Zadanie rekrutacyjne CUT - cpu usage tracker

Michał Kukowski

Styczeń 2023

1 Wymagania

- Projekt musi zostać napisany w języku C. W standardzie C99 lub wyższym. Przy czym należy napisać kod tak, aby dało się go skompilować za pomocą gcc jak i clang. Kompilacja nie może zawierać ostrzeżeń! (dla clang flaga -Weverything, dla gcc przynajmniej -Wall -Wextra). Pamiętajmy jednak, że clang ma wiele ostrzeżeń bojkotowanych przez community np, ostrzeżenie za brak ręcznego paddingu. Jeśli potraficie wytłumaczyć (lub znaleźliście), że warning jest po prostu głupi, możecie go wyłączyć,
- Projekt musi posiadać system budowania, najlepiej oparty na Makefile lub CMake. System powinien wspierać dynamiczne zmiany kompilatora (czytanie zmiennej środowiskowej CC).
- Projekt musi posiadać system kontroli wersji, najlepiej git. Należy zamieścić "ładną" historię commitów, tak aby pokazać proces powstawania aplikacji. Repozytorium musi zostać umieszczone na darmowych hostingu gita, np github. Link do Twojego repozytorium wysyłasz do działu HR.
- Aplikacja musi działać poprawnie na dowolnej distrybucji linuxa - np Ubuntu, Arch, Fedora, Debian,
- Aplikacja musi zostać przetestowana pod kątem wycieków pamięci, do tego należy użyć programu valgrind,
- Aplikacja musi posiadać przynajmniej 1 automatyczny test. Może być to test jednostkowy napisany w C (budowany za pomocą make test) lub inny test napisany w C lub w języku skryptowym odpalającym cały program i testujący jakieś zachowanie aplikacji,
- Zadbaj o odpowiednią synchronizację pomiędzy wątkami,
- Na rozmowie będziesz proszony o prezentację swojego rozwiązania (odpalenie, pokazanie kodu) oraz wykonanie drobnej modyfikacji w kodzie. **BĄDŹ NA TO PRZYGOTOWANY**.

2 Opis zadania

Twoim zadaniem jest napisanie prostej aplikacji konsolowej do śledzenia zużycia procesora.

1. Pierwszy wątek (Reader) czyta /proc/stat i wysyła odczytany ciąg znaków (jako raw data lub jako strukturę z polami odczytanymi z pliku np. idle) do wątku drugiego (Analyzer),
2. Wątek drugi (Analyzer) przetwarza dane i wylicza zużycie procesa (wyrażone w %) dla każdego rdzenia procesora widocznego w /proc/stat i wysyła przetworzone dane (zużycie procesora wyrażone w % dla każdego rdzenia) do wątku trzeciego (Printer),
3. Wątek trzeci (Printer) drukuje na ekranie w sposób sformatowany (format dowolny, ważne aby był przejrzysty) średnie zużycie procesora co sekunde,
4. Wątek czwarty (Watchdog) pilnuje aby program się nie zawiesił. Tzn jeśli wątki nie wyślą informacji przez 2 sekundy o tym, że pracują to program kończy działanie z odpowiednim komunikatem błędu,

5. Wątek piąty (Logger) przyjmuje wiadomości od wszystkich wątków i zapisuje wiadomości do pliku. Loggera używa się do zapisywania debug printów do pliku w sposób zsynchronizowany,

6. Należy także zaimplementować przechwytywanie sygnału SIGTERM i zadbać o odpowiednie zamknięcie aplikacji (zamknięcie pliku, zwolnienie pamięci, zakończenie wątków).

Do poprawnego działania programu potrzebne są minimum wątki 1 (Reader), 2 (Analyzer) i 3 (Printer). Zatem należy zaimplementować minimum te funkcjonalności. Wątek 4 (Watchdog) i 5 (Logger) jak i przechwytywanie sygnału są opcjonalne, jednak zachęcam do implementacji całości zadania.

3 Wskazówki

1. Do obliczania zużycia procesora można użyć tej formuły,

2. Pamiętaj, że procfs jest systemem plików wirtualnych. Oznacza to, że ich rozmiar będzie zawsze 0, a każdy odczyt pliku może zwrócić inne wartości (nawet różną długość). Przeanalizuj strukturę pliku /proc/stat i zaprojektuj bezpieczne czytanie z tego z pliku,

3. Wysyłanie danych pomiędzy wątkami nie trzeba rozumieć dosłownie. Można użyć do tego globalnej pamięci (np globalna tablica lub struktura),

4. Czytanie danych i wysyłanie do innego wątku to problem Producenta i konsumenta. Warto zatem poczytać o tym problemie i przeanalizować jego rozwiązania. Stosowanie roziwązań, które nie dają odetchnąć procesorowi (np. sprawdzają co chwila, czy kolejka nie jest pusta) nie są najlepszym pomysłem i nie będą akceptowalne,

5. Pomyśl o buforowaniu danych, co jeśli wątek czytający z pliku czyta szybciej niż przetwarzający? Spróbuj rozwiązać ten problem za pomocą jakieś struktury danych (RingBuffer / Queue),

6. Wielowątkowość umożliwia nam znana biblioteka pthread lub nakładka na tę bibliotęke wbudowana w język C11 i wyżej,

7. Przechwytywanie sygnału SIGTERM może być wzorowane na tym przykładzie,

8. Pamiętaj, że oceniamy nie tylko poprawność ale również styl programowania. Przejrzyj książke Nowoczesne C i zastosuj się do rad zawartych w tej książce. Pamiętaj, że język C jest tylko z pozoru trywialny, pobaw się ficzerami C99 takimi jak VLA, FAM, Compund Literals. Jednak pamiętaj, ze skomplikowane ficzery języka to broń obosieczna. Zanim czegoś użyjesz poznaj wady i zalety tego rozwiązania,

9. Staraj się napisać kod w duchu paradygmatu obiektowego, zachowaj zasady KISS, DRY oraz SOLID w swoim kodzie,

10. Do pisania testów jednostkowych, nie potrzebujesz żadnej dodatkowej biblioteki. Proste testy możesz napisać z użyciem wbudowanej funkcji assert(),

11. Aby umożliwić sobie pisanie testów, najlepiej podzielić program na części odpowiedzialne za logikę oraz część aplikacyjną odpalającą funkcje z poziomu wątków. Np Kolejka to osobna biblioteka. Funkcje zliczające liczbę procesorów, czytające z dowolnego pliku (w szczególności z proc) to też osobne moduły programu. Każdy moduł będzie miał swoje testy, a dopiero otestowane funkcje z danych modułów będą wykorzystane w wątkach aplikacji,

12. Jeśli już zdecydowałeś się na implementację loggera, zadbaj o to aby wysyłać do niego wiadomości, które serio pomogą Ci w debugowaniu problemu. Np watchdog kończy pracę aplikacji, wyślij odpowiedni komunikat, dlaczego to się stało, który wątek się zawiesił. Każda ważna informacja, która pomoże w debugowaniu powinna zostać wysłana do loggera. Oczywiście to Ty decydujesz co jest dla Ciebie przydatne a co nie.