|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **L.p.** | **Nazwa algorytmu:** | **Twórca algorytmu:** | **Ogólna idea:** | |
| **Opis:** | **Przykład:** |
| **1.** | Lamporta | Leslie Lamport | - w celu uzyskania dostępu do sekcji krytycznej proces musi mieć pewność, że jego żądanie jest najstarsze w systemie i że wiedzą o tym wszystkie inne procesy konfliktów równoczesnych,    - czas jest wyznaczany na podstawie zegara skalarnego Lamporta pełniącego rolę priorytetu w rozstrzyganiu konfliktów, |  |
| **2.** | Ricarta i Agrawali | Ashok Agrawala,  Glenn Ricart | - nie potrzeba utrzymywać kolejki,  - rozstrzyganie o pierwszeństwie może następować dynamicznie – przez wstrzymywanie odpowiedzi dla „przegrywającego”,  - uzyskuje się dzięki temu „wirtualną kolejkę”,  - procesy „przegrywające” rywalizację są wstrzymywane,  aż do zakończenia sekcji krytycznej przez proces wygrywający, |  |
| **3.** | Maekawy | Maekawy | - przecięcia zbiorów są niepuste (wymagane dla zapewnienia wzajemnego wykluczania),  - każdy węzeł należy do własnego zbioru (w celu zmniejszenia liczby wiadomości),  - zbiory mają taki sam rozmiar (każdy węzeł wykonuje równą ilość pracy),  - dowolny węzeł należy do takiej samej liczby różnych  zbiorów arbitrażu (równa “odpowiedzialność” w udzielaniu zgody - każdy potrzebuje zgody od takiej samej liczby węzłów), |  |
| **4.** | Raymonda | Raymond | - używa struktury drzewa,  - korzeniem drzewa jest proces, który posiada żeton pozwalający na wejście do sekcji krytycznej,  - każdy proces dysponuje zmienną posiadacz, która wskazuje na kolejny proces na ścieżce prowadzącej do korzenia drzewa,  - struktura zmienia się dynamicznie w zależności od posiadacza żetonu,  - każdy proces w drzewie przechowuje kolejkę żądań sąsiednich procesów, które nie posiadały jeszcze żetonu, |  |
| **5.** | Dijkstry | Edsger Wybe Dijkstra | - wyszukuje najkrótsze ścieżki z jednego źródła do wszystkich pozostałych węzłów grafu,  - przebieg algorytmu:  - dopóki zbiór Q nie jest pusty wykonuj: - pobierz ze zbioru Q wierzchołek v o najmniejszej -wartości D[v] i usuń go ze zbioru. - dla każdego następnika i wierzchołka v dokonaj relaksacji ścieżki, tzn. sprwdź, czy D[i]>D[v]+A[v,i], tzn. czy aktualne oszacowanie odległości do wierzchołka  i jest większe od oszacowania odległości do wierzchołka v plus waga krawędzi (v,i). Jeżeli tak jest, to zaktualizuj oszacowanie D[i] przypisując mu prawą stronę nierówności (czyli mniejszą wartość). |  |
| **6.** | Petersona i Lamporta | Gary L. Peterson,  Leslie Lamport | - umożliwia dwóm [procesom](https://pl.wikipedia.org/wiki/Proces_%28informatyka%29) lub [wątkom](https://pl.wikipedia.org/wiki/W%C4%85tek_%28informatyka%29) bezkonfliktowy dostęp do współdzielonego zasobu ([sekcji krytycznej](https://pl.wikipedia.org/wiki/Sekcja_krytyczna)), | while numer ≠ j do  nic;  sekcja krytycznaj;  numer := i;  resztaj; |
| **7.** | Suzuki-Kasami | Suzuki-Kasami | - wykorzystywany jest żeton, o który ubiegają się procesy chcące wejść do sekcji krytycznej,  - proces, który posiada żeton może wchodzić do sekcji krytycznej do czasu, gdy nie poprosi o niego inny proces,  - pojawiają się problemy, co zrobić ze: starymi (przedawnionymi) żądaniami, zaległymi żądaniami, | |
| **8.** | Dekkera | Theodorus Jozef Dekker | - tylko jeden z procesów może w danej chwili wykonywać ich wspólną [sekcję krytyczną](https://pl.wikipedia.org/wiki/Sekcja_krytyczna),  - pozwala dwóm wątkom na bezkonfliktową pracę na danych pochodzących z jednego źródła przy użyciu do komunikacji między nimi jedynie [pamięci dzielonej](https://pl.wikipedia.org/wiki/Pami%C4%99%C4%87_dzielona), | |
| **9.** | Morrisa | Morris | - rozwiązuje problem wzajemnego wykluczania skończonej liczby procesów (nie dopuszczając do zagłodzenia) przy użyciu semaforów binarnych,  - wady operacji semarofowych: brak wpływu jednego procesu na zakończenie działania innego procesu, brak możliwości zawieszenia procesu na określony przedział czasu, | |
| **10.** | Holta | Charles C. Holt | - służy do detekcji zakleszczenia procesów korzystających z zasobów systemu,- w algorytmie Holt'a przekształcamy macierz rang do trójwymiarowej macierzy *E*, w której znajdują się posortowane (wierszami) żądania z macierzy rang *H* wraz z numerami procesów, | |
| **11.** | Szymański | Bolesław Szymański | - wzorowany na poczekalni z drzwiami wejściowymi oraz wyjściowymi,  - początkowo drzwi wejściowe są otwarte natomiast wyjściowe są zamknięte  - wszystkie procesy żądające dostępu do sekcji krytycznej mniej więcej w tym samym czasie wchodzą do poczekalni, ostatni z nich zamyka drzwi wejściowe i otwiera drzwi wyjściowe, procesy wchodzą de sekcji krytycznej jeden po drugim (lub w większych grupach, jeśli krytyczny punkt na to zezwala), ostatni proces opuszczający sekcję krytyczną zamyka drzwi wyjściowe i ponownie otwiera drzwi wejściowe, tak aby umożliwić wejście kolejnej partii procesów, | |
| **12.** | Haberman'a | Haberman | - służy do detekcji zakleszczenia procesów korzystających z zasobów systemu.  - detekcja zakleszczenia:  1. Zainicjuj zbiór D:={1,2,...,n} oraz wektor zasobów wolnych f.  2. Szukaj zadania o indeksie j należącym do zbioru D takiego, że: H(Pj)≤f  3. Jeżeli zadanie takie nie istnieje, to zbiór zadań odpowiadający zbiorowi D jest zbiorem zadań zakleszczonych.  4. W przeciwnym razie podstaw: D:=D-{j}; f:=f+A(Pj).  5. Jeżeli zbiór D jest pusty zakończ wykonywanie algorytmu. W przeciwnym razie przejdź do kroku 2. | |

Tabela 1. Tabelaryczne zestawienie algorytmów praktycznie rozwiązujących wzajemne wykluczenie wraz z ich twórcami.

**Tabelaryczne zestawienie technologii rozproszonych obecnych na rynku:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **L.p.** | **Technologia:** | **Skrót:** | **Warstwa:** | **Przykłady implementacji:** | **Źródła:** |
| **1.** | Remote Method Invocation | RMI | middleware | Java, C# | [http://docs.oracle.com/javase/1.5.0/docs/guide/rmi/hello/hello-word.html](http://docs.oracle.com/javase/1.5.0/docs/guide/rmi/hello/hello-word.html%20) (dostęp 07.06.2016) |
| **2.** | Gniazda | (sockets) | middleware | C | [http://www.linuxhowtos.org/C\_C++/socket.html](http://www.linuxhowtos.org/C_C++/socket.html%20) (dostęp 07.06.2016) |
| **3.** | Remote Procedure Call | RPC | middleware | C, Java | [http://docs.oracle.com/javase/1.5.0/docs/guide/rmi/hello/hello-word.html](http://docs.oracle.com/javase/1.5.0/docs/guide/rmi/hello/hello-word.html%20) (dostęp 07.06.2016) |
| **4.** | Common Object Request Broker Architecture | CORBA | middleware | ORBit, OMNIErb | [http://www.opengroup.org/openbrand/prodstds/x98or.html](http://www.opengroup.org/openbrand/prodstds/x98or.html%20%20)   (dostęp 07.06.2016) |
| **5.** | Distributed Component Object Model | [DCOM](https://pl.wikipedia.org/wiki/DCOM) | middleware | C++ | COM/DCOM Product Documentation "The COM/DCOM Reference" |
| **6.** | Distributed Computing Environment | [DCE](https://pl.wikipedia.org/w/index.php?title=Distributed_Computing_Environment&action=edit&redlink=1) | middleware | Java | [http://searchnetworking.techtarget.com/definition/DCE](http://searchnetworking.techtarget.com/definition/DCE%20) (dostęp 07.06.2016) |

Tabela 2. Zestawienie technologii rozproszonych obecnych na rynku.