```cpp
#include "GPIO.h"

namespace Hardware
{
    GPIO::GPIO(int number)
    {

        this->number = number;
        gpiopath = GPIOS + NumberToString<int>(number);

        if (!isExported(number, direction, edge))
        {
            ExportPin(number);
            direction = ReadsDirection(gpiopath);
            edge = ReadsEdge(gpiopath);
        }
        usleep(250000);
    }

    GPIO::~GPIO()
    {
        UnexportPin(number);
    }

    int GPIO::WaitForEdge(CallbackType callback)
    {
        threadRunning = true;
        callbackFunction = callback;
        if (pthread_create(&this->thread, NULL, &threadedPollGPIO, static_cast<void*>(this)))
        {
            threadRunning = false;
            throw Exception::FailedToCreateGPIOPollingThreadException();
        }
        return 0;
    }

    int GPIO::WaitForEdge()
    {
        if (direction == Output) { SetDirection(Input); }
        int fd, i, epollfd, count = 0;
        struct epoll_event ev;
        epollfd = epoll_create(1);
        if (epollfd == -1)
        {
            throw Exception::FailedToCreateGPIOPollingThreadException("GPIO: Failed to create epollfd!");
        }
        if ((fd = open((gpiopath + VALUE).c_str(), O_RDONLY | O_NONBLOCK)) == -1)
        {
            throw Exception::GPIOReadException();
        }

        // read operation | edge triggered | urgent data
        ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
        ev.data.fd = fd;

        if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1)
        {
            throw Exception::FailedToCreateGPIOPollingThreadException("GPIO: Failed to add control interface!");
        }

        while (count <= 1)
        {
            i = epoll_wait(epollfd, &ev, 1, -1);
            if (i == -1)
            {
                close(fd);
                return -1;
            }
            else
            {
                count++;
            }
        }
        close(fd);
        return 0;
    }

    GPIO::Value GPIO::GetValue() { return ReadsValue(gpiopath); }
    void GPIO::SetValue(GPIO::Value value) { WritesValue(gpiopath, value);  }

    GPIO::Direction GPIO::GetDirection() {  return direction; }
    void GPIO::SetDirection(Direction direction)
    {
        this->direction = direction;
    }

    GPIO::Edge GPIO::GetEdge() { return edge; }
    void GPIO::SetEdge(Edge edge)
    {
        this->edge = edge;
        WritesEdge(gpiopath, edge);
    }

    bool GPIO::isExported(int number, Direction &dir, Edge &edge)
    {
        // Checks if directory exist and therefore is exported
        if (!DirectoryExist(gpiopath)) { return false; }

        // Reads the data associated with the pin
        dir = ReadsDirection(gpiopath);
        edge = ReadsEdge(gpiopath);
        return true;
    }

    bool GPIO::ExportPin(int number)
    {
        Write(EXPORT_PIN, NumberToString<int>(number));
        usleep(250000);
    }

    bool GPIO::UnexportPin(int number)
    {
        Write(UNEXPORT_PIN, NumberToString<int>(number));
    }


    GPIO::Direction GPIO::ReadsDirection(const string &gpiopath)
    {
        if (Read(gpiopath + DIRECTION) == "in") { return Input; }
        else { return Output; }
    }

    void GPIO::WritesDirection(const string &gpiopath, Direction direction)
    {
        switch (direction)
        {
        case Hardware::GPIO::Input:
            Write((gpiopath + DIRECTION), "in");
            break;
        case Hardware::GPIO::Output:
            Write((gpiopath + DIRECTION), "out");
            break;
        }
    }

    GPIO::Edge GPIO::ReadsEdge(const string &gpiopath)
    {
        string reader = Read(gpiopath + EDGE);
        if (reader == "none") { return None; }
        else if (reader == "rising") { return Rising; }
        else if (reader == "falling") { return Falling; }
        else { return Both; }
    }

    void GPIO::WritesEdge(const string &gpiopath, Edge edge)
    {
        switch (edge)
        {
        case Hardware::GPIO::None:
            Write((gpiopath + EDGE), "none");
            break;
        case Hardware::GPIO::Rising:
            Write((gpiopath + EDGE), "rising");
            break;
        case Hardware::GPIO::Falling:
            Write((gpiopath + EDGE), "falling");
            break;
        case Hardware::GPIO::Both:
            Write((gpiopath + EDGE), "both");
            break;
        default:
            break;
        }
    }

    GPIO::Value GPIO::ReadsValue(const string &gpiopath)
    {
        string path(gpiopath + VALUE);
        int res = StringToNumber<int>(Read(path));
        return (Value)res;
    }

    void GPIO::WritesValue(const string &gpiopath, Value value)
    {
        Write(gpiopath + VALUE, NumberToString<int>(value));
    }


    void* threadedPollGPIO(void *value)
    {
        GPIO *gpio = static_cast<GPIO*>(value);
        while (gpio->threadRunning)
        {
            gpio->callbackFunction(gpio->WaitForEdge());
            usleep(gpio->debounceTime * 1000);
        }
        return 0;
    }
}
```