Systemy operacyjne

Laboratorium 4 - synchronizacja procesów przy użyciu monitorów

Prowadzący: mgr. inż. Aleksander Pruszkowski

Zdający: Jakub Sikora

Termin: wtorek 14:15 - 16:00

1. Treść zadania

Napisz w C++ dla środowiska Linux, system kolekcjonowania krótkich wiadomości tekstowych (maks. 128 znaków, ale nie mniej niż 16 znaków). Dla przypomnienia system ma bazować na synchronizacji dostępu do zasobów wykorzystujący mechanizm monitorów.

Zadaniem budowanego systemu ma być niezawodne zbieranie od klientów wiadomości, liczba klientów może być duża, ale system musi być gotowy do obsłużenia minimum 5 klientów.

Klienci - pojedynczy pod-proces lub wątek - „wrzucają" wiadomości do systemu, oprócz samej treści wiadomości wybierają priorytet wrzucanej wiadomość (np.: 0-zwykły, 1->priorytetowy).

System może zbierać wiadomości tylko w jednym pojemnym buforze. Mechanizm wkładania nowych wiadomości do tego bufora musi uwzględniać priorytety. Wszelkie operacje na buforze powinny być optymalizowane w taki sposób by nie kopiować niepotrzebnie wiadomości, oraz czas wkładania wiadomości oraz czas wyjmowania były możliwie jak najkrótsze.

Dodatkowo dla systemu utworzony ma być pod-proces lub wątek „czytelnik" zebranych wiadomości. Jego zadaniem jest pobieranie z bufora i przedstawianie wiadomości tekstowych na konsoli tekstowej. Zakłada się, że „czytelnik" będzie pobierał wiadomości z bufora, a w buforze wiadomości będą już poukładane zarówno względem priorytetów jaki i czasu ich włożenia.

Przemyśl bardzo dokładnie metodę testowania powstałego systemu, w szczególności zwróć uwagę na pokazanie równoczesnego działania wielu procesów umieszczających wiadomości w tym z różnymi priorytetami oraz współdziałanie w tym czasie „czytelnika".

Załóż, że program testowy będą działały automatycznie generując przez klientów fikcyjne wiadomości wyłącznie tekstowe, a „czytelnik" pokazywał je na konsoli.

2. Propozycja rozwiązania

Bufor - monitor

W pierwszej kolejności zaimplementuje klasę bufora `Buffer` dziedziczącą/agregujący funkcjonalności klasy `Monitor` zaimportowanej z pliku `monitor.h` dostarczonego przez wykładowcę. Klasa będzie oferowała na zewnątrz trzy metody:

- `pop` - pobranie pierwszej w kolejności (z uwzględniem priorytetów) wiadomości
- `push` - wstawienie wiadomości zwykłej
- `push_pri` - wstawienie wiadomości priorytetowej

Każda z udostępnianych metod będzie wykorzystywać API klasy `Monitor` tak aby zapewnić synchronizowany dostęp - tylko jeden użytkownik może pracować na buforze.

Bufor komunikacyjny zostanie zrealizowany jako pojedyncza struktura przechowująca wiadomości. W pierwszej wersji projektu jako wewnętrzną strukturę danych zastosuję binarną stertę lub listę łączoną (jedno- lub dwukierunkową).

Użytkownicy

Użytkownikami systemu będą wątki procesu głównego. Założenie to znacząco ułatwia implementację struktury danych, która w tym przypadku może znaleźć się w pamięci procesu a nie w pamięci współdzielonej. Oddelegowany zostanie jeden wątek czytelnika, który z określoną częstotliwością będzie pobierał wiadomości ze struktury. Kolejne wątki (minimum 5) będą wpisywać wiadomości tekstowe. Wątki nie będą przeprowadzały żadnych czynności synchronizacyjncyh, nad wszystkim będzie czuwać sama struktura z monitorem.

3. Zbiór testów rozwiązania

Każdy nowy użytkownik, będzie wysyłał nową wiadomość co określony kwant czasu za pomocą procesu uruchomionego w tle. Każdy z użytkowników będzie wysyłał wiadomości z różnymi częstotliowściami.

W ramach testów przygotuje scenariusze testowe, podobne do tych z zadania trzeciego. Testy będą pokrywały następujące scenariusze:

- pisanie do pełnego bufora
- czytanie z pustego bufora
- zachowanie kolejności wiadomości
- poprawna obsługa wiadomości priorytetowych
- głodzenie procesów