Sprawozdanie do zad. 3. w ramach laboratorium Organizacja Systemów Komputerowych

Paweł Szczepański (188640), Igor Szczyrbak (188661)

12 maja 2024

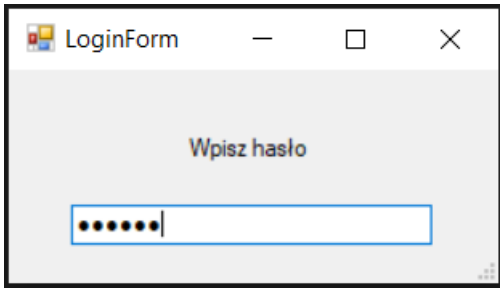
1 Założenia programu

Aplikacja służy do symulowania stanowiska dyspozytora linii produkcyjnej, z naciskiem na zabezpieczenia produkcji. Program uwzględnia logowanie do systemu, kontrolę przytomności operatora, symulację losowych awarii oraz przegrzania układu.

2 Rozwiązania programowe

2.1 Logowanie

Logowanie do systemu odbywa się przy pomocy odrębnego okna. Zawiera ono prośbę o wpisanie hasła oraz pole TextBox pozwalające wpisać hasło i maskujące wpisywane znaki.



Rysunek 1: Okno logowania

Po wciśnięciu klawisza Enter zawartość pola TextBox porównywana jest z hasłem "qwe123".

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if(e.KeyCode == Keys.Enter)
    {
        if (input == password)
        {
            logined = true;
            this.Close();
        }
        else
        {
            label1.Text = "Nieprawidłowe hasło";
            label1.ForeColor = Color.Red;
        }
    }
}
```

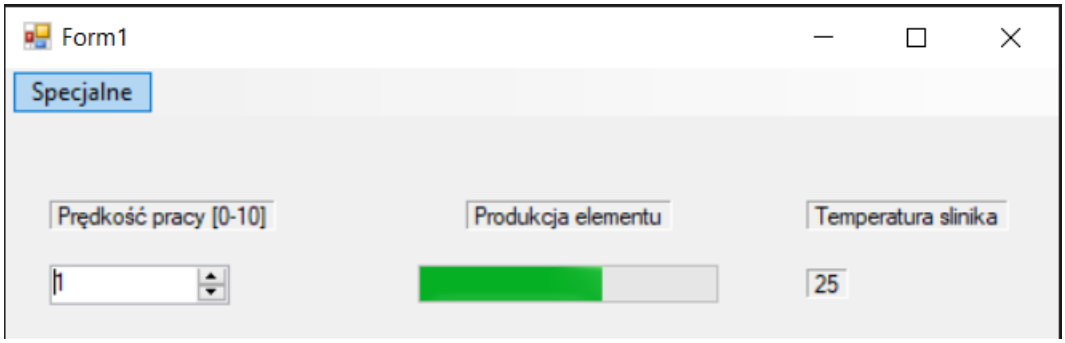
Przy próbie ręcznego zamknięcia okna logowanie zwróci ono DialogResult.Abort co doprowadzi do zamknięcia okna głównego programu.

```
private void Logout()
{
    LoginForm login = new LoginForm(this);
    DialogResult result = login.ShowDialog();

    if (result == DialogResult.Abort)
        this.Close();
}
```

2.2 Symulacja linii produkcyjnej

2.2.1 Normalna praca

Po zalogowaniu program pozwala na interakcję z oknem głównym programu. Zawiera ono informację o postępie produkcji produkowanego elemementu, temperaturze silnika i pozwala na zmianę prędkości pracy w zakresie od 0 do 10. Prędkość pracy wpływa na prędkość produkcji elementu oraz temperaturę silnik (patrz: 2.2.3). Zmiany są wywoływane 10 razy na sekundę przez Timer mainTime. Menu Specjalne służy do symulowania awarii i rozpoczęciu testu przytomności operatora.

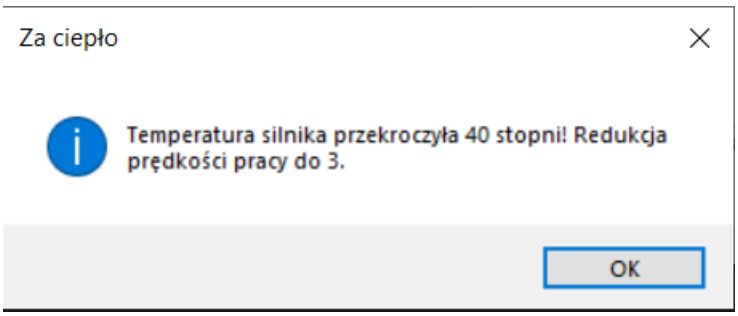


Rysunek 2: Symulacja stanowiska dyspozytora

```
private void mainTime_Tick(object sender, EventArgs e)
{
    int newValue = progressBar1.Value + (int)numericUpDown1.Value * productionRate;
    if (newValue > 100)
        newValue = 0;

    progressBar1.Value = newValue;
}
```

2.2.2 Temperatura silnika i przegrzanie

Temperatura silnika jest równa sumie rzeczywistej temperaturze procesora (przez co program wymaga uprawnień administratora) oraz symulowanej temperatury. Symulowana temperatura Rośnie, gdy prędkość pracy jest większa od 5 i maleje w przeciwnym wypadku do minimalnej wartości 0. Gdy temperatura silnika przekroczy 40 stopni celsjusza prędkość pracy jest zmieniana na 3 i wyświetlany jest odpowiedni komunikat.

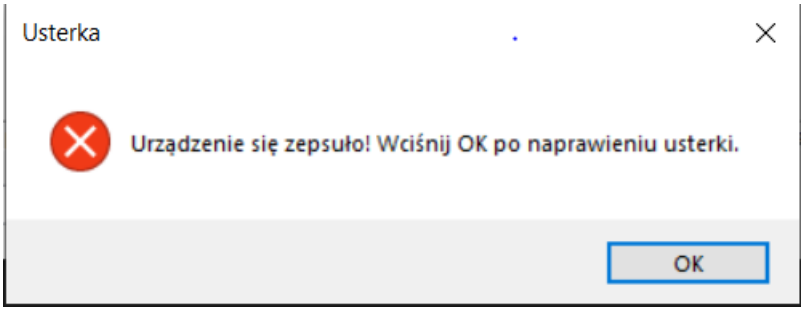


Rysunek 3: Informacja o przegrzaniu

```
private void mainTime_Tick(object sender, EventArgs e)
{
    double CPUtemp = GetProcessorTemp();

    if (numericUpDown1.Value <= 5)
        temperature -= r.NextDouble() * tempChange;
    else
    {
        temperature += r.NextDouble() * tempChange;
        if (temperature + CPUtemp > 40)
        {
            numericUpDown1.Value = 3;
            MessageBox.Show(this, "Temperatura silnika przekroczyła 40 stopni! Redukcja
                prędkości pracy do 3.",
               "Za ciepło", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    temperature = Math.Max(temperature, 0);
    label1.Text = ((int)(CPUtemp + temperature)).ToString();
}

private double GetProcessorTemp()
{
    Double temperature = 0;
    ManagementObjectSearcher searcher = new ManagementObjectSearcher
        (@"root\WMI", "SELECT * FROM MSAcpi_ThermalZoneTemperature");

    try
    {
        foreach (ManagementObject obj in searcher.Get())
        {
            temperature = Convert.ToDouble(obj["CurrentTemperature"].ToString());
            // Convert the value to celsius degrees
            temperature = (temperature - 2732) / 10.0;
        }
    }
    catch
    {
    }
    return temperature;
}
```

2.2.3 Losowe awarie

Przy każdym wywołaniu mainTime_Tick() istnieje niewielka szansa, że urządzenie się zepsuje. W takim wypadku wyświetlany jest stosowany komunikat, a operator jest instruowany o potrzebie naprawy usterki.

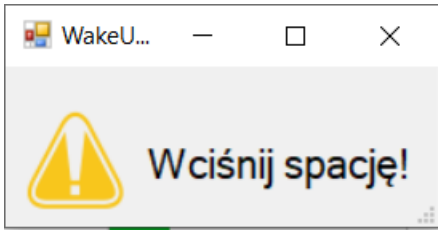


Rysunek 4: Informacja o awarii

```
private void mainTime_Tick(object sender, EventArgs e)
{
    if (r.NextDouble() <= breakChance)
        Break();
}

private void Break()
{
    MessageBox.Show(this, "Urządzenie się zepsuło! Wciśnij OK po naprawieniu usterki.",
        "Usterka", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

2.3 Kontrola przytomności operatora

Kontrola przytomności operatora odbywa się przy pomocy odrębnego okna. Po wywołaniu Timera ustawianego na losową liczbę z zakresu 10-40 sekund rozpoczyna się test przytomności. Polega on na wywołaniu odpowiedzniego okna, jeżeli wciśnie spację w ciągu 20 sekund okno testu zostanie zamknięte. W przeciwnym wypadku operator zostanie wylogowany.



Rysunek 5: Test przytomności operatora

```
private void WakeUp()
{
    WakeUpForm wakeForm = new WakeUpForm(this);
    DialogResult result = wakeForm.ShowDialog();

    if (result == DialogResult.Abort)
        Logout();
```

```
    wakeUp.Interval = r.Next(minWakeUpTime, maxWakeUpTime);
}

private void WakeUpForm_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Space)
    {
        correctKey = true;
        this.Close();
    }
}
```

---

3 Dyskusja osiągniętych wyników

Aplikacja odpowiednio symuluje zabezpieczenie linii produkcyjnej, jednak symulacja samego procesu produkcji jest wysoce abstrakcyjna.