Parallel Task Processor

1. Úvod

Tento projekt vznikl jako školní úloha v předmětu PV – Paralelní programování.
Cílem je ukázat praktické použití více vláken v Pythonu a modelu producent–konzument.
Aplikace umožňuje zadávání textových úloh, jejich paralelní zpracování a výpis výsledků.

---

2. Popis problému

V reálných systémech se často zpracovává více úloh najednou (logování, požadavky na server, práce se soubory).
Sekvenční zpracování bývá pomalé → vhodné řešení je použití **paralelních vláken** a **fronty úloh**.

Tento projekt demonstruje:

- paralelní zpracování úloh,
- thread-safe komunikaci přes frontu,
- synchronizaci sdílených zdrojů.

---

3. Architektura řešení

Aplikace se skládá ze dvou hlavních tříd:

Task

- obsahuje vstupní text,
- generuje unikátní ID,
- metoda `execute()` simuluje práci (uppercase + 1s zpoždění).

TaskProcessor

- obsahuje frontu úloh (Queue),
- vytváří worker vlákna,
- zajišťuje bezpečný zápis výsledků pomocí Locku,
- ukončuje program a zobrazuje výsledky.

Soubory projektu

- `main.py` – konzolové uživatelské rozhraní
- `task_processor.py` – logika zpracování úloh
- `DOKUMENTACE.pdf` – dokumentace

---

4. Paralelismus a synchronizace

Fronta úloh (queue.Queue)

- bezpečný přístup z více vláken,
- jednoduché vkládání a odebírání úloh.

Vlákna (threading.Thread)

- každé vlákno zpracovává úlohy samostatně,
- běží jako daemon.

Lock (threading.Lock)

- chrání sdílený log před souběžným zápisem,
- zabraňuje race conditions.

---

5. Průběh programu

1. Spuštění aplikace.
2. Vytvoření objektu TaskProcessor a worker vláken.
3. Uživatel zadává text → vzniká Task.
4. Task se vloží do fronty.
5. Worker vlákna úlohu vyzvednou, zpracují, uloží výsledek.
6. Po `exit` program:
    - čeká na dokončení fronty,
    - ukončí vlákna,
    - zobrazí seznam výsledků.

---

6. UML – textová verze

Class Task

- id : string
- data : string

- execute() : string

Class TaskProcessor

- task_queue : Queue
- result_log : list

- workers : list
- lock : Lock

- start()
- stop()
- submit(text)
- worker_loop()
- print_results()

---

7. Testování

Projekt byl testován ručně:

- zadání několika textů,
- následné zadání `exit`,
- kontrola správného převodu na uppercase,
- ověření, že pořadí se může lišit (paralelní zpracování).

---

8. Chybové stavy

- prázdný vstup se ignoruje,
- přerušení `Ctrl+C` → bezpečné ukončení,
- vlákna se korektně ukončují pomocí příznaku `running`.

---

9. Možnosti rozšíření

- nové typy úloh,
- ukládání výsledků do souboru,
- statistiky výkonu,
- priority úloh,
- webové API.

---

10. Závěr

Projekt ukazuje jednoduché, ale funkční použití paralelního zpracování v Pythonu.
Je přehledný, stabilní a snadno rozšiřitelný pro budoucí práci.