НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт
із лабораторної роботи №*1*
із дисципліни «Розподілені і хмарні обчислення»

Виконав:
студент групи КМ-03
Шаповалов Г. Г.

Керівник:
Ліскін В. О.

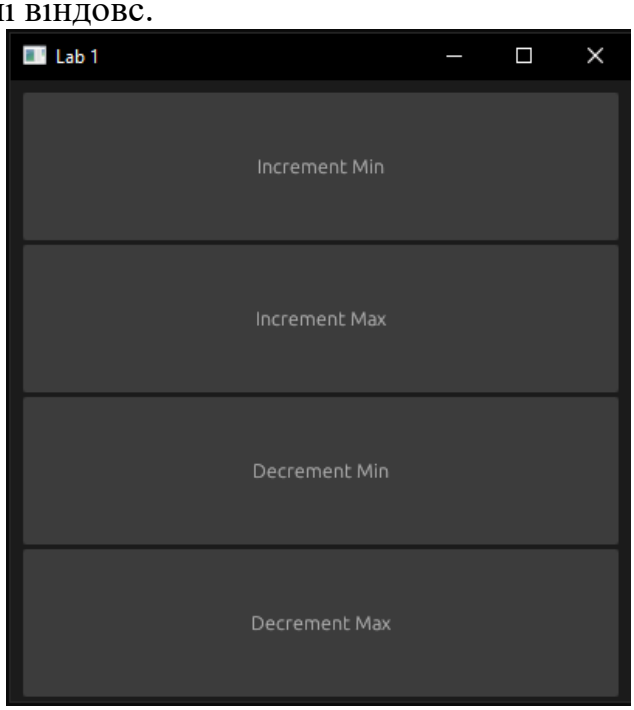
Київ — 2023

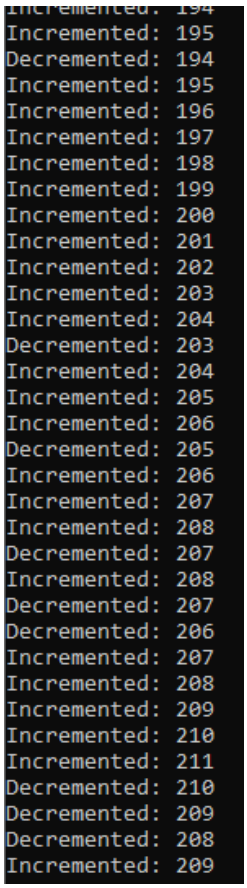
Мета роботи

Навчитись працювати з потоками

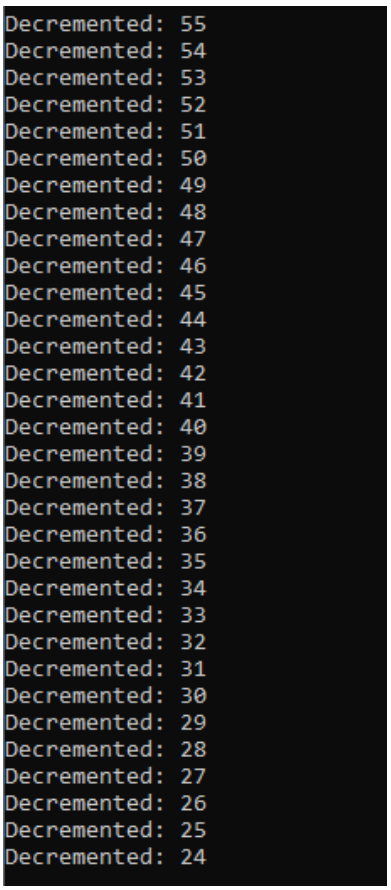
Опис програми

Створено інтерфейс із кнопками для регулювання пріоритетності потоків на операційній системі віндовс.





До натискання Decrement Max



Після

Лістинг програми

```
#[macro_use]
extern crate lazy_static;

use eframe::{egui, epi};
use std::sync::{Arc, Mutex};
use std::thread;
use std::time::Duration;
use thread_priority::*;
use std::os::windows::io::AsRawHandle;
use winapi::ctypes::c_void as winapi_c_void;


lazy_static! {
    static ref PRIORITY_MIN: ThreadPriority =
ThreadPriority::Os(WinAPIThreadPriority::Lowest.into());
    static ref PRIORITY_MAX: ThreadPriority =
ThreadPriority::Os(WinAPIThreadPriority::Highest.into());
}

fn increment(refr: Arc<Mutex<i32>>) {
    loop {
        let mut num = refr.lock().unwrap();
        *num += 1;
        println!("Incremented: {}", num);
        thread::sleep(Duration::from_millis(100));
    }
}

fn decrement(refr: Arc<Mutex<i32>>) {
    loop {
        let mut num = refr.lock().unwrap();
        *num -= 1;
        println!("Decremented: {}", num);
        thread::sleep(Duration::from_millis(100));
    }
}

struct App {
    inc_handle: *mut winapi_c_void,
    dec_handle: *mut winapi_c_void,
}

impl epi::App for App {
    fn name(&self) -> &str {
        "Lab 1"
    }

    fn update(&mut self, ctx: &egui::CtxRef, _frame: &mut epi::Frame<'_>) {
        egui::CentralPanel::default().show(ctx, |ui| {
```

```
            let button_size = egui::vec2(ui.available_width(), 95.0);
            if ui.add_sized(button_size, egui::Button::new("Increment
Min")).clicked() {
                set_thread_priority(self.inc_handle,
*PRIORITY_MIN).expect("Failed to set thread priority");
            }
            if ui.add_sized(button_size, egui::Button::new("Increment
Max")).clicked() {
                set_thread_priority(self.inc_handle,
*PRIORITY_MAX).expect("Failed to set thread priority");
            }
            if ui.add_sized(button_size, egui::Button::new("Decrement
Min")).clicked() {
                set_thread_priority(self.dec_handle,
*PRIORITY_MIN).expect("Failed to set thread priority");
            }
            if ui.add_sized(button_size, egui::Button::new("Decrement
Max")).clicked() {
                set_thread_priority(self.dec_handle,
*PRIORITY_MAX).expect("Failed to set thread priority");
            }
        });
    }
}

fn main() {
    let counter = Arc::new(Mutex::new(0));

    let counter_clone_inc = Arc::clone(&counter);
    let inc_thread = thread::spawn(move || {
        increment(counter_clone_inc);
    });

    let counter_clone_dec = Arc::clone(&counter);
    let dec_thread = thread::spawn(move || {
        decrement(counter_clone_dec);
    });

    let inc_handle = inc_thread.as_raw_handle() as *mut winapi_c_void;
    let dec_handle = dec_thread.as_raw_handle() as *mut winapi_c_void;

    let app = App {
        inc_handle,
        dec_handle,
    };

    let mut native_options = eframe::NativeOptions::default();
    native_options.initial_window_size = Some(egui::vec2(400.0, 400.0));
    eframe::run_native(Box::new(app), native_options);
}
```