CUCEI

Centro Universitario de Ciencias Exactas e Ingenierías

TAREA 2 (P1)

Computación Tolerante a Fallas

Clave: I7036 | Sección: D06

Calendario: 2023B

Michel Emanuel López Franco

Cortés Pérez Leobardo Leonel
leobardo.cortes4751@alumnos.udg.mx
Departamento De Ciencias Computacionales

DIVTIC

División de Tecnologías para la Integración Ciber-Humana

Herramientas para el manejo de errores

Objetivo:

Genera un reporte con otras herramientas para el manejo de errores en programación.

Desarrollo:

En C++ moderno, en la mayoría de los escenarios, la manera preferida de notificar y controlar los errores lógicos y los errores en tiempo de ejecución es usar excepciones. Esto es especialmente cierto cuando la pila puede contener varias llamadas a función entre la función que detecta el error y la función que tiene el contexto para controlar el error. Las excepciones proporcionan una forma formal y bien definida para que el código que detecta errores pase la información a la pila de llamadas.

Uso de excepciones para código excepcional

Los errores de programa a menudo se dividen en dos categorías: errores lógicos causados por errores de programación, por ejemplo, un error "índice fuera del intervalo". Y, errores en tiempo de ejecución que están fuera del control del programador, por ejemplo, un error "servicio de red no disponible".

En la programación de estilo C y en COM, los informes de errores se administran devolviendo un valor que representa un código de error o un código de estado para una función determinada, o estableciendo una variable global que el autor de la llamada puede recuperar opcionalmente después de cada llamada a función para ver si se han notificado errores.

Las excepciones se prefieren en C++ moderno por los siguientes motivos:

- Una excepción obliga al código que llama a reconocer una condición de error y controlarla. Las excepciones no controladas detienen la ejecución del programa.

- Una excepción salta al punto de la pila de llamadas que puede controlar el error. Las funciones intermedias pueden dejar que la excepción se propague. No tienen que coordinarse con otras capas.

- El mecanismo de desenredo de la pila de excepciones destruye todos los objetos del ámbito después de iniciar una excepción, según reglas bien definidas.

- Una excepción permite una separación limpia entre el código que detecta el error y el código que lo controla.

El mecanismo de excepción tiene un costo de rendimiento mínimo si no se produce ninguna excepción. Si se produce una excepción, el costo del recorrido y el desenredo de pila es comparable aproximadamente al costo de una llamada a función. Se requieren estructuras de datos adicionales para realizar un seguimiento de la pila de llamadas después de especificar un bloque try, y se requieren instrucciones adicionales para desenredar la pila si se produce una excepción.

Sin embargo, en la mayoría de los escenarios, el costo en el rendimiento y la superficie de memoria no es apreciable. Es probable que el efecto adverso de las excepciones en el rendimiento sea apreciable solo en sistemas con limitaciones de memoria. O bien, en bucles críticos para el rendimiento, donde es probable que se produzca un error con regularidad y haya un acoplamiento estricto entre el código para controlarlo y el código que lo notifica.

En cualquier caso, es imposible conocer el costo real de las excepciones sin generar perfiles y medir. Incluso en esos casos poco frecuentes en que el costo es significativo, puede sopesarlo con respecto al aumento de la exactitud, la facilidad de mantenimiento y otras ventajas proporcionadas por una directiva de excepciones bien diseñada.

Especificaciones de excepciones y noexcept

Las especificaciones de excepciones se introdujeron en C++ como una manera de especificar las excepciones que podría producir una función. Sin embargo, las especificaciones de excepciones demostraron problemas en la práctica y están en desuso en el borrador estándar de C++11.

No se recomienda usar especificaciones de excepciones throw, salvo para throw(), que indica que la función no permite escapar excepciones. Si debe usar especificaciones de excepciones del formato en desuso throw(type-name), la compatibilidad con MSVC es limitada. El especificador noexcept se introduce en C++11 como alternativa preferida a throw().