```rust
use std::{fs, thread, process};
use std::fs::File;
use std::io::{self};
use std::io::{Write, ErrorKind::WouldBlock};
use std::time::{Duration, Instant, SystemTime};
use chrono::{Utc, DateTime}; // MIT license
use scrap::{Capturer, Display}; // MIT licence
use repng; // MIT license


// This program is called using lore-rapid-fire-screenshots.exe
// The licence file will be written to disk each time lore-rapid-fire-screenshots.exe is run

// Copyright 2022 Tarjin Rahman
// Licensed under the MIT License

// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files
// (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge,
// publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do
// so, subject to the following conditions:

// The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
// MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
// FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
// WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

fn main() {
    let start_time = Instant::now();
    let start_time_utc = Utc::now().time();

    const PROGRAM: &str = "lore-subprocess-capture-one-png.exe";
    const VERSION: &str = "v1.0.2022";

    match fs::create_dir_all("./logs/") {
        Err(why) => println!("! {:?}", why.kind()),
        Ok(_) => {
            // nothing
        },
    }

    match fs::create_dir_all("./screenshots/rapid_fire_screenshots/") {
        Err(why) => println!("! {:?}", why.kind()),
        Ok(_) => {
            // nothing
        },
    }

    screenshot();
    println!();
}
```

```rust


fn log_info(message: &str) {

    let mut file = fs::OpenOptions::new()
    .read(true)
    .write(true)
    .create(true)
    .append(true)
    .open("./logs/log.txt")
    .unwrap();

    let system_time = SystemTime::now();
    let datetime: DateTime<Utc> = system_time.into();
    write!(file, "[{} UTC] INFO: {}", datetime.format("%Y-%m-%d %T"), message);
}


fn screenshot() {
    // this function takes a screenshot of the entire desktop display for whatever is currently showing and saves it to a png file

    let capture_start_time = Instant::now();

    while capture_start_time.elapsed() < Duration::from_millis(1000) {
        // this is just one second, but is arbitrary since we will quit after attempting one screenshot regardless of success or failure

        let x = process_screenshot();
        let x = match x {
            Ok(x) => x,
            Err(error) => false
        };

        if x == false {
            let message = "Error capturing screenshot\n";
            log_info(message);
            process::exit(0); // quit program
        }

        process::exit(0); // always quit program after attempting one screenshot regardless of success or failure
    }
}

fn process_screenshot() -> Result<bool, io::Error> {
    let one_second = Duration::new(1, 0);

    let display = Display::primary().unwrap();
    let mut capturer = Capturer::new(display);

    let mut capturer = match capturer {
        Ok(Capturer) => Capturer,
        Err(error) => {
            println!("Error capturing display");
            thread::sleep(one_second);
            return Err(error)
        }
    };

    let (w, h) = (capturer.width(), capturer.height());

    loop {
        // Wait until there's a frame.

        let buffer = match capturer.frame() {
            Ok(buffer) => buffer,
            Err(error) => {
                if error.kind() == WouldBlock {
                    // Keep spinning.
                    thread::sleep(one_second);
                    continue;
                } else {
                    println!("Error buffering frame");
                    return Err(error);
                }
            }
        };

        // Flip the ARGB image into a BGRA image.

        let mut bitflipped = Vec::with_capacity(w * h * 4);
        let stride = buffer.len() / h;

        for y in 0..h {
            for x in 0..w {
                let i = stride * y + 4 * x;
                bitflipped.extend_from_slice(&[
                    buffer[i + 2],
                    buffer[i + 1],
                    buffer[i],
                    255,
                ]);
            }
        }

        // Save the image.
        let timestamp = Utc::now().to_string().replace(":", "_").replace(" ", "_");
        let screenshot_filename = timestamp + &".png";

        repng::encode(
            File::create("./screenshots/rapid_fire_screenshots/".to_string() + &screenshot_filename).unwrap(),
            w as u32,
            h as u32,
            &bitflipped,
        ).unwrap();

        println!("{} saved", &screenshot_filename);

        return Ok(true)
    }
}
```