Universidad de Guadalajara
Sistema de Educación Media Superior
Centro Universitario de Ciencias Exactas e Ingenierías

Parte 2 Otras herramientas para el manejar errores

Materia: Computación Tolerante a Fallas

D06 2023 B

Alumno: Esquivel Barbosa Diego Humberto

Código: 211401635

Carrera: INCO

Fecha: 25/08/2023

Introducción

La computación tolerante a fallos se centra en el diseño, desarrollo y despliegue de sistemas que tienen la capacidad de resistir, mitigar y recuperarse de fallos y errores, manteniendo su funcionalidad esencial en situaciones adversas. Estos sistemas no solo aspiran a evitar la interrupción total, sino que también buscan ofrecer una operación confiable en escenarios donde componentes individuales pueden experimentar problemas. Ya sea en aplicaciones críticas para la seguridad, sistemas médicos, redes de comunicación o dispositivos de consumo, la tolerancia a fallos se ha convertido en un aspecto esencial para garantizar la integridad y la confianza en la tecnología que utilizamos diariamente.

En esta investigación, nos adentraremos en las herramientas y técnicas fundamentales del campo del manejo de errores. Pondremos el foco en un ejemplo concreto en el lenguaje de programación Python, donde aplicaremos estas herramientas para ilustrar cómo impactan en la construcción de software robusto. El programa que desarrollaremos servirá como un caso de estudio, demostrando cómo las herramientas de manejo de errores pueden ser implementadas en situaciones reales, con el objetivo de mejorar la estabilidad y calidad de las aplicaciones.

En el transcurso de este análisis, exploraremos cómo las herramientas de depuración, el registro de errores, la gestión de excepciones y otras técnicas se amalgaman para proporcionar un enfoque integral y eficaz para afrontar y superar los desafíos de la programación. Al concluir, habremos obtenido una comprensión más profunda de cómo estas herramientas no solo contribuyen a resolver problemas, sino que también enriquecen la perspectiva de los desarrolladores y mejoran la calidad de las aplicaciones resultantes.

Ejemplo utilizando herramientas.

Supongamos que estamos desarrollando un programa que realiza cálculos matemáticos complejos y queremos asegurarnos de manejar adecuadamente cualquier error que pueda ocurrir durante la ejecución.

Depuradores (Debuggers): Usaremos un depurador para ejecutar nuestro programa paso a paso y observar cómo se comporta en diferentes etapas. Esto nos ayudará a identificar posibles errores en el flujo de ejecución.

Logging y Registro de Errores: Utilizaremos la biblioteca logging de Python para registrar información sobre el estado del programa y los errores en un archivo de registro. Esto nos permitirá rastrear y analizar los problemas que ocurran durante la ejecución.

Gestión de Excepciones: Implementaremos bloques try y except para manejar y controlar las excepciones que puedan surgir durante la ejecución del programa.

Código

```
import logging

# Configuración de logging
logging.basicConfig(filename='registro_errores.log', level=logging.ERROR)

def realizar_calculos(numero):
    try:
        resultado = 10 / numero
        return resultado
    except ZeroDivisionError:
        logging.error("Error: División por cero")
        print("Error: División por cero")
        return None

def main():
    while True:
        try:
            entrada_usuario = input("Ingrese un número (escriba 'exit' para salir): ")
            if entrada_usuario.lower() == 'exit':
                break
            numero_usuario = int(entrada_usuario)
            resultado_calculo = realizar_calculos(numero_usuario)
            if resultado_calculo is not None:
                print(f"El resultado es: {resultado_calculo}")
        except ValueError:
            logging.error("Error: Entrada inválida. Ingrese un número.")
            print("Error: Entrada inválida. Ingrese un número.")

if __name__ == "__main__":
    main()
```

Explicación

Depuradores: Utilizamos un depurador para ejecutar el programa paso a paso, lo que nos permite observar cómo se ejecuta cada línea de código y cómo cambian las variables en tiempo real.

Logging y Registro de Errores: Hemos configurado la biblioteca logging para que registre los errores en un archivo llamado "registro_errores.log". Cada vez que se encuentre un error, se registrará en este archivo, lo que nos permitirá analizarlos posteriormente.

Gestión de Excepciones: Hemos utilizado bloques try y except para manejar errores específicos. En este caso, capturamos la excepción ZeroDivisionError que podría ocurrir si el usuario ingresa 0 como divisor.

```python
import logging

# Configuración de logging
logging.basicConfig(filename='registro_errores.log', level=logging.ERROR)

def realizar_calculos(numero):
    try:
        resultado = 10 / numero
        return resultado
    except ZeroDivisionError:
        logging.error("Error: División por cero")
        return None

def main():
    while True:
        try:
            entrada_usuario = input("Ingrese un número (escriba 'exit' par
            if entrada_usuario.lower() == 'exit':
                break
            numero_usuario = int(entrada_usuario)
            resultado_calculo = realizar_calculos(numero_usuario)
            if resultado_calculo is not None:
                print(f"El resultado es: {resultado_calculo}")
        except ValueError:
            logging.error("Error: Entrada inválida. Ingrese un número.")
            print("Error: Entrada inválida. Ingrese un número.")

if __name__ == "__main__":
    main()
```

```
Ingrese un número (escriba 'exit' para salir): 0
Error: División por cero
Ingrese un número (escriba 'exit' para salir): 5
El resultado es: 2.0
Ingrese un número (escriba 'exit' para salir): hola
Error: Entrada inválida. Ingrese un número.
Ingrese un número (escriba 'exit' para salir):
```

Conclusión

Este ejemplo práctico nos brinda una visión concreta de cómo las herramientas y técnicas de manejo de errores en Python trabajan en conjunto para fortalecer la robustez y fiabilidad de nuestras aplicaciones. Al utilizar depuradores, registros de errores y la gestión de excepciones, podemos identificar, anticipar y resolver problemas de manera eficaz.

Al usar estas herramientas en el código se visualiza que la idea principal es proteger la integridad de nuestro programa que si bien se puede presentar errores o problemas se intenta solucionar todos o la mayoría de los posibles casos.

Bibliografía

- Laprie, J. C. (1985). Dependability: Basic Concepts and Terminology. Springer. DOI: 10.1007/978-0-387-35031-4

- Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, 1(1), 11-33. DOI: 10.1109/TDSC.2004.2

- Bondavalli, A., Di Giandomenico, F., & Trivedi, K. S. (2002). Modeling and Assessment of Fault Tolerance Approaches. IEEE Transactions on Computers, 51(5), 548-560. DOI: 10.1109/TC.2002.1004593

- Chandra, S., Hadzilacos, V., & Toueg, S. (1996). The Weakest Failure Detector for Solving Consensus. Journal of the ACM, 43(4), 685-722. DOI: 10.1145/234533.234549