UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

Sistemas Operativos y Laboratorio

Proyecto Final “Creación de una API para analizar el rendimiento de aplicaciones”

Equipo:
Johan Sebastian Henao Cañas
johan.henao1@udea.edu.co
Alejandro Becerra Acevedo
alejandro.becerraa@udea.com.co

Tutores:

Ronal Montoya
ronal.montoya@udea.edu.co
Henry Alberto Arcila Ramirez
henry.arcila@udea.edu.co

Medellín

2024

Resumen

Este proyecto tiene como objetivo desarrollar una API que analice el rendimiento de aplicaciones en tiempo real, enfocándose en parámetros clave como el uso de CPU, memoria y dispositivos de entrada/salida (E/S). Además, permitirá realizar perfiles detallados de funciones específicas dentro de las aplicaciones, proporcionando información valiosa para optimizar el uso de recursos y mejorar la eficiencia. Para lograr esto, se integrarán herramientas avanzadas de análisis de rendimiento como Valgrind y perf, las cuales permitirán monitorear el comportamiento del sistema y de las aplicaciones, identificando posibles cuellos de botella.

El análisis de los datos recolectados por la API se validará mediante técnicas estadísticas como Análisis de Varianza (ANOVA), pruebas de hipótesis y regresión lineal, para verificar la efectividad de la API en diferentes condiciones de uso. Estas técnicas permitirán evaluar si los cambios en el rendimiento de las aplicaciones son significativos y proporcionar información fiable para futuras optimizaciones. Este proyecto está profundamente vinculado a los principios de sistemas operativos, ya que estudia cómo gestionar de manera eficiente los recursos del sistema y maximizar el rendimiento de las aplicaciones en entornos multitarea.

Introducción

En el contexto actual, la eficiencia y el rendimiento de las aplicaciones juegan un papel crucial en el éxito de las organizaciones, ya que el uso óptimo de los recursos del sistema puede mejorar la productividad y reducir costos operativos. Sin embargo, muchas aplicaciones presentan cuellos de botella y problemas de gestión de recursos, lo que afecta su desempeño en situaciones de alta demanda. Estos problemas son particularmente críticos en sistemas multitarea, donde varias aplicaciones compiten simultáneamente por el uso de CPU, memoria y dispositivos de entrada/salida (E/S).

El desafío que aborda este proyecto es la creación de una API que permita a los desarrolladores y administradores de sistemas analizar el rendimiento de aplicaciones de forma detallada. Al proporcionar perfiles específicos sobre el uso de recursos, esta API permitirá identificar con precisión las funciones o procesos que impactan negativamente en el rendimiento general de una aplicación. Asimismo, brindará un monitoreo continuo en tiempo real, proporcionando datos clave para la toma de decisiones en optimización de recursos.

Este proyecto es relevante en el contexto tecnológico actual, dado que la creciente complejidad de las aplicaciones modernas y la diversidad de entornos donde operan requieren herramientas más robustas para identificar problemas de rendimiento de manera rápida y eficiente. La capacidad de optimizar el uso de recursos es especialmente importante en aplicaciones distribuidas, plataformas en la nube y sistemas con recursos limitados, donde cualquier ineficiencia puede tener un impacto significativo en la experiencia del usuario y en los costos de operación.

Antecedentes o marco teórico

Para desarrollar la API que permita analizar el rendimiento de aplicaciones, es crucial tener un sólido entendimiento de varios conceptos técnicos y comandos utilizados en sistemas operativos.

La API deberá manejar procesos, gestionar memoria y medir el uso de recursos como CPU y almacenamiento, apoyándose en herramientas y funciones específicas del sistema operativo.

Un aspecto central es la gestión de procesos, donde comandos como fork() y exec() son fundamentales. La llamada fork() permite crear nuevos procesos, lo que es esencial para gestionar la concurrencia en la API, mientras que las variaciones de exec() permiten ejecutar nuevos programas dentro de esos procesos. Además, el comando wait() o waitpid() se utilizará para sincronizar procesos y asegurarse de que los procesos hijos se ejecuten y finalicen correctamente antes de que el padre continúe.

Otro concepto crucial es la medición del uso de recursos, para lo cual se emplearán herramientas como perf y Valgrind. Perf es una herramienta poderosa para recopilar información sobre el rendimiento del sistema, como contadores de eventos, trazas de ejecución y análisis de hotspots de CPU. Los comandos clave de perf incluyen:

- **perf stat**: para obtener conteos de eventos específicos del sistema como instrucciones ejecutadas y fallos de caché.
- **perf record**: para grabar eventos de rendimiento y analizarlos posteriormente.
- **perf report**: para desglosar el rendimiento por función o proceso.

Por otro lado, Valgrind se utilizará para analizar la gestión de memoria, identificando fugas de memoria y errores en la gestión de punteros. Esto es crucial para garantizar que la API funcione correctamente bajo condiciones de carga sin consumir innecesariamente los recursos del sistema.

En cuanto a los métodos estadísticos, su uso es esencial para analizar y validar los resultados obtenidos de las mediciones de rendimiento. Algunos de los métodos más relevantes son:

- **Distribuciones de frecuencias**: Al medir el rendimiento del sistema (por ejemplo, el uso de CPU o memoria), se pueden recolectar datos en intervalos regulares. La distribución de frecuencias permite agrupar estos datos en intervalos (o bins) para visualizar la frecuencia de ocurrencias, detectando picos de uso y periodos de inactividad.
- **Media y desviación estándar**: Estos indicadores son fundamentales para evaluar el rendimiento promedio y la variabilidad de los resultados obtenidos por la API. La media proporciona una visión general del comportamiento típico, mientras que la desviación estándar permite entender la dispersión de los tiempos de ejecución o el uso de recursos, detectando variaciones significativas que puedan indicar problemas.
- **Correlación**: Permite analizar la relación entre diferentes métricas de rendimiento, como el uso de CPU frente al uso de memoria. De esta forma, se puede identificar si un aumento en un recurso (por ejemplo, memoria) afecta de manera directa o indirecta el comportamiento de otro (como el tiempo de respuesta).
- **Análisis de regresión**: Este método puede ser utilizado para predecir el impacto de cambios en la configuración del sistema sobre su rendimiento. Por ejemplo, se puede usar regresión para prever cómo se comportará el sistema bajo diferentes cargas de trabajo o configuraciones de hardware, basándose en datos históricos recolectados por la API.

Estas técnicas permiten transformar los datos brutos recolectados por perf, Valgrind u otras herramientas, en información útil que facilite la toma de decisiones sobre mejoras en el sistema.

Además de los métodos estadísticos, se utilizarán herramientas como ps y top para monitorear el estado de los procesos en tiempo real, mientras que time permitirá medir el tiempo de ejecución de comandos específicos y obtener datos sobre el tiempo de CPU usado. Estos comandos facilitarán la identificación de posibles cuellos de botella en el sistema operativo.

En cuanto al manejo de archivos y descriptores de archivos, comandos como open(), read(), y write() serán necesarios para gestionar la entrada y salida de archivos de manera eficiente. Esto es especialmente importante cuando se estén procesando logs de rendimiento o se necesite almacenar los resultados de las mediciones en archivos.

Finalmente, la API también debe ser capaz de gestionar la concurrencia y el paralelismo, para lo cual se utilizarán llamadas al sistema como pipe() para conectar procesos y permitir la comunicación entre ellos. Este mecanismo será útil para construir pipelines de procesamiento que permitan la transmisión de datos entre los diferentes componentes de la API.

Objetivo principal

El objetivo principal del proyecto es implementar una API que permita medir y analizar el rendimiento de aplicaciones en sistemas operativos Linux, utilizando herramientas y funciones del sistema operativo para monitorear el uso de recursos (CPU, memoria, disco, entre otros) y gestionar procesos de manera eficiente. Esta API será capaz de recopilar y reportar datos estadísticos detallados que permitan identificar cuellos de botella y optimizar el rendimiento de las aplicaciones.

Objetivos específicos

1. Crear una API en lenguaje C que utilice llamadas al sistema y comandos para medir el rendimiento. La API debe ser capaz de ejecutar programas y monitorear su comportamiento en términos de uso de recursos.
2. Incluir en la API la funcionalidad para medir el tiempo de ejecución de los procesos y recolectar métricas sobre el uso de recursos (CPU, memoria, I/O) mediante herramientas como perf stat y perf report.
3. Implementar en la API la capacidad de manejar procesos concurrentes y paralelos, utilizando llamadas al sistema para permitir la comunicación entre procesos y gestionar su ciclo de vida.
4. Aplicar métodos estadísticos como la media, desviación estándar, y distribuciones de frecuencias para analizar los datos recolectados por la API. Esto permitirá identificar patrones y anomalías en el rendimiento de las aplicaciones.
5. Crear una serie de pruebas y benchmarks para validar la precisión y eficiencia de la API, comparando los resultados con los obtenidos de herramientas tradicionales de análisis de rendimiento en sistemas operativos.

Metodología

***Herramientas principales**:*

1. **Lenguaje de programación C**: Será el lenguaje base para implementar la API. C ofrece control sobre la gestión de memoria y acceso directo a las llamadas del sistema, lo que es ideal para este tipo de proyectos.
2. **Llamadas al sistema en Linux**: Utilizaremos llamadas como fork(), exec(), wait(), pipe(), y open(), fundamentales para la creación y gestión de procesos, así como para el manejo de archivos en sistemas operativos Linux.
3. **Herramientas de monitoreo de rendimiento**:
    - **Valgrind**: Para detectar errores de memoria y realizar análisis detallados de uso de recursos.
    - **Perf**: Para obtener datos sobre el rendimiento del CPU, manejo de procesos, eventos del sistema, y trazas detalladas de ejecución.
    - **time**: Para medir el tiempo de ejecución de comandos y programas.
4. **Métodos estadísticos**:
    - **Media y desviación estándar**: Para analizar los tiempos de ejecución y el uso de recursos.
    - **Histogramas y distribuciones de frecuencia**: Para visualizar los resultados de los análisis de rendimiento.

Actividades:

1. **Investigación preliminar y revisión de herramientas**:
    - Realizar una revisión exhaustiva de las herramientas a utilizar, incluyendo los comandos y las bibliotecas necesarias para la interacción con el sistema operativo.
    - Familiarización con las herramientas de análisis de rendimiento perf y Valgrind para comprender cómo integrarlas en el desarrollo de la API.
2. **Diseño del esquema de la API**:
    - Planificar la estructura de la API, incluyendo los métodos principales, parámetros de entrada y salida, y la forma en que se gestionará la información recopilada de los procesos.
    - Definir las funciones para la gestión de procesos y monitoreo de recursos.
3. **Implementación de la API**:
    - **Implementación básica**: Desarrollar las primeras funciones que permitan crear y gestionar procesos utilizando las llamadas al sistema de Linux, como fork(), exec(), y wait().
    - **Monitoreo de rendimiento**: Integrar las herramientas perf y Valgrind en la API para recopilar datos sobre el uso de CPU, memoria y otros recursos. Incluir la medición de tiempos de ejecución con gettimeofday() y time.
    - **Procesos concurrentes**: Añadir la funcionalidad para gestionar procesos concurrentes y comunicación entre procesos mediante pipe().
4. **Pruebas y análisis**:
    - Diseñar y ejecutar pruebas de rendimiento con diferentes programas y parámetros para verificar la efectividad de la API en el monitoreo y gestión de procesos.
    - Utilizar los métodos estadísticos (media, desviación estándar, histogramas) para analizar los datos obtenidos y generar reportes de rendimiento.

Diseño de Experimentos

Título del Experimento: Validación del Rendimiento y Funcionalidad de la API para la Gestión de Procesos en Sistemas Operativos

Fecha:

[Especificar la fecha]

Breve Descripción:

Este experimento se centra en evaluar la eficacia y eficiencia de una API desarrollada para la gestión de procesos en sistemas operativos. Se analizará el rendimiento en términos de tiempo de ejecución, uso de memoria y CPU, así como la capacidad de la API para manejar múltiples procesos concurrentes.

Objetivos del Experimento:

1. Evaluar el tiempo de ejecución de la API bajo diferentes cargas de trabajo.
2. Medir el uso de memoria y CPU durante la ejecución de procesos.
3. Determinar el número máximo de procesos que la API puede gestionar sin degradar el rendimiento.

Hipótesis:

La API puede gestionar múltiples procesos concurrentes de manera eficiente, manteniendo un rendimiento aceptable en tiempo de ejecución, uso de memoria y CPU.

Predicciones:

- El tiempo de ejecución aumentará linealmente con el número de procesos.
- El uso de memoria se incrementará de acuerdo con la cantidad de procesos concurrentes.
- La API presentará un límite superior en el manejo de procesos, más allá del cual el rendimiento se verá afectado.

Métodos:

1. **Configuración del Entorno:** Implementación de la API en un entorno de pruebas controlado, con recursos definidos (CPU, memoria).
2. **Generación de Cargas:** Creación de diferentes escenarios de carga variando el número de procesos (10, 20, 50, 100).
3. **Recolección de Datos:** Durante cada prueba, se registrarán los tiempos de ejecución, el uso de memoria y CPU, y el número de procesos gestionados.
4. **Análisis de Datos:** Los datos recolectados serán analizados estadísticamente para identificar tendencias y determinar el rendimiento general de la API.

Lectura Experimental Real:

- **Métricas:** Tiempo de ejecución, uso de memoria, uso de CPU, número de procesos concurrentes.
- **Manipulación de Datos:** Normalización de datos según el número de procesos y recursos disponibles.

Réplicas y Estadísticas:

- Se realizarán al menos 10 pruebas por cada condición de carga.
- Cada condición se repetirá 3 veces para asegurar la validez de los resultados.

Controles:

- **Controles de Baseline:** Prueba inicial con un solo proceso para establecer una línea base.
- **Pruebas de Carga:** Evaluación de la API con diferentes niveles de carga para medir la eficiencia en condiciones extremas.

Conclusiones Esperadas:

El análisis debe revelar si la API puede manejar eficientemente múltiples procesos concurrentes y si hay límites en su rendimiento que deben ser abordados antes de su implementación en producción.

Ubicación de Datos:

Los resultados y scripts estarán documentados y almacenados en un repositorio de GitHub específico para el proyecto.

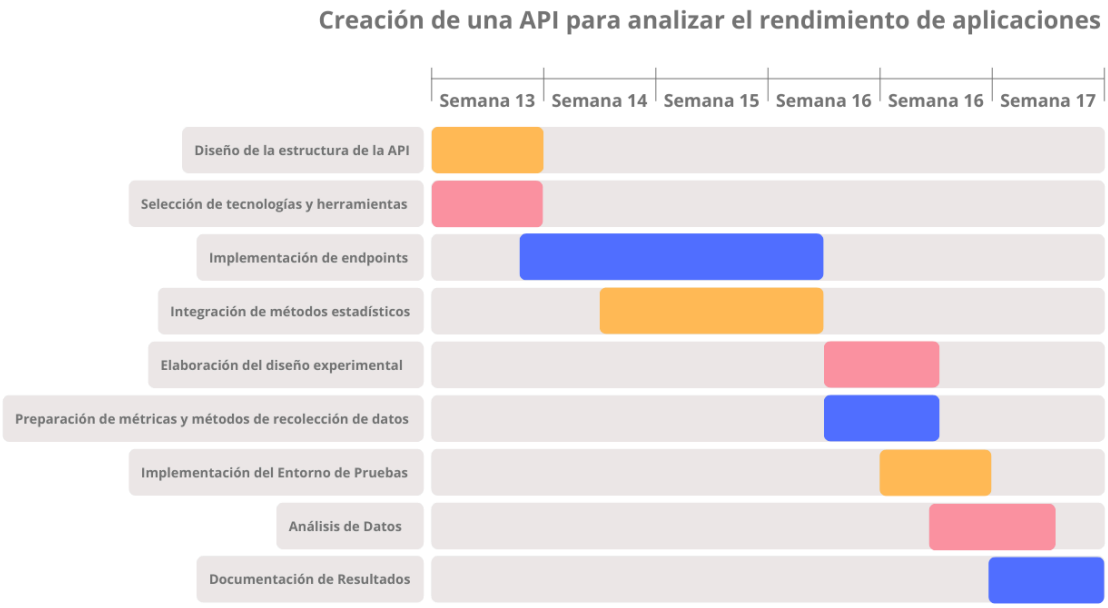
Contactos:

- **Instructores:** Claus Aranha, [Detalles de contacto].
- **Asistente de enseñanza:** Yifan He, [Detalles de contacto].

Notas y Recordatorios:

- Evaluar la API en diferentes escenarios de carga.
- Documentar cualquier anomalía en el rendimiento para futuras referencias.

Cronograma



Referencias

1. Hernández, A., & García, C. (2016). *Métodos Estadísticos en la Investigación Científica*. Editorial Universitaria.
2. Linux Documentation Project. (2024). *Linux Command Line and Shell Scripting Bible*. Wiley.
3. Kerrisk, M. (2010). *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press.
4. McKusick, M. K., & Neville, M. (2020). *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley.
5. Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Tesis de doctorado, Universidad de California, Irvine. https://ics.uci.edu/~fielding/pubs/dissertation/top.htm.
6. Pahl, C., & Sousa, J. (2014). *Cloud Computing for Data-Intensive Applications: A Comparative Study of Cloud Service Providers*. *IEEE Transactions on Cloud Computing*, 2(3), 248-263.
9. Valgrind. https://valgrind.org/
10. Kernel Performance Wiki. https://perf.wiki.kernel.org/index.php/Main_Page
11. SystemTap. https://es.wikipedia.org/wiki/SystemTap
12. Stanford University. *Experimental Design for the Behavioral and Social Sciences*. https://sing.stanford.edu/cs303-sp11/.