Título del Proyecto: Expansión del Kernel de Linux con Nuevas Funcionalidades

---

1. Resumen Ejecutivo

Este proyecto desafía a los estudiantes a personalizar y expandir el kernel de Linux, permitiéndoles modificar su comportamiento y agregar nuevas funcionalidades. Involucra la configuración de un entorno de desarrollo adecuado para la compilación del kernel, la implementación de cambios personalizados en el sistema y el desarrollo de módulos que añaden llamadas al sistema. A través de este proyecto, los estudiantes adquirirán habilidades fundamentales en programación de sistemas y entenderán los principios básicos de los núcleos de sistemas operativos.

---

2. Objetivos de Aprendizaje

2.1 Objetivo General:

- Modificar y personalizar el kernel de Linux para añadir nuevas llamadas al sistema y funcionalidades específicas.

2.2 Objetivos Específicos:

- Configurar un entorno de desarrollo para la compilación y modificación del kernel.
- Descargar y compilar el kernel de Linux desde el código fuente.
- Modificar el kernel para personalizar el nombre del sistema y añadir mensajes personalizados en el arranque.
- Desarrollar módulos del kernel que permitan obtener estadísticas del sistema.

---

3. Enunciado del Proyecto

3.1 Descripción del problema a resolver:

- En el ámbito de los sistemas operativos, la capacidad de personalizar y expandir el kernel permite a los desarrolladores ajustar el funcionamiento del sistema a necesidades específicas. Este proyecto aborda el desarrollo de nuevas funcionalidades en el kernel de Linux, brindando a los estudiantes la oportunidad de entender su estructura y funcionamiento interno, además de aprender a trabajar con módulos y llamadas al sistema personalizadas.

3.2 Alcance del proyecto:

- Los estudiantes trabajarán en una máquina virtual Linux para realizar modificaciones controladas del kernel. Las personalizaciones estarán limitadas a aspectos específicos como la modificación del nombre del sistema, mensajes de arranque y el desarrollo de módulos para obtener estadísticas del sistema. No se espera una modificación extensa del sistema operativo ni el desarrollo de nuevas arquitecturas o drivers avanzados.

3.3 Requerimientos técnicos:

- Entorno de desarrollo: Máquina virtual con Linux (recomendado Ubuntu o Debian).
- Herramientas: Make, GCC, GDB, QEMU (para pruebas), y Git para control de versiones.
- Tecnologías: Lenguaje C para la programación en el kernel y BASH para automatizar procesos de compilación y pruebas.
- Distribución: Linux Mint 22, Ubuntu 20.04 - 24.04 o derivados de Debian compatibles con el kernel a utilizar
- Kernel de Linux: Versión 6.8.0 de www.kernel.org

3.4 Entregables:

1. **Entorno de desarrollo configurado y documentado:**

   Los estudiantes deben entregar una máquina virtual o un sistema Linux configurado, documentando los pasos seguidos para instalar el kernel y las herramientas necesarias (Make, GCC, GDB, QEMU, etc.). Esto incluye capturas de pantalla y/o archivos de configuración que demuestren que el entorno está listo para compilar y ejecutar el kernel personalizado.

2. **Código fuente del kernel modificado:**

   Se debe entregar el código fuente del kernel con modificaciones básicas. Esto incluye:

   - **Personalización del nombre del sistema:** Modificar el nombre del sistema que aparece en el arranque para reflejar un nombre personalizado, elegido por el estudiante, que demuestre que el kernel ha sido modificado.
   - **Mensajes de inicio personalizados:** Añadir un mensaje de bienvenida personalizado que aparezca durante el arranque del sistema.

   Estos cambios deben estar bien documentados en el código (con comentarios claros) y en un informe breve que describa cómo y dónde se hicieron las modificaciones en el código del kernel.

3. **Implementación de nuevas llamadas al sistema:**

   Los estudiantes deben desarrollar las siguientes llamadas al sistema personalizadas, integradas en el kernel y debidamente documentadas.

   - **capture_memory_snapshot**
     - **Descripción:** Esta llamada realiza una captura del estado de la memoria en un instante de tiempo, guardando un "snapshot" o instantánea de las áreas de memoria ocupadas y libres, así como información sobre las páginas de memoria activas. Esta información puede ayudar a entender el uso exacto de la memoria y el estado de fragmentación en tiempo real.
     - **Implementación:** Esta llamada accede a la tabla de páginas del sistema y devuelve detalles estructurados del uso de memoria (páginas activas, caché, swap, etc.). Para esta implementación, los estudiantes deben entender el manejo de tablas de páginas dentro del kernel y desarrollar estructuras que recopilen y organicen esta información en un formato fácil de interpretar.
   - **track_syscall_usage**
     - **Descripción:** Esta llamada permite al usuario monitorizar cuántas veces y cuándo se ejecuta una lista de llamadas al sistema específicas, como open, write, read, fork, entre otras. Esto es útil para auditorías de seguridad o para aplicaciones de rendimiento en donde es necesario entender los patrones de uso de las llamadas al sistema.
     - **Implementación:** Los estudiantes deberán modificar la tabla de llamadas del sistema y realizar un rastreo de cada una de las llamadas seleccionadas. La implementación incluye

interceptar estas llamadas en tiempo de ejecución, contabilizar el número de veces que se llaman y almacenar la información en estructuras internas que pueden consultarse.

- **get_io_throttle**
  - **Descripción:** Esta llamada permite al usuario obtener información estadística de uso de I/O de cada proceso. Por ejemplo, cantidad de bytes escrita, cantidad de bytes leída, cantidad de bytes escrita a disco, cantidad de bytes leída de disco, tiempo esperando completación de operación I/O, entre otros. Queda a discreción del estudiante que estadísticas usar exactamente, deben ser por lo menos 5 relevantes.
  - **Implementación:** Los estudiantes deberán integrar esta llamada con las operaciones de entrada y salida del kernel. Esto implica interceptar las operaciones I/O en el subsistema de bloques, o usar estructuras existentes como task_struct->ioac (task_io_ accounting), o utilizando la informacion disponible a través de /proc/<pid>.

Los estudiantes deben documentar el diseño y la implementación de las llamadas en el código, explicando el funcionamiento en un informe técnico detallado.

4. **Módulos del kernel para la recopilación de estadísticas del sistema:**

Desarrollar y compilar un módulo de kernel por cada llamada creada del sistema anteriormente. El módulo debe mostrar ordenadamente estas estadísticas en consola (se sugiere mediante una entrada a /proc). Además, se creará un módulo extra que devuelva la siguiente información.

- **Estadísticas de CPU:** El módulo debe obtener y registrar el porcentaje de uso de CPU.
- **Estadísticas de memoria:** Debe mostrar el uso actual de memoria total y memoria disponible.
- **Estadísticas de almacenamiento:** Muestra el espacio total y el espacio libre en el disco, específico a una partición indicada.

Estos módulos deben probarse y documentarse, demostrando su funcionamiento con un script que recopile y muestre estos datos en la terminal.

5. **Informe técnico final:**

Los estudiantes deben redactar un informe técnico completo que incluya:

- **Introducción y objetivos del proyecto:** Una breve explicación del propósito del proyecto y las metas de las modificaciones realizadas.
- **Configuración del entorno:** Una descripción detallada del proceso de configuración, compilación y arranque del kernel modificado.
- **Descripción de modificaciones en el kernel:** Explicación de las personalizaciones del nombre del sistema y los mensajes de inicio, incluyendo los archivos modificados y el código.

- **Detalles de las nuevas llamadas al sistema:** Documentación completa de las llamadas realizadas, con su diseño, propósito y el código implementado, así como ejemplos de uso.
- **Pruebas realizadas:** Una descripción de las pruebas hechas en el kernel y los módulos, incluyendo los resultados de las pruebas y cualquier problema o ajuste realizado.
- **Reflexión personal:** Conclusiones del proyecto y áreas de mejora.

---

4. Metodología

1. **Investigación preliminar:**
    - Los estudiantes investigarán el proceso de compilación del kernel y la estructura básica del código fuente del kernel de Linux.
2. **Configuración del entorno:**
    - Crear un entorno de desarrollo en una máquina virtual o equipo de prueba, instalando las herramientas y dependencias necesarias.
3. **Modificación básica del kernel:**
    - Descargar y compilar el kernel de Linux.
    - Personalizar el nombre del sistema y añadir mensajes personalizados en el arranque.
4. **Desarrollo de nuevas funcionalidades:**
    - Desarrollar módulos específicos del kernel que proporcionen estadísticas del sistema (como uso de CPU, memoria y disco).
5. **Pruebas y ajustes:**
    - Realizar pruebas exhaustivas del kernel modificado, asegurando la estabilidad del sistema y documentando cualquier error o conflicto encontrado.
6. **Documentación final:**
    - Crear un informe técnico que describa el proceso, el código desarrollado, los problemas encontrados y los resultados obtenidos.

---

5. Desarrollo de Habilidades Blandas

5.1 Autogestión del Tiempo:

- Los estudiantes deben planificar un cronograma de actividades que incluya la configuración, modificación, desarrollo y pruebas del kernel.

5.2 Responsabilidad y Compromiso:

- Al asumir el control total sobre un sistema Linux y su kernel, el estudiante tiene la oportunidad de trabajar de manera cuidadosa y metódica, entendiendo la importancia de cada cambio.

5.3 Resolución de Problemas:

- La modificación del kernel es una tarea compleja, por lo que el estudiante debe ser capaz de identificar problemas y buscar soluciones eficaces de manera autónoma.

5.4 Reflexión Personal:

- Al finalizar el proyecto, el estudiante realizará una autoevaluación, reflexionando sobre las decisiones técnicas tomadas, los desafíos superados y las áreas donde podría mejorar.

---

6. Cronograma

1. **Dia 1:** Compilación y modificación básica del kernel.
2. **Dia 2:** Desarrollo de módulos personalizados del kernel.
3. **Dia 3-5:** Creación de las syscalls en el kernel
4. **Dia 6:** Pruebas y ajustes de la solución.
5. **Dia 7:** Documentación final y presentación del proyecto.

---

7. Evaluación

La evaluación del proyecto se basará en los siguientes criterios:

1. Implementación de Llamadas al Sistema (50 puntos en total)

- **capture_memory_snapshot (15 puntos)**
    - Implementación del snapshot de memoria: 7.5 puntos

    - Organización y optimización de datos: 4.5 puntos
    - Prueba de funcionamiento: 3 puntos
- **track_syscall_usage (20 puntos)**
    - Implementación del monitoreo de llamadas al sistema: 10 puntos
    - Manejo de sincronización y control de acceso: 6 puntos
    - Prueba de funcionamiento: 4 puntos
- **get_io_throttle (15 puntos)**
    - Estadísticas de escritura/lectura general: 7.5 puntos
    - Estadísticas de escritura/lectura específicas a disco: 4.5 puntos
    - Prueba de funcionamiento: 3 puntos

2. Calidad del Código y Documentación (15 puntos)

- Documentación del código y comentarios claros: 5 puntos
- Organización de archivos y estructura del proyecto: 5 puntos
- Informe técnico final con explicación completa: 5 puntos

3. Pruebas y Validación (10 puntos)

- Pruebas exhaustivas y detalladas: 5 puntos
- Scripts de prueba claros y funcionales: 5 puntos

4. Habilidades Blandas (10 puntos)

- Gestión del tiempo y cronograma: 2.5 puntos
- Resolución de problemas y manejo de dificultades: 2.5 puntos
- Responsabilidad y compromiso: 2.5 puntos
- Reflexión personal y autoevaluación: 2.5 puntos