Systemy Operacyjne - synchronizacja i komunikacja procesów

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 1 listopada 2020

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
    1. Sytuacje hazardowe
    2. Problem sekcji krytycznej
    3. Warunki poprawności rozwiązania
    4. Rozwiązanie programowe dla dwóch procesów
    5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
    1. Niepodzielne rozkazy
    2. Semafory
    3. Regiony krytyczne
    4. Monitory
3. Klasyczne problemy synchronizacji
    1. Problem ograniczonego buforowania
    2. Problem czytelników i pisarzy
    3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
    1. Komunikacja pośrednia i bezpośrednia
    2. Buforowanie
    3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
    1. Sytuacje hazardowe
    2. Problem sekcji krytycznej
    3. Warunki poprawności rozwiązania
    4. Rozwiązanie programowe dla dwóch procesów
    5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
    1. Niepodzielne rozkazy
    2. Semafory
    3. Regiony krytyczne
    4. Monitory
3. Klasyczne problemy synchronizacji
    1. Problem ograniczonego buforowania
    2. Problem czytelników i pisarzy
    3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
    1. Komunikacja pośrednia i bezpośrednia
    2. Buforowanie
    3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Plan wykładu

1. Synchronizacja
   1. Sytuacje hazardowe
   2. Problem sekcji krytycznej
   3. Warunki poprawności rozwiązania
   4. Rozwiązanie programowe dla dwóch procesów
   5. Rozwiązania programowe dla wielu procesów
2. Środki synchronizacji
   1. Niepodzielne rozkazy
   2. Semafory
   3. Regiony krytyczne
   4. Monitory
3. Klasyczne problemy synchronizacji
   1. Problem ograniczonego buforowania
   2. Problem czytelników i pisarzy
   3. Problem ucztujących filozofów
4. Sieci Petriego
5. Komunikacja międzyprocesowa
   1. Komunikacja pośrednia i bezpośrednia
   2. Buforowanie
   3. Obsługa wyjątków

Wprowadzenie

W systemach wielozadaniowych, podczas realizacji przez ustaloną liczbę procesów dostępu współbieżnego do dzielonych zasobów może dojść do *sytuacji hazardowych*, nazywanych również *wyścigiem* (ang. *race condition*), które prowadzą do błędów przetwarzania. Zjawiska te pojawiają się, kiedy następuje *przeplot operacji* modyfikacji lub operacji modyfikacji i odczytu stanu współdzielonego zasobu, przy czym operacje te pochodzą od różnych procesów. Jeśli dostęp do zasobu ogranicza się wyłącznie do odczytu, to jest dostępem zawsze bezpiecznym, nawet jeśli dochodzi od przeplotu operacji. Sytuacje hazardowe występują zarówno w przypadku procesów, jak i wątków (choć w dalszej części wykładu będziemy posługiwać się głównie pojęciem procesu). Obojętnym jest również, czy system komputerowy, w którym są wykonywane procesy jest wyposażony w jeden, czy większą liczbę procesorów.

Przykład

Rozważmy dwa procesy, które chcą zmodyfikować wartość współdzielonej zmiennej, przy czym pierwszy chce tę wartość zwiększyć o jeden, a drugi zmniejszyć o jeden. Przyjmijmy, że wartość początkowa zmiennej wynosi 5. Zakładamy, że pojedynczy proces, aby zmodyfikować wartość zmiennej musi ją najpierw pobrać z pamięci operacyjnej, zapisać w rejestrze [1], wykonać właściwą operację, a następnie zapisać wynikową wartość do pamięci. Procesy wykonujące wspomniane operacje mogą utracić procesor w każdej chwili, w wyniku zadziałania mechanizmu wywłaszczającego. Następne plansze prezentują dwa z możliwych scenariuszy wykonania wcześniej opisanych modyfikacji wartości wspólnej zmiennej (kropki oznaczają, że proces jest w danej chwili nieczynny).

[1] Proszę zauważyć, że ze względu na to iż w wyniku przełączenia kontekstu stany rejestrów są zapamiętywane, to procesy mogą posługiwać się tym samym rejestrem

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz pierwszy

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. Zwiększ wartość rejestru R1 o jeden. (R1=6)
3. ...
4. ...
5. Zapisz zawartość rejestru R1 do pamięci. (M=6)
6. ...

Proces drugi

1. ...
2. ...
3. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=4)

Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Scenariusz drugi

Proces pierwszy

1. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
2. ...
3. Zwiększ zawartość rejestru R1 o jeden. (R1=6)
4. ...
5. ...
6. Zapisz zawartość rejestru R1 do pamięci. (M=6)

Proces drugi

1. ...
2. Odczytaj wartość z pamięci (M=5) i umieść ją w rejestrze R1. (R1=5)
3. ...
4. Zmniejsz zawartość rejestru R1 o jeden. (R1=4)
5. Zapisz zawartość rejestru R1 do pamięci. (M=4)
6. ...

Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

Sekcja krytyczna

Fragment kodu, podczas którego realizacji proces wykonuje dostęp do zasobów współdzielonych nazywamy *sekcją krytyczną*. Zasoby te mogą być zarówno zasobami fizycznymi, jak i logicznymi, mogą mieć prostą budowę (jak zmienne prostych typów) lub złożoną (jak struktury danych). Jak wynika ze wcześniejszych rozważań, aby dostęp do zasobu współdzielonego był bezpieczny musimy zagwarantować niepodzielność wykonania przez procesy sekcji krytycznych. Inaczej: jeśli jeden z procesów korzystających z zasobu dzielonego rozpoczął wykonywanie sekcji krytycznej, to żaden z pozostałych procesów nie może rozpocząć wykonywania sekcji krytycznej dotyczącej tego samego zasobu, dopóki ten pierwszy jej nie skończy. Rozpoczynanie sekcji krytycznej określamy mianem *wchodzenia do sekcji krytycznej* zaś jej kończenie *wychodzeniem* lub *opuszczaniem sekcji krytycznej*. Część kodu bezpośrednio poprzedzającą sekcję krytyczną nazywamy *sekcją wejściową*, natomiast część umieszczoną bezpośrednio za sekcją krytyczną określamy mianem *sekcji wyjściowej*. Pozostałą część kodu procesu będziemy nazywać *resztą*.

Warunki poprawności rozwiązania problemu sekcji krytycznej

Każde rozwiązanie problemu sekcji krytycznej musi spełniać trzy warunki, aby być w pełni poprawnym:

- **Wzajemne wykluczanie (ang. *mutual exclusion*)** W danym czasie, w sekcji krytycznej może znajdować się tylko jeden proces.
- **Postęp** Jeśli nie wymaga tego warunek wzajemnego wykluczania, to proces nie powinien być wstrzymywany przed wejściem do sekcji krytycznej.
- **Ograniczone czekanie** Oczekiwanie każdego procesu na wejście do sekcji krytycznej powinno kiedyś się zakończyć. Inne procesy nie mogą wstrzymywać go w nieskończoność przed wejściem do sekcji krytycznej.

Poprawne rozwiązanie programowe dla dwóch procesów

Pierwszym, który podał prawidłowe rozwiązanie problemu sekcji krytycznej dla dwóch procesów był holenderski matematyk T.J.Dekker. Zaprezentowane na następnej planszy rozwiązanie, w postaci fragmentu kodu w języku (pseudo)C jest modyfikacją algorytmu Dekkera i nosi nazwę algorytmu Petersona od nazwiska jego autora Gary'ego Petersona. Listing zawiera (oprócz zewnętrznej, nieskończonej pętli do. . . while) jedynie kod sekcji wejściowej i wyjściowej (w tym wypadku składa się ona z tylko jednego wiersza). Kod sekcji krytycznej i reszty procesu został zastąpiony ciągami znaków *Sekcja Krytyczna* i *Reszta*.

Kod rozwiązania

Współdzielone zmienne wymagane przez algorytm

bool *flaga*[2]; //tablica flag gotowości procesów do wejścia do s.k
unsigned int *numer*; //numer procesu (0 lub 1), któremu zezwolono wejść do s.k.

Proces P_0

```
do {
    flaga[0]=true;
    numer=1;
    while(flaga[1] && numer==1)
        ;
    Sekcja Krytyczna
    flaga[0]=false;
    Reszta
} while(1);
```

Proces P_1

```
do {
    flaga[1]=true;
    numer=0;
    while(flaga[0] && numer==0)
        ;
    Sekcja Krytyczna
    flaga[1]=false;
    Reszta
} while(1);
```

Dowód poprawności

Wzajemne wykluczanie

Jeden z dwóch procesów wchodzi do sekcji krytycznej wtedy i tylko wtedy, kiedy flaga procesu przeciwnego ma wartość *false* lub gdy zmienna *numer* zawiera jego identyfikator. Może zaistnieć sytuacja, w której oba procesy będą miały ustawione flagi, ale zmienna *numer* może przyjąć tylko jedną wartość, a więc jeden z nich będzie wykonywał pętlę **while**, a drugi wejdzie do sekcji krytycznej.

Dowód poprawności

Wzajemne wykluczanie

Jeden z dwóch procesów wchodzi do sekcji krytycznej wtedy i tylko wtedy, kiedy flaga procesu przeciwnego ma wartość *false* lub gdy zmienna *numer* zawiera jego identyfikator. Może zaistnieć sytuacja, w której oba procesy będą miały ustawione flagi, ale zmienna *numer* może przyjąć tylko jedną wartość, a więc jeden z nich będzie wykonywał pętlę **while**, a drugi wejdzie do sekcji krytycznej.

Postęp

Załóżmy, że proces o numerze 0 chce wejść do sekcji krytycznej, a proces o numerze 1 nie jest nią zainteresowany (bo wykonuje swoją *resztę*). Zmienna *numer* będzie miała wartość 1, ale flaga procesu o numerze 1 nie będzie ustawiona, a więc proces 0 nie zostanie powstrzymany przed wejściem do sekcji krytycznej.

Dowód poprawności

Wzajemne wykluczanie

Jeden z dwóch procesów wchodzi do sekcji krytycznej wtedy i tylko wtedy, kiedy flaga procesu przeciwnego ma wartość *false* lub gdy zmienna *numer* zawiera jego identyfikator. Może zaistnieć sytuacja, w której oba procesy będą miały ustawione flagi, ale zmienna *numer* może przyjąć tylko jedną wartość, a więc jeden z nich będzie wykonywał pętlę **while**, a drugi wejdzie do sekcji krytycznej.

Postęp

Załóżmy, że proces o numerze 0 chce wejść do sekcji krytycznej, a proces o numerze 1 nie jest nią zainteresowany (bo wykonuje swoją *resztę*). Zmienna *numer* będzie miała wartość 1, ale flaga procesu o numerze 1 nie będzie ustawiona, a więc proces 0 nie zostanie powstrzymany przed wejściem do sekcji krytycznej.

Ograniczone oczekiwanie

Instrukcja przypisania z sekcji wyjściowej gwarantuje, że proces będzie czekał na wejście do sekcji krytycznej tylko do momentu, gdy drugi z procesów zakończy swoją sekcję krytyczną i wykona sekcję wyjściową.

Poprawne rozwiązanie programowe dla wielu procesów

Uogólnienie przedstawionego wcześniej rozwiązania na *n* procesów, nie jest proste, a otrzymany kod jest mniej czytelny niż w przypadku dwóch procesów. Zmianie ulega typ zmiennej współdzielonej, która jest tablicą flag. Każda flaga może przyjmować teraz trzy wartości: *puste* - proces jest poza sekcją krytyczną, *gotowy* - proces zgłasza swą gotowość do wejścia do sekcji krytycznej, *w_sekcji* proces jest w sekcji krytycznej. Zmienna *numer* również przyjmuje większy zakres wartości (od 0 do n-1) niż poprzednio. Uwaga: Zmienna *j* jest zmienną lokalną procesu, tzn. nie jest współdzielona z innymi procesami (poza właścicielem tej zmiennej, żaden inny proces nie ma do niej dostępu).

Kod rozwiązania dla i–tego procesu

Współdzielone zmienne wymagane przez algorytm

```
enum stan {puste, gotowy, w_sekcji};
enum stan flaga[n];
unsigned int numer;          //przyjmuje wartości od 0 do n-1
```

Proces P_i

```
unsigned int j;      // przyjmuje wartości od 0 do n
do {
   do {
      flaga[i]=gotowy;
      j=numer;
      while(i!=j)
         if(flaga[j]!=puste) j=numer;
         else j=(j+1) % n;
      flaga[i]=w_sekcji;
      j= 0;
      while(j<n&&(j==i||flaga[j]!=w_sekcji)) j++;
   } while(j<n&&(numer!=i||flaga[numer]!=puste));
   numer=i;
   Sekcja Krytyczna
   j=(numer+1) % n;
   while(flaga[j]==puste) j=(j+1) % n;
   numer=j;
   flaga[i]=puste;
   Reszta
} while(1);
```

Dowód poprawności

Wzajemne wykluczanie

Proces z grupy procesów ubiegających się o dostęp do współdzielonego zasobu wchodzi do sekcji krytycznej wtedy i tylko wtedy, gdy jego flaga gotowości ma wartość *w_sekcji*, a flagi pozostałych procesów mają inną wartość. Ponieważ tylko on może ustawić swoją flagę na wspomnianą wartość oraz dokonuje sprawdzenia flag pozostałych procesów po jej ustawieniu, to warunek wzajemnego wykluczania jest zachowany.

Dowód poprawności

Wzajemne wykluczanie

Proces z grupy procesów ubiegających się o dostęp do współdzielonego zasobu wchodzi do sekcji krytycznej wtedy i tylko wtedy, gdy jego flaga gotowości ma wartość *w_sekcji*, a flagi pozostałych procesów mają inną wartość. Ponieważ tylko on może ustawić swoją flagę na wspomnianą wartość oraz dokonuje sprawdzenia flag pozostałych procesów po jej ustawieniu, to warunek wzajemnego wykluczania jest zachowany.

Postęp

Wartość zmiennej *numer* ulega zmianie tylko wtedy gdy proces wchodzi lub wychodzi z sekcji krytycznej. Jeśli tylko jeden proces jest zainteresowany wejściem do sekcji krytycznej, a żaden inny nie wykonuje jej, ani nie ubiega się o wejście do niej, to może on wykonać sekcję krytyczną poza kolejnością wyznaczaną przez zmienną *numer*.

Dowód poprawności

Wzajemne wykluczanie

Proces z grupy procesów ubiegających się o dostęp do współdzielonego zasobu wchodzi do sekcji krytycznej wtedy i tylko wtedy, gdy jego flaga gotowości ma wartość *w_sekcji*, a flagi pozostałych procesów mają inną wartość. Ponieważ tylko on może ustawić swoją flagę na wspomnianą wartość oraz dokonuje sprawdzenia flag pozostałych procesów po jej ustawieniu, to warunek wzajemnego wykluczania jest zachowany.

Postęp

Wartość zmiennej *numer* ulega zmianie tylko wtedy gdy proces wchodzi lub wychodzi z sekcji krytycznej. Jeśli tylko jeden proces jest zainteresowany wejściem do sekcji krytycznej, a żaden inny nie wykonuje jej, ani nie ubiega się o wejście do niej, to może on wykonać sekcję krytyczną poza kolejnością wyznaczaną przez zmienną *numer*.

Ograniczone oczekiwanie

Każdy proces opuszczający sekcję krytyczną w sekcji wyjściowej wyznacza swojego następcę do wejścia do sekcji krytycznej. W ten sposób każdy proces, który ubiega się o wykonanie sekcji krytycznej dostanie pozwolenie po co najwyżej *n-1* próbach.

Algorytm piekarni

Innym rozwiązaniem problemu sekcji krytycznej dla n procesów jest algorytm piekarni, który został opracowany przez Lesliego Lamporta. Nazwa tego algorytmu wzięła się od sposobu w jaki piekarnie w Stanach Zjednoczonych sprzedają chleb. W polskich realiach odpowiada to sposobowi przyjmowania pacjentów przez lekarzy w przychodniach. Każdy pacjent musi się zarejestrować i otrzymać swój numer. Im niższa jest wartość tego numeru, tym szybciej jest jego właściciel obsługiwany. Podobne rozwiązanie można zastosować dla procesów ubiegających się o wejście do sekcji krytycznej. Jednakże w tym przypadku może zdarzyć się, że dwa procesy otrzymają ten sam numer. Taki konflikt rozstrzyga się porównując ich identyfikatory (PID), które też są numerami. Zanim zostanie przedstawiony kod rozwiązania musimy zdefiniować operację porównywania par liczb, oraz operację wybierania liczby maksymalnej ze zbioru liczb:
$(a, b) < (c, d) \iff a < c \cup (a = c \cap b < d)$
$max(a_0, a_1, \ldots, a_{n-1})$ jest taką liczbą k, że $k \geqslant a_i$ dla $i = 0, \ldots, n-1$

Pseudokod rozwiązania dla procesu P_i

Współdzielone zmienne wymagane przez algorytm

```
bool wybrane[n];
int numer[n];
```

Proces P_i

```
do {
    wybrane[i]=true;
    numer[i]=max(numer[0],numer[1],...,numer[n-1])+1;
    wybrane[i]=false;
    for(j=0;j<n;j++)
    {
      while(wybrane[j])
            ;
      while(numer[j]!=0 && (numer[j],j)<(numer[i],i))
            ;
    }
    Sekcja Krytyczna
    numer[i]=0;
    Reszta
} while(1);
```

Dowód poprawności

Wzajemne wykluczanie

Zauważmy, że każdy proces, który wchodzi do sekcji krytycznej otrzymuje numer, który jest następnikiem największego z dotychczas wybranych numerów. Ponieważ operacja wybierania nie jest niepodzielna, to dodatkowo sprawdzane są unikatowe numery identyfikacyjne procesów, gdyby pojawiły się dwa lub większa liczba procesów o takich samych wybranych numerach. Operacja porównania numerów wstrzymywana jest do czasu zakończenia wybierania numeru przez nadchodzące procesy. To wszystko gwarantuje spełnienie warunku wzajemnego wykluczania.

Dowód poprawności

Wzajemne wykluczanie

Zauważmy, że każdy proces, który wchodzi do sekcji krytycznej otrzymuje numer, który jest następnikiem największego z dotychczas wybranych numerów. Ponieważ operacja wybierania nie jest niepodzielna, to dodatkowo sprawdzane są unikatowe numery identyfikacyjne procesów, gdyby pojawiły się dwa lub większa liczba procesów o takich samych wybranych numerach. Operacja porównania numerów wstrzymywana jest do czasu zakończenia wybierania numeru przez nadchodzące procesy. To wszystko gwarantuje spełnienie warunku wzajemnego wykluczania.

Postęp

Ponieważ tylko procesy gotowe do wejścia do sekcji krytycznej wybierają numery, to zapewniony jest warunek postępu.

Dowód poprawności

Wzajemne wykluczanie

Zauważmy, że każdy proces, który wchodzi do sekcji krytycznej otrzymuje numer, który jest następnikiem największego z dotychczas wybranych numerów. Ponieważ operacja wybierania nie jest niepodzielna, to dodatkowo sprawdzane są unikatowe numery identyfikacyjne procesów, gdyby pojawiły się dwa lub większa liczba procesów o takich samych wybranych numerach. Operacja porównania numerów wstrzymywana jest do czasu zakończenia wybierania numeru przez nadchodzące procesy. To wszystko gwarantuje spełnienie warunku wzajemnego wykluczania.

Postęp

Ponieważ tylko procesy gotowe do wejścia do sekcji krytycznej wybierają numery, to zapewniony jest warunek postępu.

Ograniczone oczekiwanie

Procesy wchodzą do sekcji krytycznej w takim porządku w jakim nadeszły, a więc spełnienie warunku ograniczonego czekania jest zapewnione.

Podsumowanie

Wszystkie przedstawione tu rozwiązania programowe są z teoretycznego punktu widzenia poprawne. Niestety, w praktyce te rozwiązania mogą zawieść, jeśli program będzie wykonywany na procesorze stosującym wykonywanie instrukcji poza kolejnością (ang. *out of order execution*). Poprawność tych rozwiązań może również być naruszona podczas wykonywania przez kompilator optymalizacji kodu wynikowego. Stosując te rozwiązania należy pamiętać o dodatkowych środkach, które pozwalają uniknąć opisanych problemów. Innymi wadami przedstawionych algorytmów są problemy z zastosowaniem ich do bardziej skomplikowanych zadań oraz to, że wymagają one aktywnego oczekiwania. Ta ostatnia wada wiąże się z problemem nazywanym *inwersją priorytetów*. Występuje on w systemach stosujących szeregowanie priorytetowe. Załóżmy, że w takim systemie pozwolenie na wejście do sekcji krytycznej uzyskuje proces niskopriorytetowy. Podczas jej wykonywania zostaje on wywłaszczony przez proces wysokopriorytetowy, który zaczyna ubiegać się od dostęp do tej samej sekcji. Ponieważ jest ona zajęta, to ten proces przechodzi w stan aktywnego oczekiwania, a ze względu na jego priorytet procesor nie zostanie przydzielony procesowi niskopriorytetowemu, który mógłby zakończyć to oczekiwanie wychodząc z sekcji.

Wprowadzenie

Najprostszym sposobem zapewnienia wyłączności i niepodzielności wykonania sekcji krytycznej jest wyłącznie na czas, kiedy przebywa w niej proces, systemu przerwań. Niestety, to rozwiązanie może prowadzić do większych problemów, niż sytuacje hazardowe (Co jeśli któraś z instrukcji z sekcji krytycznej wygeneruje wyjątek?). Nie daje się ono również zastosować w systemach wieloprocesorowych. Zamiast tak radykalnego rozwiązania twórcy sprzętu, systemów operacyjnych i translatorów oferują szereg środków, które wspierają programistów w rozwiązywaniu problemu sekcji krytycznej. Dalej zostaną omówione najpopularniejsze z nich.

Niepodzielne rozkazy sprzętowe

Większość współczesnych architektur sprzętowych oferuje rozkazy, które pozwalają wykonać w sposób niepodzielny, na prostych zmiennych takie operacje, jak dodawanie, odejmowanie, operacje binarne. Istnieją również rozkazy pomagające rozwiązać problem sekcji krytycznej dla operacji na strukturach danych. Są nimi rozkazy typu „testuj i ustaw" oraz „wymień". Pierwszy z nich pozwala w sposób niepodzielny odczytać wartość zmiennej, a następnie ją zmodyfikować. Drugi z nich dokonuje zamiany wartości dwóch zmiennych, które pełnią rolę zamka i klucza. Oba rozkazy pozwalają zapewnić spełnienie warunku wzajemnego wykluczania i mogą posłużyć do implementowania opisanych wcześniej algorytmów rozwiązywania problemu sekcji krytycznej. Ponieważ blokują one magistralę pamięci, a w związku z tym także dostęp do określonej lokacji RAM innym rozkazom, to są one szczególnie przydatne w systemach wieloprocesorowych.

Semafor

Semafor jest zmienną całkowitą, do której dostęp można uzyskać (poza inicjalizacją) jedynie za pomocą dwóch specjalnych operacji: *czekaj* i *sygnalizuj*. Pierwsza sprawdza, czy semafor ma wartość większą od zera. Jeśli tak, to zmniejsza ją o jeden, jeśli nie, to czeka aż tę wartość osiągnie i dopiero wtedy ją zmniejsza. Druga polega na zwiększeniu wartości semafora o jeden. Obie operacje są w całości wykonywane niepodzielnie. Takie rozwiązanie pierwszy zaproponował Edsger Dijkstra, dlatego te operacje są oznaczane odpowiednio symbolami P (od proben) i V (od verhogen). Stosuje się również angielskie określenia *up* i *down*. Istnieją różne wersje semaforów, niektóre z nich mogą przyjmować różne wartości, inne tylko dwie. Te ostatnie nazywane są semaforami binarnymi lub *muteksami*. Istnieją również dwa sposoby implementacji operacji czekaj. Pierwszy z nich wymaga aktywnego oczekiwania na podniesienie semafora i został już opisany wyżej. Semafory posiadające tak zaimplementowaną operację oczekiwania nazywa się „wirującymi blokadami” (ang. *spin-lock*) i są one stosowane w systemach wieloprocesorowych, wtedy, kiedy istnieje pewność, że proces będzie krótko czekał na podniesienie semafora. Drugi sposób polega na umieszczeniu wszystkich procesów czekających na podniesienie semafora w kolejce oczekiwania na to zdarzenie. Kiedy semafor zostanie podniesiony uaktywniany jest jeden z tych procesów i on może kontynuować swoje działanie. O procesie, który oczekuje na podniesienie semafora w kolejce mówimy, że został uśpiony, a proces, który został wybrany z tej kolejki oczekiwania mówimy, że został obudzony.

Realizacja P i V

Aktywne oczekiwanie

czekaj(S): **while**(S⩽0);
S−−;

sygnalizuj(S): S++;

Usypianie

czekaj(S): S−−;
if(S<0)
Uśpij(proces);

sygnalizuj(S): S++;
if(S⩽0)
Obudź(proces);

Mutex

czekaj(S): **if**(S==1)
S=0;
else
Uśpij(proces);

sygnalizuj(S): **if**(!*Czeka*(proces))
S=1;
else
Obudź(proces);

Regiony

Choć semafory są skuteczne w rozwiązywaniu sekcji krytycznej i proste w zastosowaniu, to mogą być dosyć niewygodne w użyciu, tym samym prowadząc do powstania błędów logicznych w programach. Programista może np.: zapomnieć o umieszczeniu w programie instrukcji podnoszącej semafor. W takim wypadku może dojść do zablokowania działania wszystkich procesów czekających na podniesienie semafora. Aby zapobiec takim sytuacjom w językach programowania (głównie proceduralnych) wprowadzono instrukcje pozwalające tworzyć tzw. regiony krytyczne. Słowo kluczowe **shared** pozwala na określenie zmiennej, jako współdzielonej przez kilka procesów, natomiast instrukcja **region** *zmienna_współdzielona* **do** *operacja*; gwarantuje, że operacja wykonana na zmiennej współdzielonej będzie niepodzielna. Kompilator do realizacji tej niepodzielności może użyć niejawnie semaforów. Do rozwiązywania zaawansowanych problemów synchronizacji przydaje się konstrukcja regionu warunkowego: **region** *zmienna_współdzielona* **when** *warunek* **do** *operacja*;. Operacja zostanie wykonana, tylko wtedy, kiedy warunek będzie spełniony. Praktyczne problemy synchronizacji wymagają dosyć często, aby warunek był sprawdzany nie przed wejściem do sekcji krytycznej, ale w trakcie jej trwania, np.: przed wykonaniem jednej z kilku operacji. Do realizacji oczekiwania na spełnienie warunku służy instrukcja **await**(*warunek*). Regiony krytyczne można zagnieżdżać, ale należy robić to ostrożnie, gdyż może to prowadzić do problemów zwanych *zakleszczeniami*.

Słowo kluczowe volatile

W językach C, C++ i Java, w deklaracjach zmiennych można użyć słowa kluczowego *volatile*, które jest informacją dla kompilatora, że nie powinien stosować optymalizacji w dostępie do tej zmiennej, polegającej na odczycie kopii jej wartości z rejestru procesora. Tym samym wszelkie operacje dokonywane na tej zmiennej, są dokonywane bezpośrednio na jej oryginalnej, aktualnej wartości umieszczonej w pamięci operacyjnej. Mimo, że w pewnych warunkach i językach programowania słowo kluczowe *volatile* może dać efekt niepodzielnego dostępu do zmiennej, to nie należy go traktować jako środka synchronizacji. Nieporozumienia co do sposobu działania tego słowa często prowadzą do powstania programów, które są nieodporne na sytuacje hazardowe.

Monitory

Przykład w języku Java

```
class Monitor {
   private int field;
   public void synchronized setField(int f) {
     field = f;
   }
   public int synchronized getField() {
     return field;
   }
}
```

Monitory są środkiem synchronizacji właściwym dla języków obiektowych, choć zostały opracowane niezależnie od techniki obiektowej. Konstrukcja ta jest autorstwa Per Brinch Hansena i C.A.R Hoare'a. Monitor można określić jako obiekt, którego wszystkie pola są prywatne, a ich wartości osiągalne tylko za pomocą metod obiektu, wykonywanych w sposób niepodzielny. Z obiektem monitora może być, podobnie jak z regionem, związany warunek. Monitory znalazły zastosowanie w języku Java, niestety ich realizacja nie jest do końca zgodna z intencjami twórców tego środka synchronizacji. Warunek w Javie jest związany z monitorem niejawnie i osiągalny za pomocą metod *wait*(), *notify*(), *notifyAll*().

Klasyczne problemy synchronizacji

Opisane w dalszej części wykładu problemy stanowią ważne zagadnienia dotyczące współbieżności, bowiem większość rzeczywistych problemów synchronizacji daje się sprowadzić do któregoś z nich. Dzięki temu można zastosować do nich znane i sprawdzone rozwiązania. Problemy te stanowią również dobry zestaw testowy (ang. *test suit*) dla każdego nowego rozwiązania problemu synchronizacji.

Problem ograniczonego buforowania

Dane są dwa procesy. Jeden z nich jest *producentem*, który produkuje kolejne jednostki informacji, drugi z nich jest *konsumentem*, który zużywa te informacje. Żeby żadna z informacji wyprodukowanych przez producenta nie została stracona, są one buforowane. Jeśli konsument działa z taką samą lub większą szybkością niż producent, to zawsze będą wolne bufory i co najwyżej konsument będzie czekał na nowe jednostki informacji. Mamy wtedy do czynienia z problemem *nieograniczonego buforowania*. Jeśli jednak ten warunek nie jest spełniony, to może zabraknąć pustych buforów i producent będzie musiał zaczekać aż konsument opróżni któryś z zajętych buforów. Ten problem jest nazywany problemem *ograniczonego buforowania*. Na następnych planszach znajdują się dwa rozwiązania tego problemu. Pierwsze z nich pozwala zapełnić co najwyżej n-1 buforów, drugie wymaga z kolei niepodzielności operacji modyfikacji wartości zmiennej *licznik*.

Pierwsze rozwiązanie

Współdzielone elementy

```
const int n = ...;
typedef ... jednostka;
jednostka bufor[n];
unsigned int we,wy; //przyjmują wartości od 0 do n-1
```

Producent

```
do {
   ...
   Produkuj jednostkę w nastp
   ...
   while((we + 1) % n == wy)
      ;
   bufor[we]=nastp;
   we=(we + 1) % n;
} while(1);
```

Konsument

```
do {
   while(we == wy)
      ;
   nastk= bufor[wy];
   wy=(wy + 1) % n;
   ...
   Konsumuj jednostkę z nastk
   ...
} while(1);
```

Drugie rozwiązanie

Współdzielone elementy

Te same co poprzednio oraz dodana jest zmienna całkowita *licznik*, a usunięte zostały zmienne *we* i *wy* (są teraz lokalne).

Producent

```
do {
   ...
   Produkuj jednostkę w nastp
   ...
   while(licznik == n)
      ;
   bufor[we]=nastp;
   we=(we + 1) % n;
   licznik++;
} while(1);
```

Konsument

```
do {
   while(licznik == 0)
      ;
   nastk= bufor[wy];
   wy=(wy + 1) % n;
   licznik--;
   ...
   Konsumuj jednostkę z nastk
   ...
}while(1);
```

Problem czytelników i pisarzy

W problemie czytelników i pisarzy istnieją dwie grupy procesów, które mają dostęp do współdzielonego zasobu. Do pierwszej należą procesy, które wyłącznie odczytują stan współdzielonego zasobu i dlatego nazywamy je *czytelnikami*, do drugiej należą procesy, które mogą stan zasobu odczytywać lub zapisywać. Nazywamy je *pisarzami*. Jednoczesny dostęp kilku czytelników do zasobu jest możliwy, gdyż nie powoduje żadnych komplikacji, natomiast dostęp pisarzy musi odbywać się na zasadzie wyłączności i niepodzielności. Istnieje kilka wersji problemu czytelników i pisarzy, tu zostaną opisane tylko dwie. W *pierwszym* problemie czytelników i pisarzy, żaden czytelnik nie powinien czekać na dostęp do zasobu. Innymi słowy praca pisarzy może być wstrzymywana przez czytelników. W *drugim* problemie czytelników i pisarzy, jeśli któryś z pisarzy jest gotów do modyfikacji zasobu, to żaden czytelnik nie powinien uzyskać do niego dostępu. Innymi słowy, praca czytelników jest wstrzymywana przez pisarzy. Rozwiązania obu przypadków można uzyskać stosując dwa semafory, jeden zapewniający niepodzielność zliczania procesów pisarzy/czytelników czekających na dostęp do zasobu, drugi zapewniający wyłączność dostępu do zasobu.

Problem pięciu ucztujących filozofów

W problemie ucztujących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztuciec (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztuciec. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem *zakleszczenia*. Innym razem może dojść do *zagłodzenia* jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

- Pozwolić jednocześnie zasiadać do stołu co najwyżej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

Problem pięciu ucztujących filozofów

W problemie ucztujących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztuciec (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztuciec. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem *zakleszczenia*. Innym razem może dojść do *zagłodzenia* jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

- Pozwolić jednocześnie zasiadać do stołu co najwyżej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

Problem pięciu ucztujących filozofów

W problemie ucztujących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztuciec (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztuciec. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem *zakleszczenia*. Innym razem może dojść do *zagłodzenia* jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

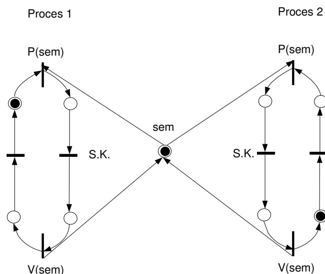
- Pozwolić jednocześnie zasiadać do stołu co najwyżej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

Problem pięciu ucztujących filozofów

W problemie ucztujących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztuciec (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztuciec. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem *zakleszczenia*. Innym razem może dojść do *zagłodzenia* jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

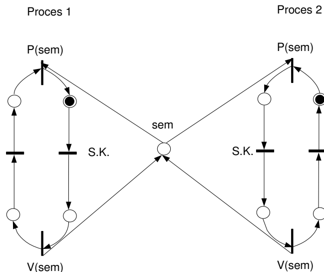
- Pozwolić jednocześnie zasiadać do stołu co najwyżej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

Sieci Petriego



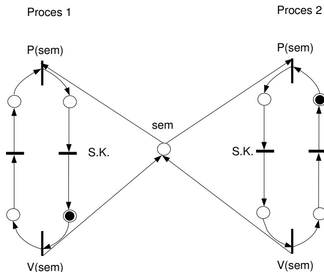
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Sieci Petriego



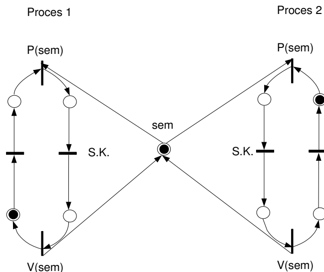
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Sieci Petriego



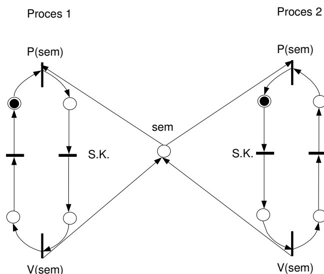
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Sieci Petriego



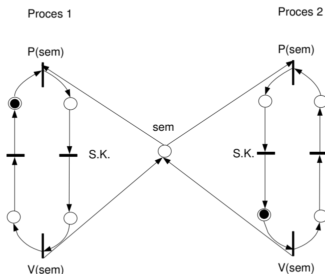
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Sieci Petriego



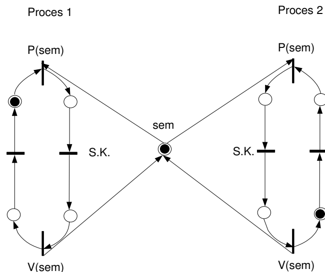
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Sieci Petriego



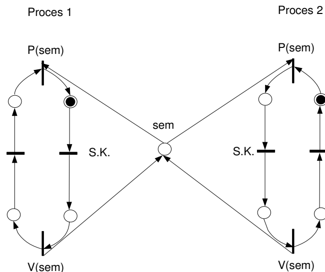
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Sieci Petriego



Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Sieci Petriego



Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejścia modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia *łącza*. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia *łącza*. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia *łącza*. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia *łącza*. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia *łącza*. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia *łącza*. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Komunikacja bezpośrednia lub pośrednia

W komunikacji bezpośredniej łącze jest tworzone tylko między parą procesów, przy czym może ono być jednokierunkowe lub dwukierunkowe. Procesy określają odbiorcę za pomocą jego identyfikatora, którym może być nazwa lub PID, podczas tworzenia łącza. W przypadku komunikacji pośredniej procesy przesyłają sobie komunikaty za pośrednictwem *portu*, który jest nazywany również *skrzynką pocztową*. Skrzynka ta może być tworzona za pomocą wywołania systemowego przez procesy. Właścicielem skrzynki zostaje proces, który wywołał to wywołanie, ale może on również przekazać prawa własności innemu procesowi. Port jest niszczony również za pomocą odpowiedniego wywołania systemowego. Prawa własności skrzynki mogą być również niezbywalne, wówczas taka skrzynka niszczona jest wraz z zakończeniem procesu-właściciela. W komunikacji pośredniej należy zapewnić system identyfikacji odbiorcy i nadawcy komunikatów znajdujących się w skrzynce.

Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

1. **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
2. **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
3. **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

1. nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
2. praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

1. **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
2. **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
3. **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

1. nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
2. praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

1. **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
2. **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
3. **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

1. nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
2. praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

1. **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
2. **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
3. **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

1. nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
2. praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

1. **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
2. **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
3. **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

1. nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
2. praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

1. **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
2. **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
3. **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

1. nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
2. praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwanie będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Zniekształcenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się redundancję (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwanie będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Zniekształcenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się redundancję (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwanie będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Zniekształcenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się redundancję (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwanie będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Zniekształcenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się redundancję (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!