

Wyznaczanie czasu globalnego	
Lamporta	<ul style="list-style-type: none"> • Zdarzenie wewnętrzne - inkrementacja zegara. • Wystanie wiadomości - inkrementacja zegara. • Odebranie wiadomości - inkrementacja $\max(\text{zegar odebrany, własny zegar})$
Matterna	<ul style="list-style-type: none"> • Działa analogicznie do Lamporta, zegary mają postać wektorową. • Przy dowolnym zdarzeniu inkrementujemy swoją pozycję. • Przy odbiorze sprawdzamy pozycje wszystkich procesów i bierzemy max wartość z wiadomości/ własną.
Kanały	
Müllendera - FIFO Send <ul style="list-style-type: none"> • Numer sekwencyjny wiadomości do danego procesu ($\text{seqno}[j]$) - inkrementowany przed wysłaniem Receive <ul style="list-style-type: none"> • Pakiet porównuje otrzymany w wiadomości numer seqno z odpowiednią pozycją własnej tablicy delivno (liczba (numer) wiadomości dostarczonych od danego procesu): <ul style="list-style-type: none"> ◦ jeśli warunek $\text{pktIn.seqNo} == \text{delivNo}_i[j] + 1$ jest spełniony: <ul style="list-style-type: none"> ■ Wiadomość jest dostarczana, ■ Wartość $\text{delivno}[j]$ jest inkrementowana, ■ zmienna delivered ustawiana na true ◦ Wpp. wiadomość wrzucana do bufora, delivered ustawiana na false • Dopóki $\text{delivered} == \text{true}$, próbujemy dostarczyć wiadomości oczekujące w buforze chandy, La	
Kearnsa, Campa, Ahuja - Flush Channels Trytodeliver <ul style="list-style-type: none"> • Jeżeli wiadomość jest typu OF lub BF (mogą wyprzedzać): <ul style="list-style-type: none"> ◦ Jeżeli wiadomość nie czeka na żadną wiadomość, lub wiadomość, na którą czeka znajduje się w odebranych, odbieramy ją i wrzucamy do zbioru odebranych. • Wpp. (jeżeli wiadomość typu FF lub TF): <ul style="list-style-type: none"> ◦ Jeżeli wszystkie wiadomości, na które czeka wiadomość zostały już dostarczone, może ona zostać odebrana i dodana do zbioru odebranych. Send OF: <ul style="list-style-type: none"> • Do waitforno (wiadomość, na którą czekam) przypisujemy backFP (ostatnią wiadomość typu BF lub TF, która znajduje się w kanale). Send FF <ul style="list-style-type: none"> • Do waitforno przypisujemy nr wiadomości -1 (czekamy na poprzednią wiadomość, nikogo nie wyprzedzamy). Send BF <ul style="list-style-type: none"> • Do waitforno przypisujemy backFP (ostatnią wiadomość typu BF lub TF, która znajduje się w kanale). • Po wysłaniu wiadomości, jej seqno przypisujemy do backFP (nie można nas wyprzedzić). Send TF <ul style="list-style-type: none"> • Do waitforno przypisujemy nr wiadomości -1 • Po wysłaniu wiadomości, jej seqno przypisujemy do backFP. Receive: <ul style="list-style-type: none"> • Try to deliver • Jeśli udało się dostarczyć, próba dostarczenia oczekujących wiadomości 	
Środowisko zachowujące uporządkowanie przyczynowe	

Alg. Brimana, Schipera, Stephenson

Założenia: a-la zegar wektorowy

Send:

- Broadcast do wszystkich poza sobą

Receive:

- Czekamy do momentu, aż:
 - nasz zegar na pozycji nadawcy będzie równy zegarowi z wiadomości na pozycji nadawcy -1 (czy mamy wiadomość poprzednią od tego nadawcy)
 - i na każdej pozostałej pozycji nasz zegar będzie większy równy zegarowi z wiadomości.
- Wtedy jeżeli wiadomość jest do nas, dostarczamy ją.
- Następnie aktualizujemy poprzez maxa swój zegar wektorowy na wszystkich pozycjach.

Alg. Schipera, Egli, Sandoza

Struktura - zbiór:

- `Pid`
- Zegar wektorowy - moment wysłania poprzedniej wiadomości do procesu o id `Pid`

Send:

- Wiadomość wysyłana z zegarem Matterna i strukturą.
- Struktura aktualizowana dopiero po wysłaniu.

`TryToDeliver` (z perspektywy odbiorcy):

- Jeżeli w strukturze znajduje się wiadomość do mnie i mój zegar jest za młody (zegar ze struktury jest nie mniejszy od mojego), to wiadomość idzie do bufora.
- Wpp. wiadomość jest dostarczana, a zmienna `deliveredi` ustawiana na `true`
- Jeżeli `deliveredi == true`, to dla wszystkich elementów ze struktury, które nie dotyczą mnie,
 - szukam tych, które dotyczą procesów, o których nie mam informacji w swojej strukturze i dodaje do niej,
 - Jeżeli mam informacje, to aktualizuję ją (max po zegarach).
- Zegar obsługiwany wg Matterna.

Receive

- `deliveredi` ustawiane na `false`
- `TryToDeliver`
- Dopóki `delivered == true`:
 - Ustawiamy `delivered` na `false`,
 - Próbuje dostarczyć wiadomości z bufora.

Stan spójny

Alg. Chandy-Lamport

Założenia:

- Niezawodne kanały FIFO
- Stan reprezentowany jest w postaci złożenia lokalnych stanów procesów w i stanów kanałów
- Pełen asynchronizm komunikacji i przetwarzania
- Brak zegara globalnego

Złożoność czasowa: d - średnica grafu zorientowanego odpowiadającego topologii przetwarzania rozproszonego,

m - liczba krawędzi tego grafu, wówczas pomijając fazę przesyłania wiadomości o stanie procesów i kanałów złożoność czasowa algorytmu Chandy-Lamporta wynosi $d+1$, a

Złożoność komunikacyjna, w sensie liczby przesyłanych znaczników, wynosi m .

Idea:

- Płukanie kanałów

- Inicjator zapamiętuje stan lokalny i wysyła wiadomość kontrolną do wszystkich incydentnych monitorów.
- Jeżeli odebrany znacznik jest pierwszym w danym procesie detekcji:
 - Monitor zapamiętuje stan procesu uznaje stan kanału $C_{j,i}$ za pusty, i propaguje znacznik przez wszystkie swoje kanały wyjściowe.
- Jeżeli stan procesu został już wcześniej zapamiętany:
 - Monitor uznaje za stan kanału $C_{j,i}$ zbiór tych wszystkich wiadomości, które dotarły tym kanałem po zapamiętaniu stanu a przed otrzymaniem znacznika.
- Po odebraniu znaczników ze wszystkich kanałów wejściowych, monitor przesyła zapamiętany stan lokalny oraz stan kanałów do monitora inicjatora.

Alg. Lai-Yang

Złożoność czasowa: 1

Złożoność komunik: n-1

Założenia: Reprezentacja stanów lokalnych obejmują historię komunikacji,

Kolor pakietu:

- White – pakiet bez znacznika
- Red – dołączenie znacznika do wiadomości

Kolor procesora (monitora):

- White – proces nie zapamiętał jeszcze stanu
- Red – stan procesu został już zapamiętany

Idea:

- Procesy inicjowane jako White.
- Monitor zmienia kolor procesu na Red po zapamiętaniu stanu.
- Każdej wiadomości wysłanej przez proces koloru White przypisany jest kolor White, a każdej wysłanej przez proces Red – kolor Red.
- Stan procesu koloru White można zapamiętać w dowolnej chwili, lecz koniecznie przed odebraniem wiadomości koloru Red.

Inicjalizacja:

- Monitor inicjatora zapisuje stan skojarzonego ze sobą procesu aplikacyjnego.
- Zmienia kolor procesu na Red.
- Wysyła pusty znacznik dummyOut (w celu inicjalizacji) o kolorze Red do wszystkich procesów.
- Następnie rozsyła również swój stan.

Wiadomość aplikacyjna:

- Wysyłając jakąkolwiek wiadomość aplikacyjną, monitor zapamiętuje ją w zmiennej $outLog_i$, oraz oznacza ją kolorem procesu.

Otrzymanie wiadomości:

- Otrzymując pakiet o kolorze Red:
 - monitor procesu aplikacyjnego o kolorze White zmienia jego kolor na Red
 - i rozsyła jego stan.
- Przed dostarczeniem wiadomości do procesu P_i monitor zapamiętuje ją w zmiennej $inLog_i$.

Alg. Matterna

- Zał: zegary wektorowe
- Wysyłamy broadcastem w wiadomości czas z przyszłości (swój +1), by na ten moment skrzyknąć zczytywanie stanu spójnego.
- Odbiorcy wysyłają potwierdzenie.
- Gdy inicjator dostanie wszystkie potwierdzenia, wysyła wiadomość rozpoczynającą zczytywanie.
- Stan kanału - Jeżeli monitor zapisywał już stan aplikacyjny, a etykieta odebranego pakietu jest wcześniejsza od momentu wyznaczenia stanu zapisanego w $vRecordClock_i$, monitor wysyła informację o otrzymaniu tego pakietu do inicjatora konstrukcji spójnego obrazu stanu globalnego.

Alg. Z kolorowaniem procesów i wiadomości

Założenia: Inicjator nie inicjuje współbieżnie wielu detekcji, dlatego wprowadzenie większej liczby kolorów (wartości elementu czasu wektorowego) nie jest konieczne.

Idea:

- Uproszczenie algorytmu Matterna
- W kontekście detekcji stanu globalnego znaczenie ma tylko zmiana stanu na pozycji $n+1$, stąd dwa stany tej pozycji są wystarczające dla rozróżnienia kolejnych faz wyznaczania obrazu globalnego:
 - White – przed zapamiętaniem stanu lokalnego procesu
 - Red – po zapamiętaniu stanu lokalnego.
- Kolory pakietów:
 - White – pakietu został wysłany przed zapamiętaniem stanu lokalnego procesu.
 - Red – pakietu został wysłany po zapamiętaniu stanu lokalnego procesu.
- Monitory procesów White wysyłają tylko pakiety White, a monitory procesów Red – tylko pakiety koloru Red.
- Każdy proces pierwotnie ma kolor White, a staje się Red po zapamiętaniu swojego stanu lokalnego a przed przekazaniem mu pierwszej wiadomości z pakietem koloru Red.
- Inicjator detekcji stanu globalnego zapamiętuje stan lokalny procesu, staje się Red i wysyła pusty pakiet dummyOut koloru Red do wszystkich monitorów.
- Algorytm praktycznie identyczny jak algorytm Lai-Yang:
 - różnicą jest prostsza reprezentacja stanu lokalnego
 - nie ma potrzeby pamiętania historii komunikacji,
 - Pociąga to za sobą konieczność wprowadzenia dodatkowego mechanizmu wyznaczania stanów kanałów komunikacyjnych:
 - Wiadomości w kanałach, to wiadomości w pakietach koloru White odebrane przez monitor koloru Red.
 - Za każdym razem, gdy monitor otrzymuje tego typu pakiet, przesyła zawartą w nim wiadomość do inicjatora.

Inicjalizacja:

- Inicjator rozpoczyna konstrukcję obrazu spójnego:
 - Zapamiętuje stan,
 - Zmienia kolor procesu na Red,
 - Wysyła pusty pakiet aplikacyjny o kolorze Red do pozostałych procesów.

Wysyłanie wiadomości aplikacyjnej:

- Każda wysyłana wiadomość zawiera znacznik określający kolor nadawcy tej wiadomości.

Odebranie wiadomości aplikacyjnej:

- Jeżeli proces skojarzony z monitorem odbiorcy posiada kolor Red, to otrzymanie pakietu aplikacyjnego w kolorze White powoduje przesłanie informacji o tym do monitora inicjatora.
- Z kolei otrzymania pakietu o kolorze Red, gdy proces skojarzony z monitorem odbiorcy posiada kolor White powoduje zmianę koloru procesu, oraz zapamiętanie i przesłanie informacji o jego stanie do monitora inicjatora.

Alg. Chandy-Lamporta (kanały FC)

Idea:

- Różnica od implementacji dla FIFO polega na zastąpieniu typu MARKER typem TMARKER będącego komunikatem typu TF.

Inicjalizacja:

- Monitor inicjatora zapamiętuje spontanicznie stan skojarzonego z nim procesu aplikacyjnego i wysyła wiadomość kontrolną (znacznik) typu TMARKER do wszystkich incydentnych monitorów.
- W odróżnieniu od wersji algorytmu dla kanałów FIFO, nie jest wysyłany stan za pomocą procedury SENDSTATE.

Odebranie znacznika:

- Monitory po odebraniu znacznika z kanału $C_{j,i}$ sprawdzają, czy stan lokalny został już wcześniej zapamiętany.
 - Jeśli jest to pierwszy znacznik, to:
 - stan lokalny jest zapamiętany przed dopuszczeniem do wysłania kolejnej wiadomości aplikacyjnej,
 - Znacznik wysyłany jest poprzez wszystkie kanały wyjściowe.
 - Stan kanału $C_{j,i}$ przyjmuje się przy tym jako zbiór pusty.
 - Jeżeli odebrano już znaczniki TMARKER z wszystkich kanałów wejściowych:
 - rozsyłany stan jest stan procesu i kanałów do wszystkich pozostałych monitorów.

Odebranie wiadomości aplikacyjnej:

- Sprawdzenie, czy stan lokalny był już zapamiętany.
- Jeżeli tak, to wszystkie wiadomości odebrane między momentem zapamiętania stanu a momentem otrzymania znacznika są włączane do zbioru określającego stan kanału wejściowego $C_{j,i}$.

Alg. Ze znacznikami FF/BF (kanały FC)

Idea:

- Rozwinięcie algorytmu Chandy-Lamporta dla kanałów FC
- Stany lokalne zapamiętane w wyniku odebrania znaczników typu BF (BMARKER)
- Znacznik typu FF (FMARKER), wysłany kanałem zaraz po zapamiętaniu stanu lokalnego - wyznaczenie stanu kanału.
- Aby wyróżnić interesujące nas wiadomości tworzące stan kanału $C_{j,i}$ w obrazie spójnym, wystarczy dołączyć do wszystkich wiadomości aplikacyjnych oraz znaczników FMARKER, etykietę określającą numer sekwencyjny ostatnio wysłanego komunikatu.

Inicjalizacja:

- Monitor inicjatora spontanicznie zapisuje stan za pomocą procedury RECORDSTATE, która :
 - Zapamiętuje stan lokalny procesu,
 - Inicjuje wszystkie elementy tablicy $pcktBuf_i$ na \emptyset
 - Zaznacza, że proces wziął już udział w przetwarzaniu,
 - wysyła dwa znaczniki: bMarkOut i fMarkOut.

Odebranie BMARKERA:

- Monitor odbiorcy, który jeszcze nie zapisał swojego stanu, wywołuje funkcję RECORDSTATE.

Odebranie wiadomości aplikacyjnej:

- Monitor odbiorcy, który już zapisał swój stan (zmienna $involved_i == True$) od monitora, od którego nie otrzymano dotąd znacznika FMARKER, dopisuje otrzymany pakiet do zbioru na odpowiedniej pozycji tablicy $pcktBuf_i$.

Odebranie FMARKERA:

- Zapisanie tego faktu w tablicy $recvMark_i$ poprzez nadanie j -temu elementowi wartości True.
- Opróżnienie odpowiedniej j -tej pozycji tablicy $chanState_i$ i dodanie do niej wszystkich pakietów z j -tego wpisu tablicy $pcktBuf_i$, których numer sekwencyjny jest nie większy niż numer sekwencyjny z pola $seqNoPred$ nadesłanego znacznika.
- Jeżeli FMARKER otrzymano już wszystkimi kanałami wejściowymi, rozsyłany jest stan procesu.

Detekcja zakończenia

Alg. Dijkstra, Feijen, van Gasteren

- Model synchroniczny (problem zakończenia sprowadza się do sprawdzenia czy wszystkie procesy są jednocześnie pasywne)y
- Ciąg cykli detekcyjnych

- Monitorom (procesom) przypisany jest kolor `White` albo `Black`.
- Monitory przesyłają wzdłuż pierścienia `TOKEN`, który również może mieć kolor `White` albo `Black`.
- Początkowo monitory mają kolor `White`.
- Monitory zmieniają kolor na `Black`, gdy odpowiadający im proces aplikacyjny wyśle wiadomości do procesu o indeksie większym.
- Monitor czeka aż obserwowany przez niego proces stanie się pasywny i wówczas
 - Jeżeli monitor odbierze token `Black`, wysyła token `Black`.
 - Wpp. wysyła token `White`.
- Po wysłaniu znacznika monitorowi przypisywany jest kolor `White`.
- Jeżeli inicjator odbierze token:
 - kolor procesu jak i znacznika jest `White` - **wykrycie zakończenia**.
 - kolor procesu, albo kolor znacznika równy jest `Black`, inicjator rozpoczyna kolejną rundę algorytmu.

Alg. Dijkstry-Scholtena

- Model przetwarzania dyfuzyjnego
- Kanały niezawodne
- Procesy nie ulegają awarii
- Inicjatorem przetwarzania dyfuzyjnego - korzeń drzewa
- Procesy odsyłają specjalne wiadomości, poczynając od liści drzewa.
- Każdy proces – węzeł drzewa przesyła wiadomość do swojego procesu angażującego, jeśli zebrał już wiadomości od wszystkich procesów potomnych.
- W momencie, w którym inicjator zbierze wszystkie wiadomości od swoich procesów potomnych, może uznać, że zostało **wykryte zakończenie**.

Twierdzenie

Jeżeli przetwarzanie dyfuzyjne uległo zakończeniu, fakt ten ulega wykryciu przez algorytm Dijkstry-Scholtena.

Bezpieczeństwo - liczniki

Postęp

Liczniki wiadomości

- Model atomowy (problem detekcji zakończenia można sprowadzić do wyznaczania stanów kanałów)
- $sc_i(\tau)$ – liczba wiadomości wysłanych do chwili τ przez P_i .
- $rc_i(\tau)$ – liczba wiadomości odebranych do chwili τ przez P_i .
- $SC(\tau)$ – sumaryczna liczba wiadomości wysłanych przez wszystkie procesy do chwili τ .
- $RC(\tau)$ – sumaryczna liczba wiadomości odebranych przez wszystkie procesy do chwili τ .
- W modelu atomowym równość $SC(\tau) = RC(\tau)$ oznacza, że każda wiadomość wysłana została odebrana. Tym samym wszystkie kanały są w chwili puste, a więc osiągnięty został stan zakończenia.
- Wiadomość kontrolna wysłana przez inicjatora przesyłana jest między monitorami, a po dotarciu do wszystkich monitorów wraca do inicjatora.
- Celem wysłania wiadomości kontrolnej jest wyznaczenie sumarycznej liczby wiadomości wysłanych i odebranych przez wszystkie procesy aplikacyjne
- Każdy z monitorów dodaje do znacznika stany liczników sc_i i rc_i odpowiadające bieżącej chwili τ_i . Ponieważ kanały wprowadzają opóźnienia, zebrane liczniki odpowiadają różnym momentom τ_i : $1 \leq i \leq n$.
- Oznaczmy przez SC^* i RC^* sumy zebranych przez znacznik liczników: $SC^* = \sum_{i=1}^n sc_i(\tau_i)$,

$$RC^* = \sum_{i=1}^n rc_i(\tau_i),$$

- pamiętając, że:

$$SC(\tau) = \sum_{i=1}^n sc_i(\tau),$$

$$RC(\tau) = \sum_{i=1}^n rc_i(\tau),$$

- W algorytmie detekcji zakończenia rozważane są dwie fazy detekcji:
 - Pierwsza rozpoczyna się o chwili τ_b^1 i kończy w chwili τ_e^1 ,
 - druga odpowiednio w chwilach τ_b^2 i τ_e^2
 - ponadto, że: $\tau_b^1 < \tau_e^1 < \tau_b^2 < \tau_e^2$.
- SC^* i RC^* - liczniki wyznaczone w pierwszej fazie.
- SC^{**} i RC^{**} - liczniki wyznaczone w drugiej fazie.
- Jeżeli $RC^* = SC^* = RC^{**} = SC^{**}$, to monitorowane przetwarzanie aplikacyjne osiągnęło stan zakończenia przed zakończeniem procesu detekcji.

Twierdzenie

Jeżeli $RC^* = SC^* = RC^{**} = SC^{**}$, to monitorowane przetwarzanie aplikacyjne osiągnęło stan zakończenia przed zakończeniem procesu detekcji.

Dowód

Alg. Jednofazowy

- Model atomowy
- Monitory połączone są w logiczny pierścień

Wiadomość `TOKEN` zawiera:

- `initId` - identyfikator inicjatora
- `detectNo` - numer sekwencyjny cyklu detekcji,
- `SRAccu` - suma liczników `SRBalancei` monitorów odwiedzonych już przez znacznik
- `invalid` - flaga niepoprawności procesu detekcji - przyjmuje ostatecznie wartość `True`, jeżeli którykolwiek proces aplikacyjny otrzymał między kolejnymi cyklami detekcji wiadomość, która narusza warunek konieczny poprawności detekcji: $maxDetectNo_i \geq tokenIn.detectNo$,

Zmienne:

- `detectNoi` - pole zawierające numer sekwencyjny cyklu detekcji.
- `SRBalancei` - lokalne liczniki o wartości `sci - rci` (początkowo 0).
- `maxDetectNoi` - zmienna określająca numer sekwencyjny cyklu detekcji skojarzony z wiadomością wysłaną najpóźniej, spośród wszystkich wiadomości odebranych przez `Qi`.
- `terminationDetectedi` - zmienna zostaje ustawiona na `True` jeżeli wykryte zostało zakończenie.

Inicjalizacja:

- Inicjator inkrementuje licznik cyklu detekcji `detectNoa` oraz wysyła znacznik do swojego następnika w pierścieniu.

Wysłanie wiadomości aplikacyjnej:

- Monitor inkrementuje zmienną `SRBalancei`
- przypisuje jej numer cyklu detekcji zakończenia.









Otrzymanie wiadomości aplikacyjnej:

- Monitor dekrementuje licznik `SRBalancei`.
- Jeżeli wartość numer cyklu detekcji przesłana w otrzymanej wiadomości aplikacyjnej jest większa niż bieżąca wartość `maxDetectNoi`, to zmiennej `maxDetectNoi` przypisana zostaje właśnie ta wartość.

Otrzymanie tokenu:

<ul style="list-style-type: none"> • Uaktualnienie numeru sekwencyjnego bieżącego cyklu detekcji. • Jeżeli token dotrze do inicjatora: <ul style="list-style-type: none"> ◦ jeżeli wartość pola $SRAccu == 0$ i flaga $invalid == False$ - wykrycie zakończenia. ◦ Wpp. rozpoczęcie nowego cyklu detekcyjnego. • Jeżeli token dotrze do innego monitora ($Q_i \neq Q_a$): <ul style="list-style-type: none"> ◦ <i>Do</i> $SRAccu$ dodawana jest aktualna wartość zmiennej $SRBalance_i$ ◦ fladze $invalid$ przypisywana jest suma logiczna bieżącej wartości tej flagi oraz wartości relacji $maxDetectNo_i \geq tokenIn.detectNo$: <ul style="list-style-type: none"> ■ zachodzi, jeżeli do monitora dotarł już pewien pakiet $pckt$ z etykietą $pckt.detectNo$ o wartości większej lub równej niż numer sekwencyjny bieżącego cyklu detekcji ($tokenOut.detectNo$) ■ Oznacza to, że pakiet ten został wysłany po wizycie bieżącego tokenu. ■ Zdarzenie odbioru pakietu $pckt$ będzie uwzględnione w końcowej wartości licznika $SRAccu$, a nie będzie uwzględnione zdarzenie nadania tego pakietu. ■ Konkluzja dotycząca zakończenia przetwarzania byłaby formułowana na podstawie niespójnego obrazu przetwarzania, a więc nie byłaby poprawna. ■ Wynik bieżącego cyklu detekcji - niepoprawny. ■ Konieczny kolejny cykl detekcji. 	
Detekcji zakończenia statycznego	<p>Złożoność czasowa: Graf w pełni połączony: 3×2, Pierścień: $3n$ Złożoność komunik: $3 \times 2n = 6n$ (bo potrzeba 2 cykli + zakończenie bieżącego)</p> <p>Inicjator Topologia każdy z każdym - pytamy "skończyłeś?" i odpowiedź Continously passive notAcki - służy do weryfikowania in transit Potrzebne dwie tury rozgłaszania</p>
Detekcji zakończenia dynamicznego	<p>Estymacja IT - vsendno, vrecvno (wektory - ile wiadomości wysłanych/odebranych przez dany proces i do/od kogo) Dwa cykle Na koniec wiemy, że $AIT[k] = IT[k+1]$</p>
Kanały	
Implementacja kanałów rzetelnych	<p>Rzetelne dostarczanie (ang. fair loss delivery): Jeżeli wiadomość M wysyłana jest nieskończoną liczbę razy przez proces P_i do procesu P_j i oba te procesy są poprawne, to wiadomość M jest dostarczona nieskończoną liczbę razy do P_j Ograniczone powielanie (ang. finite duplication): Jeżeli wiadomość M wysyłana jest skończoną liczbę razy przez proces P_i do procesu P_j, to wiadomość ta nie może być dostarczona nieskończoną liczbę razy do procesu P_j Brak samogeneracji (ang. no creation): Jeżeli wiadomość została dostarczona do procesu P_j, to została ona wcześniej wysłana do procesu P_j przez jakiś inny proces P_i</p>
Implementacja kanałów wytrwałych	<ul style="list-style-type: none"> • Zał: dostępność mechanizmu kanałów rzetelnych • Po stronie nadawcy implementacja mechanizmu kanałów wytrwałych sprowadza się do wysyłania wiadomości nieskończoną liczbę razy. • Po stronie odbiorcy wiadomości, każde odebrana wiadomość jest po prostu przekazywana procesowi aplikacyjnemu – adresatowi. <p>Kanał wytrwały:</p> <ul style="list-style-type: none"> • Własność wytrwałego dostarczania - jeżeli wiadomość M wysyłana jest przez proces P_i do procesu P_j i oba te procesy są poprawne, to wiadomość M jest dostarczona nieskończoną liczbę razy do P_j. • Własność braku samogeneracji - jeżeli wiadomość została dostarczona do procesu P_j, to została ona wcześniej wysłana do tego procesu przez jakiś inny proces P_i. Innymi słowy, kanał nie tworzy samorzutnie wiadomości.

Implementacja kanałów niezawodnych	<ul style="list-style-type: none"> • Zał: dostępność mechanizmu kanałów wytrwałych • Różnica między kanałami wytrwałymi i niezawodnymi polega tylko na sposobie dostarczania wiadomości. • Wiadomość jest dostarczana przez monitor procesowi aplikacyjnemu tylko wtedy, gdy nie zawiera się on w zbiorze <i>delivered_i</i>. • Po dostarczeniu procesowi wiadomość jest dodawana do zbioru <i>delivered_i</i>. • Innymi słowy, wiadomość jest dostarczana tylko wtedy, gdy już wcześniej nie była dostarczona. <p>Kanał niezawodny:</p> <ul style="list-style-type: none"> • Własność <i>niezawodnego dostarczania</i> - jeżeli wiadomość M wysłana jest przez proces P_i do procesu P_j i oba te procesy są poprawne, to wiadomość M jest ostatecznie dostarczona do P_j. • Własność <i>braku powielania</i> - żadna wiadomość wysłana do P_j nie może być dostarczona do procesu P_j więcej niż raz. • Własność <i>braku samogeneracji</i> - jeżeli wiadomość została dostarczona do procesu P_j, to została ona wcześniej wysłana do procesu P_j przez jakiś inny proces P_i. Innymi słowy, kanał nie tworzy samorzutnie wiadomości.
Detektory	
Heartbeat - idea	<ul style="list-style-type: none"> - Proces monitorowany co pewien zdefiniowany czas T_i^p wysyła wiadomość typu „I'm alive”. • Odebranie tego typu wiadomości po drugiej stronie łączy komunikacyjnego przez detektor FD w czasie T_i^{to} od ostatnio odebranej wiadomości pozwala uznać proces P_j za poprawny. • Wpp. detektor FD zaczyna podejrzewać proces P_j. • Aktywność leży po stronie procesu monitorowanego, który musi okresowo wysyłać wiadomości.
Mechanizm odpytywania - idea	<ul style="list-style-type: none"> • Modyfikacją heartbeat • Proces monitorujący co pewien zdefiniowany okres czasu T_i^p wysyła pytanie o status typu „Are you alive?”. • Proces monitorowany po odebraniu tego typu komunikatu odpowiada na niego wysyłając wiadomość typu „I am alive”. • Jeśli odpowiedź nie dotrze do procesu monitorującego (detektora awarii FD) przez zadany czas T_i^{to}, to detektor zacznie podejrzewać P_j o awarię. • Ten schemat monitorowania umożliwia lepszą kontrolę procesu monitorowania, gdyż aktywność leży w tym rozwiązaniu po stronie detektora awarii.
Heartbeat, detektor P	<p>Zał:</p> <ul style="list-style-type: none"> • Niezawodne kanały komunikacyjne • Synchroniczny model systemu • Czasy przetwarzania lokalnego i przesunięcia zegarów lokalnych pomijalnie małe w stosunku do maksymalnego czasu transmisji komunikatu <p>Cykliczne zdarzenie upływu czasu wysłania pulsu:</p> <ul style="list-style-type: none"> • Rozesłanie wiadomości typu HEARTBEAT do wszystkich pozostałych procesów. <p>Cykliczne zdarzenie upływu czasu maksymalnego oczekiwania na otrzymanie pulsu:</p> <ul style="list-style-type: none"> • Jeżeli nie otrzymano w ostatnim czasie wiadomości typu HEARTBEAT od danego procesu (nie znajduje się on w zbiorze <i>correct_i</i>) i nie jest on jeszcze podejrzewany, to dołączany on jest do zbioru podejrzewanych procesów. <p>Otrzymanie pulsu:</p> <ul style="list-style-type: none"> • Otrzymanie takiej wiadomości powoduje dodanie procesu do zbioru <i>correct</i>. • Długość T_i^{to} jest tak dobrana, by wszystkie wysłane wiadomości mogły dotrzeć do detektora.
Heartbeat, detektor ◇P	<p>Zał:</p> <ul style="list-style-type: none"> • Niezawodne kanały komunikacyjne • Częściowo synchroniczny model systemu

	<ul style="list-style-type: none">Czasy przetwarzania lokalnego i przesunięcia zegarów lokalnych są pomijalnie małe w stosunku do maksymalnego czasu transmisji komunikatu Cykliczne zdarzenie upływu czasu wysłania pulsu: <ul style="list-style-type: none">Rzeszanie wiadomości typu HEARTBEAT do wszystkich pozostałych procesów. Cykliczne zdarzenie upływu czasu maksymalnego oczekiwania na otrzymanie pulsu: <ul style="list-style-type: none">Co pewien czas: T_i^{to} detektor dołącza wszystkie procesy, które nie należą do zbioru $correct_i$ czyli takie, od których nie otrzymano ostatnio wiadomości typu HEARTBEAT do zbioru $suspected_i$.Jeśli natomiast proces należy zarówno do zbioru $correct_i$ jak i do zbioru $suspected_i$ to oznacza to, że detektor popełnił pomyłkę, która powinna być skorygowana.W takim przypadku proces przestaje być podejrzewany (usunięcie ze zbioru $suspected_i$) oraz zwiększany jest czas, po którym procesy zostają uznane za podejrzane, jeśli nie nadejdzie od niego żadna wiadomość typu HEARTBEAT.Zbiorowi $correct$ przypisana jest wartość zbioru pustego.Co pewien czas proces wysyła też wiadomości typu HEARTBEAT do wszystkich pozostałych procesów. Otrzymanie pulsu: <ul style="list-style-type: none">Proces będący nadawcą jest dołączany do zbioru procesów uznanych za poprawne $correct_i$.				
Rozgłaszanie					
Podstawowe rozgłaszanie niezawodne	<p>Założenia:</p> <ul style="list-style-type: none">niezawodne kanały komunikacyjneprocesy działają deterministycznie i poprawnie do ewentualnego załamaniamodel przetwarzania z ukrytymi awariami <p>Złożoność czasowa: 1</p> <p>Złożoność komunik: n</p> <ul style="list-style-type: none">Odpowiedzialność za niezawodność ciąży na nadawcy.Nie zapewnia przy tym dostarczenia wiadomości do wszystkich procesów, jeżeli nadawca ulegnie awarii. <p>Własności:</p> <ul style="list-style-type: none">Własność <i>ważności</i> - jeżeli procesy P_i oraz P_j są poprawne, to każda wiadomość rozgłaszana przez P_i jest ostatecznie dostarczona do P_j.Własność <i>braku powielania</i> - jeżeli wiadomość jest dostarczona, to jest dostarczona co najwyżej raz.Własność <i>braku samogeneracji</i> - jeżeli jakaś wiadomość jest dostarczona do procesu P_j, to została wcześniej rozgłoszona przez jakiś proces P_i. <p>Innymi słowy, własność ta gwarantuje, że kanały nie generują samorzutnie wiadomości.</p>				
Pasywne zgodne rozgłaszanie niezawodne	<ul style="list-style-type: none">Doskonały detektor awariiBest-effort broadcast	Wykrycie awarii u nadawcy. Jeżeli poprawne procesy odebrały wiadomość, rozpropagują ją.	 1  n-1 //w prezentacji jest n	 n  $O(n^2)$	Dokładność detektora nie jest konieczna - będą zbędne wiadomości, ale filtrowane. Silna kompletność jest niezbędna.
Aktywne zgodne rozgłaszanie niezawodne	<ul style="list-style-type: none">Best-effort broadcast	Jeżeli poprawny proces odebrał, to od razu wysyła do wszystkich.	 1  n	 $O(n^2)$  n^2	Brak konieczności użycia doskonałego detektora kosztem złożoności.
Jednolite rozgłaszanie	<p>Założenia</p> <ul style="list-style-type: none">Podstawowe rozgłaszanie niezawodne				

niezawodne z potwierdzeniami od wszystkich	<ul style="list-style-type: none"> Niezawodne kanały komunikacyjne Doskonały detektor awarii <p>Złożoność czasowa: best: 2, worst: $n+1$</p> <p>Złożoność komunik: best: $O(n^2)$, worst: n^2</p> <p>Wysyłanie:</p> <ul style="list-style-type: none"> Wysyłając wiadomość monitor umieszcza ją w pakiecie, który jest dodany do zbioru oczekujących, po czym pakiet zostaje rozgłoszony. <p>Dostarczenie do monitora:</p> <ul style="list-style-type: none"> Odbierane pakiety monitor dodaje do zbioru acki razem z identyfikatorem nadawcy. Jeżeli dostarczony pakiet nie zawiera się jeszcze w zbiorze pendingi to jest do niego dodawany a następnie rozgłoszony. <p>Odebranie przez proces:</p> <ul style="list-style-type: none"> Wiadomość jest dostarczana do procesu aplikacyjnego dopiero wtedy, kiedy monitor Q_i otrzyma tę wiadomość w wyniku retransmisji od wszystkich poprawnych procesów. <p>Czy jest potrzebny doskonały detektor awarii? Implozja potwierzeń \leftarrow struktura drzewiasta \leftarrow problem w przypadku awarii</p>
Jednolite rozgłaszanie niezawodne z potwierdzeniami od większości	<p>Złożoność czasowa: (best: 2, worst: $n+1$)</p> <p>Złożoność komunikacyjna: (best: $O(n^2)$, worst: n^2)</p> <p>Założenia</p> <ul style="list-style-type: none"> Podstawowe rozgłaszanie niezawodne Większość procesów jest poprawna <p>Pozbycie się detektora awarii, ale kosztem założenia, że większość procesów jest poprawna.</p> <p>Implozja potwierzeń</p>
Aktywne probabilistyczne rozgłaszanie niezawodne	<p>Wiadomość wysyłana do podzbioru sąsiadów.</p>
Konsensus	
Rozgłoszeniowy Algorytm konsensusu podstawowego	<p>Złożoność czasowa: (best: 1, worst: n)</p> <p>Złożoność komunikacyjna: (best: n^2, worst: n^3)</p> <p>Założenia:</p> <ul style="list-style-type: none"> Best effort broadcast Detektor p <p>Algorytm działa w rundach. Jeśli wiadomości od wszystkich poprawnych procesów i brak awarii - podejmowana jest decyzja. Jeśli wystąpi awaria - różne procesy mogą mieć różne wartości, więc nie mogą podjąć decyzji - rozpoczyna się nowa runda.</p>
Hierarchiczny algorytm konsensusu	<p>Złożoność czasowa: (best: 1, worst: n)</p> <p>Złożoność komunikacyjna: (best: n, worst: n^2)</p>

podstawowego	<p>Założenia:</p> <ul style="list-style-type: none"> • Podstawowe rozgłaszanie niezawodne • Doskonały detektor awarii <p>Każdy po kolei staje się liderem. Lider rundy rozgłasza swoją decyzję, która jest przejmowana przez pozostałe procesy, po czym zaczyna się kolejna runda z nowym liderem. Jeżeli lider ulegnie awarii, jego rolę przejmuje kolejny proces i tak dalej.</p> <ul style="list-style-type: none"> • Jeżeli <ul style="list-style-type: none"> ◦ monitor Q_i stwierdza, że zostaje liderem bieżącej, k-tej rundy (co dzieje się wtedy, gdy jego identyfikator jest równy identyfikatorowi zapisanemu w k-tym elemencie tablicy $leaderSet_i$) ◦ oraz jeżeli zmienna $proposal_i$ nie jest pusta, ◦ a także Q_i jeszcze nie podejmował decyzji, to Q_i: <ul style="list-style-type: none"> ◦ decyduje się na wartość zapisaną w $proposal_i$, ◦ zapamiętuje ten fakt i powiadamia o tym pozostałe monitory. • Z własności ważności podstawowego rozgłaszania niezawodnego wynika, że każdy proces poprawny otrzyma rozgłoszoną wiadomość, o ile rozgłaszający proces również jest poprawny. - Jeżeli więc monitor podejmuje decyzję i jest poprawny, to wszystkie inne poprawne monitory otrzymają wiadomość o podjętej przez niego decyzji. • Jeżeli monitor Q_i podejrzewa lidera bieżącej rundy lub już otrzymał od niego wiadomość, zwiększa numer rundy. • Jeżeli monitor Q_i otrzymuje powiadomienie o decyzji, to ignoruje ją, gdy: <ul style="list-style-type: none"> ◦ pochodzi ona od monitora skojarzonego z procesem znajdującym się niżej w hierarchii procesów niż P_i, ◦ lub numer rundy nadesłanej wiadomości jest mniejszy niż największy numer rundy, w której monitor Q_i otrzymał decyzję - może to oznaczać, że wiadomość ta została wysłana przez niepoprawny monitor tuż przed jego awarią. • Wpp. monitor Q_i zapisuje otrzymaną propozycję decyzji do zmiennej $proposal_i$ oraz zapamiętuje, że w rundzie, zapisanej w polu $roundNo$ odebranego komunikatu, otrzymał decyzję lidera.
--------------	---