Parallel Task Processor

1. Úvod

Cílem projektu je vytvořit jednoduchý paralelní procesor úloh v jazyce Python, který využívá reálnou paralelizaci pomocí modulu `multiprocessing`.
Aplikace ukazuje typickou architekturu producent–konzument a komunikaci procesů přes sdílené fronty.

---

2. Popis problému

V praxi je často potřeba zpracovávat více úloh paralelně – např. textové operace, logování, šifrování, práce s daty.
Tento projekt řeší paralelní zpracování řetězců pomocí více procesů, které mohou běžet současně na vícejádrových procesorech.

---

3. Architektura řešení

Projekt využívá dvě hlavní třídy:

Task

- jednotka práce,
- obsahuje ID a vstupní text,
- metoda `execute()` provádí textovou operaci (uppercase).

TaskProcessor

- spravuje frontu úloh (`multiprocessing.Queue`),
- vytváří worker procesy,
- přijímá výsledky přes výstupní frontu,
- řídí ukončování celého systému.

Soubory:

- **main.py** – konzolové rozhraní,
- **task_processor.py** – logika zpracování úloh,
- **DOKUMENTACE.pdf** – dokumentace.

---

4. Paralelismus a komunikace procesů

Aplikace používá:

multiprocessing.Process

Každý worker je samostatný proces → skutečná paralelizace.

multiprocessing.Queue

Slouží pro:

- předávání úloh,
- předávání výsledků.

Queue je bezpečná pro více procesů a nenastávají race conditions.

Ukončení procesů

Do fronty se vloží speciální hodnota `None`, která říká procesu, že má skončit.

---

5. Průběh programu

1. Spuštění aplikace.
2. Vytvoření několika worker procesů.
3. Uživatel zadává texty.
4. Každý text se přidá do fronty jako úloha.
5. Procesy úlohy paralelně zpracují.
6. Program po `exit` vyčká na výsledky a ukončí procesy.
7. Vypíše všechny výsledky.

---

6. UML (textová verze)

Class Task

- id : string
- data : string

- execute() : string

Class TaskProcessor

- task_queue : Queue
- result_queue : Queue
- workers : list

- start()
- submit(text)
- wait_for_all_results()
- shutdown()
- print_results()

---

7. Testování

Projekt byl testován ručně:

- zadávání více textů po sobě,
- kontrola paralelního pořadí výsledků,
- ukončení příkazem `exit`,
- ověření, že procesy korektně skončí.

---

8. Závěr

Aplikace splňuje požadavek na reálné paralelní zpracování.
Používá multiprocessing místo vláken, tudíž obchází GIL a efektivně využívá vícejádrový procesor.
Projekt je jednoduchý, přehledný a vhodný jako základ paralelních systémů.