el proceso NO esta en la lista:devolver error -102
- La cantidad de memoria especificada es negativa: -EINVAL
- El PID es negativo : -EINVAL
- Si no es sudoer, -EPERM
```

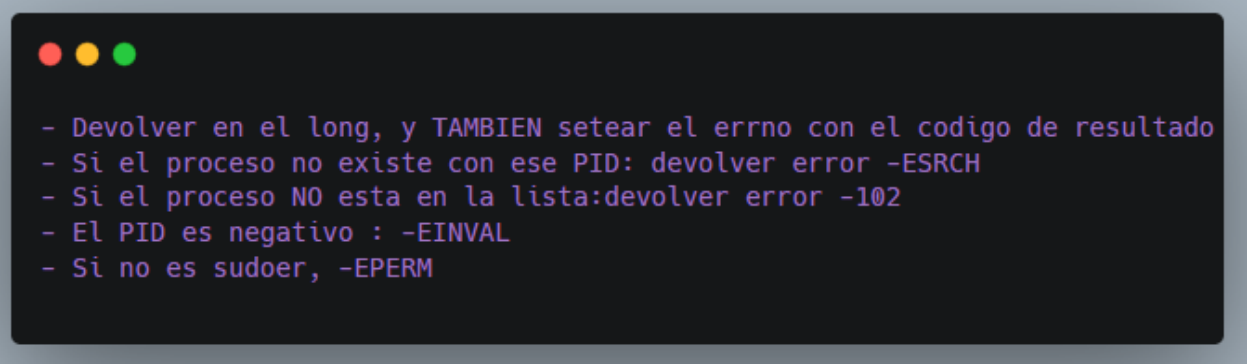
- **Syscall 4: Remover el límite de un proceso**

1. Se debe poder remover entradas de procesos previamente limitados
2. Debe estar limitada al uso de **usuarios sudoers**
3. Debe tener el **ID de syscall 560**
4. Debe contar con la siguiente **definición de función**



```
long SYSCALL_DEFINE1(so2_remove_memory_limit,  
pid_t, process_pid)
```

5. Debe manejar errores de la siguiente forma



```
- Devolver en el long, y TAMBIEN setear el errno con el codigo de resultado  
- Si el proceso no existe con ese PID: devolver error -ESRCH  
- Si el proceso NO esta en la lista:devolver error -102  
- El PID es negativo : -EINVAL  
- Si no es sudoer, -EPERM
```

- **Pruebas y scripts de demostración:**

1. Scripts en C/C++ que hagan uso de estas syscalls verificando los errores descritos.
2. Scripts en C o bash para demostrar que las operaciones CRUD de procesos limitados funcionan correctamente
3. Scripts en C o bash que demuestren los límites de memoria en los procesos seleccionados.

- **Documentación:**

1. Explicación del diseño de sus nuevas syscalls y llamadas a funciones internas del kernel
2. Cronograma de actividades
3. Problemas encontrados con el formato (foto problema/causa/solución)

4. **Mensaje PERSONAL de conclusión.** Debe ser una conclusión escrita por ustedes. Por ejemplo curiosidades, qué aprendieron, qué se les dificultó en el desarrollo de esta fase del proyecto, qué opinan acerca del curso, etc.
-

4. Restricciones

1. Es obligatorio la implementación de esta funcionalidad a través del kernel de Linux. Está prohibido el uso de módulos de kernel.
 2. Es **OBLIGATORIO** el mensaje personal de conclusión dentro de la documentación para tener derecho a la calificación.
 3. Es obligatorio usar el formato y reglas de desarrollo del kernel de Linux para este proyecto. Específicamente la implementación de las syscalls en archivo(s) por aparte. Documentar, comentar y elegir nombres claros y significativos es parte importante de su desarrollo como ingenieros en sistemas.
 4. Se calificará sobre el último commit realizado a su repositorio
 5. Queda a discreción del estudiante si hacer uso de **rlimit** o de interceptar las llamadas que el usuario hace desde userspace con **malloc** para limitar la memoria de los procesos. Tener en cuenta la necesidad de remover el límite establecido por el usuario, regresando a sus límites normales.
-

5. Metodología

6. **Investigación preliminar:** Los estudiantes deberán investigar sobre algoritmos de asignación de memoria en espacio de usuario, específicamente malloc y free.
 7. **Diseño de limitación de memoria:** Desarrollo de las operaciones CRUD de la lista de procesos
 8. **Integración en el kernel:** Modificación e integración de la solución, seguido de la recopilación y prueba.
 9. **Pruebas y validación:** Ejecución de pruebas para verificar el funcionamiento del limitador de memoria. La presentación de los datos queda a discreción del estudiante, media vez sea un medio ordenado y fácil de interpretar.
-

6. Cronograma

- **Día 1:** Investigación de estructura de listas en kernel y maneras de limitar recursos a procesos.
- **Día 2:** Diseño e implementación de la estructura de datos para la lista.
- **Día 2-3:** Desarrollo de la limitación de memoria
- **Día 4:** Validaciones de errores de argumentos y runtime de las syscalls
- **Día 5:** Pruebas, resolución de errores y elaboración de documentación