Systemy rozproszone 2021 - Lista 4

Radosław Zazulczak

14 kwietnia 2021

Zadanie 1

W wielu protokołach warstwowych każda warstwa ma własny nagłówek. Z pewnością bardziej efektywne do używania wszystkich tych osobnych nagłówków byłoby poprzedzenie komunikatu jednym nagłówkiem zawierającym całość informacji sterującej. Dlaczego się tego nie robi?

Wyobraźmy sobie sytuację, w której przesyłany pakiet miałby jeden nagłówek (posiadający całość informacji sterującej). Z uwagi na wiele różnych od siebie protokołów sieciowych taki nagłówek musiałby posiadać informacje dla każdego dostępnego protokołu. Wtedy przesłanie pakietu wiązałoby się z tym, że jego nagłówek posiadałby mnóstwo zbędnych danych (przesyłany pakiet między dwoma komputerami połączonymi siecią Ethernet musiałby posiadać również dane z innego nagłówka warstwy łącza danych).
Kolejną wadą upakowania wszystkich informacji w jednym nagłówku jest brak możliwości łatwej modyfikacji bądź wymiany jednego protokołu. Zmiana jednego protokołu np. IPv6 powoduje, że cały nasz jeden nagłówek staje się w pewnym sensie nieaktualny, ponieważ nie uwzględnia on nowego protokołu bądź zmian w nim wprowadzonych. Zatem używanie osobnych nagłówków dla różnych warstw pozwala na modyfikację jednego dowolnego protokołu bez ingerencji w pozostałe.

Zadanie 2

Dlaczego usługi komunikacyjne warstwy transportu są często nieodpowiednie do budowania aplikacji rozproszonych?

Usługi komunikacyjne warstwy transportowej mają wiele niedogodności. W przypadku protokołu TCP, który zapewnia nam niezawodne działania, wadą okazuje się duży koszt, który powoduje protokół. Jest to szczególnie odczuwalne w sieciach zawodnych. Aby zminimalizować koszt (narzut) protokołu TCP można go zastąpić protokołem UDP. W tym przypadku musimy sami dodatkowo zadbać m.in. o gubione pakiety, kontrolę błędów itp. To wymaga znacznie więcej pracy od twórcy aplikacji.

Zadanie 4

Rozważ procedurę *zwiększ* z dwoma liczbami całkowitymi jako parametrami. Procedura dodaje liczbę 1 do każdego parametru. Załóżmy teraz, że wywołano ją z tą samą zmienną w obu parametrach, np. *zwiększ[i,i]*. Przyjmując, że zmienna i ma na początku wartość 0, określ ile wyniesie jej wartość, jeśli zastosuje się przekazywanie przez odniesienie? A co się stanie, gdy parametry będą przekazywane przez kopiowanie-odtwarzanie?

Zobaczmy to na przykładzie poniższego kodu. W funkcji *zwieksz* przekazujemy parametry przez kopiowanie, a w *zwieksz2* przekazujemy przez odniesienie (referencję).

```
#include <iostream>
void zwieksz(int i, int j) {
    ++i;
    ++j;
}
void zwieksz2(int& i, int& j) {
    ++i;
    ++j;
}
int main() {

    int a = 0;
    zwieksz2(a,a);
    std::cout << "Zwieksz2:" << a << std::endl;

    int b = 0;
    zwieksz(b,b);
    std::cout << "Zwieksz:" << b << std::endl;
    return 0;
}
```

Po wykonaniu powyższego kodu a wynosi 2, b wynosi 0.

Jeśli wywołując procedurę *zwieksz[i,i]* zastosujemy przekazywanie przez odniesienie, wartość zmiennej i wyniesie 2, ponieważ dwa razy będziemy inkrementować tą samą zmienną (działamy na oryginalnej zmiennej, a nie na jej kopii). W przypadku przekazywania przez kopiowanie, wartość zmiennej i wyniesie 0, ponieważ zostaną utworzone dwie kopie i, które zostaną zinkrementowane. Te utworzone kopie będą zmiennymi lokalnymi, więc po powrocie z funkcji *zwieksz* wartość zmiennej i nadal będzie 0.

Zadanie 11

Wyjaśnij różnicę między elementarnymi operacjami *MPI_bsend* i *MPI_isend* w MPI.

Na podstwie:
Van Steen M., Tanenbaum A.S. „Distributed Systems" 3rd edition 2017;
Tanenbaum A.S., van Steen M „Systemy rozproszone. Zasady i paradygmaty" 2007

W przeszłości sieci połączeń oraz multikomputery były dostarczane z gotowymi bibliotekami do przekazywania komunikatów. Takie biblioteki oferowały zazwyczaj efektywne operacje, lecz nie potrafiły ze sobą współpracować. Stworzono zatem standard MPI, czyli **interfejs przekazywania komunikatów** (Message-Passing Interface). Interfejs **MPI** tworzą podstawowe operacje do przesyłania komunikatów.

- *MPI_bsend* dzięki tej operacji osiągniemy przejściową komunikację asynchroniczną. Komunikat jest kopiowany do lokalnego bufora systemu wykonawczego. W ten sposób nadawca może działać dalej.
- *MPI_isend* przeznaczenie tej operacji jest podobne do *MPI_bsend* z tą różnicą, że w buforze systemu wykonawczego umieszczany jest tylko wskaźnik do komunikatu. W ten sposób unika się kopiowania całego komunikatu z bufora użytkownika do bufora systemu wykonawczego MPI.

Zadanie 18

Wyjaśnij, dlaczego przejściowa komunikacja synchroniczna jest sama przez się przyczyną problemów ze skalowalnością i jak można temu zaradzić.

Dla przypomnienia w komunikacji synchronicznej nadawca jest blokowany aż do momentu, w którym zostanie doręczony komunikat lub tak długo jak komunikat jest przechowywany w lokalnym buforze. W komunikacji przejściowej komunikat może zostać odrzucony jeśli nie może zostać przekazany dalej.
W przypadku małych sieci lokalnych przejściowa komunikacja synchroniczna sprawdza się dobrze. Jeśli jednak mamy sieć o dużej ilości maszyn, to takie rozwiązanie nie sprawdzi się. Budowanie rozległego systemu tylko za pomocą komunikacji synchronicznej spowoduje, że np. żądanie klienta będzie blokowało go aż do otrzymania odpowiedzi, co może potrwać dość długo. Komunikat będzie przekazywany do kolejnych maszyn i działania każdej z nich będzie blokowane aż do np. potwierdzenia odbioru. To rodzi poważne problemy ze skalowalnością systemu (jego responsywność będzie niska).
Aby zapobiec problemom skalowalności zazwyczaj łączy się kilka rodzajów typów komunikacji m.in. trwałą komunikacje asynchroniczną i synchroniczną oraz przejściowe odmiany komunikacji synchronicznej.