Proyecto: Asignador de memoria con lazy-zeroing en Linux

---

1. Resumen Ejecutivo

Este proyecto tiene como objetivo el desarrollo de un nueva variante en la familia de asignadores de memoria el cual asegure la inicialización de memoria en 0 sin marcar toda su memoria como utilizada inmediatamente. Los estudiantes trabajarán en la creación de una de esta variante, la cual tomará el nombre de tamalloc, la cual asegurará que las páginas pertenecientes a la memoria reservada sean inicializadas en 0 una por una según sean accedidas, permitiendo que su memoria en estado RSS no se dispare inmediatamente. Esto con el fin de mantener esa memoria sin páginas físicas inmediatamente, permitiendo al sistema dar un mejor manejo del over-commit.

---

2. Objetivos del Aprendizaje

2.1 Objetivo General

- Desarrollar un asignador de memoria que inicialice memoria en 0 u otro char predeterminado sin reservar páginas físicas inmediatamente.

2.2 Objetivos Específicos

- Comprender los principios de asignación de memoria, páginas físicas, RSS, memoria reservada y memoria alojada.
- Implementar técnicas de control de acceso a páginas que fueron reservadas con este nuevo asignador.
- Entender el concepto de lazy-loading, lazy-init, lazy-zeroing.
- Entender mapping de memoria virtual con masks como MAP_NORESERVE, MAP_PRIVATE y MAP_ANONYMOUS
- 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

Algunos procesos buscan inicializar espacios de memoria en 0 para control de algunas estructuras o arrays. Pero en el proceso de hacerlo, activan la dedicación de páginas físicas inmediatamente a todo ese espacio de memoria. Para batallar este problema, se necesita de una función de asignación de memoria que sea capaz de garantizar que el espacio en memoria sea inicializado en 0, sin necesidad de disparar inmediatamente el uso de páginas físicas. Esto a través de retrasar la necesidad y el llenado de “0” hasta el primer acceso de lectura/escritura a los segmentos de memoria individuales del bloque reservado.

3.2 Alcance del proyecto

- Implementación de un algoritmo de asignación de memoria con inicialización en cero.
- Diseño de técnicas de control de acceso a páginas
- 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 básico 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 1, Linux 6.8.0-49-usac1

3.4 Entregables

- **Código fuente del kernel modificado:**

    1. **Implementación de Algoritmo de Asignación de Memoria:**

        Los estudiantes deben modificar el código del kernel para implementar un algoritmo de asignación de memoria como se describe a continuación:

        - **Tamalloc con lazy-zeroing:**

            Este algoritmo es una variación del asignador de memoria malloc. Su principal diferencia es la inicialización del espacio de memoria en 0. En esto tiene más parecido a kzalloc, calloc y parecidos. La diferencia principal con estos asignadores de memoria, es qué Tamalloc debe estar diseñado para NO reservar páginas físicas inmediatamente al momento de asignar memoria. **Debe ser hasta el primer acceso a cada página** (ya sea escritura o lectura) que cause un page fault, **que esta región de página debe ser inicializada en 0.**

El código debe incluir comentarios detallados sobre los pasos necesarios para cumplir con los siguientes requisitos:

- Asignar la memoria suficiente para abarcar la memoria solicitada
- No causar page faults al momento de asignar esta memoria
- Inicializar cada página en 0 hasta su primer acceso (lectura o escritura)

2. **Recolección de estadísticas de asignación de memoria**

   Los estudiantes deberán implementar 2 syscalls para la recolección general de estadísticas de uso de memoria reservada y de memoria marcada como utilizada. Las estadísticas a recolectar son:

   - Para CADA proceso individual:
     - Memoria reservada (Reserved) (en KB/MB)
     - Memoria utilizada (Commited) (en KB/MB, y en % de memoria reservada)
     - OOM SCore
   - En resumen de TODOS los procesos:
     - Memoria total reservada (Reserved) (en MB)
     - Memoria total utilizada (Commited) (en MB)

- **Pruebas y scripts de demostración:**
  1. Scripts en C/C++ que hagan uso de esta syscall para asignar memoria.
  2. Scripts en C o bash para demostrar que la memoria siendo asignada no es marcada inmediatamente utilizada (RSS, no page-fault, etc).
  3. Scripts en C o bash para mostrar en consola las estadísticas recolectadas de memoria.
- **Informe técnico detallado:**
  1. Explicación del diseño del algoritmo de asignación de memoria y sus llamadas a funciones internas del kernel.
  2. Análisis de los resultados obtenidos de memoria solicitada vs memoria realmente siendo utilizada.
  3. Cronograma de actividades
  4. Problemas encontrados (problema/causa/solución)
  5. Mensaje PERSONAL de conclusión.

4. Metodología

1. **Investigación preliminar:** Los estudiantes deberán investigar sobre algoritmos de asignación de memoria como kmalloc, kzalloc, malloc, calloc. mmap, free, new, delete.
2. **Diseño de algoritmos de memoria:** Desarrollo del algoritmo Tamalloc de asignación de memoria con lazy-zeroing
3. **Integración en el kernel:** Modificación e integración de los algoritmos, seguido de la recopilación y prueba.
4. **Pruebas y validación:** Ejecución de pruebas para verificar el funcionamiento del asignador de memoria y de las estadísticas. La presentación de los datos queda a discreción del estudiante, media vez sea un medio ordenado y fácil de interpretar.

---

5. Cronograma

- **Día 1:** Investigación de algoritmos de asignación de memoria.
- **Día 2-4:** Diseño e implementación del algoritmo de asignación con lazy-zeroing.
- **Día 5:** Desarrollo de la recolección de estadísticas generales de memoria.
- **Día 6:** Integración en el kernel, pruebas finales y elaboración del informe técnico.
- **Día 7:** Resolución de errores y detalles e implementación final con las pruebas.