Programación de aplicaciones — Excepiones

Sistemas Operativos 2024/2025

Jesús Torres

Vamos a ver como manejar los errores de las librerías del sistema usando excepciones en C++.

Contenidos

1. Manejo de errores con excepciones

En C++ y otros lenguajes con orientación a objetos, la forma más común de propagar errores es mediante excepciones. Si estás familiarizado con este concepto y lo prefieres, **puedes utilizar excepciones para gestionar los errores en la práctica**.

Para utilizar excepciones solo necesitas saber que podemos lanzar una excepción para un código de error de `errno` concreto:

```
throw std::system_error(errno, std::system_category(), "Error en foo()");
```

Si, por ejemplo, se lanza una excepción de tipo `std::system_error` desde la función `read_file()` cuando `read()` falla, la excepción se puede capturar para manejarla adecuadamente:

```
try
{
    read_file(int fd, buffer);
}
catch (std::system_error& e)
{
    std::println(std::cerr, "Error ({}): {}", e.code().value(), e.what());
}
```

Las excepciones `std::system_error`[1] guardan el código de error `errno` en el objeto. Llamando al método `code()` del objeto de la excepción –la variable `e` en el ejemplo anterior– se obtiene el objeto `std::error_code` que contiene el código de error con el que se lanzó la excepción. Dicho código se puede obtener con el método `value()` del objeto `std::error_code` devuelto por `code()`:

```
try
{
    // ...
}
catch (std::system_error& e)
{
    int errno_value = e.code().value();                                    ①
    std::println(std::cerr, "Código de error {}", errno_value);
}
```

① Obtener el valor de `errno` con el que se ha lanzado la excepción.

Además, el objeto de la excepción `e` guarda un mensaje de texto descriptivo del error que se puede obtener con el método `what()`. Este mensaje descriptivo es equivalente al que se obtiene con `std::strerror(errno)`, pero al que se le añade la cadena indicada en el tercer argumento del constructor de `std::system_error`.

Este tercer argumento puede ser útil para construir un mensaje donde se indique la línea y el fichero donde se ha producido el error, lo que puede facilitar la localización del problema durante el desarrollo:

```
throw std::system_error(errno, std::system_category(),
    std::format("Lanzado desde '{}' línea {}", __FILE__, __LINE__));       ①
```

① Crear un mensaje de error más informativo para la excepción, con el nombre del fichero –usando el valor de la macro `__FILE__`– y el número de línea –usando el valor de `__LINE__`– donde se ha lanzado la excepción.

```
try
{
    // ...
}
catch (std::system_error& e)
{
    std::println(std::cerr, "Error ({}): {}",
        e.code().value(), e.what());                                       ①
}
```

① Imprimir el mensaje de error de la excepción. En esta caso la cadena devuelta por `what()` incluye el nombre del fichero y el número de línea donde se ha lanzado la excepción.

[1] `std::system_error` se declara en `<system_error>`.

2. Patrón `protected_main()`

Un patrón muy común para hacer el código más legible es separar el manejo de excepciones del resto del código de la función `main()`. Básicamente, consiste en extraer el código de la aplicación que puede lanzar excepciones y ponerlo en una función llamada `protected_main()`:

```
int protected_main(int argc, char* argv[])
{
    // Código de la aplicación...

    read_file(int fd, buffer);

    // Más código de la aplicación...

    return EXIT_SUCCESS;
}
```

Y luego crear una función `main()` con el bloque `try-catch` que llame a `protected_main()` dentro de `try` y gestione las excepciones en `catch`:

```
int main(int argc, char* argv[])
{
    try
    {
        return protected_main();
    }
    catch(std::system_error& e)
    {
        std::println(std::cerr,
            "Error ({}): {}", e.code().value(), e.what());
    }
    catch(std::exception& e)
    {
        std::println(std::cerr,
            "Error: Excepción: {}", e.what());
    }

    return EXIT_FAILURE;
}
```

Esto permite programar libremente en `protected_main()` las tareas y funcionalidades de nuestra aplicación, sabiendo que cualquier excepción será interceptada y tratada adecuadamente en `main()`, antes de salir del programa.