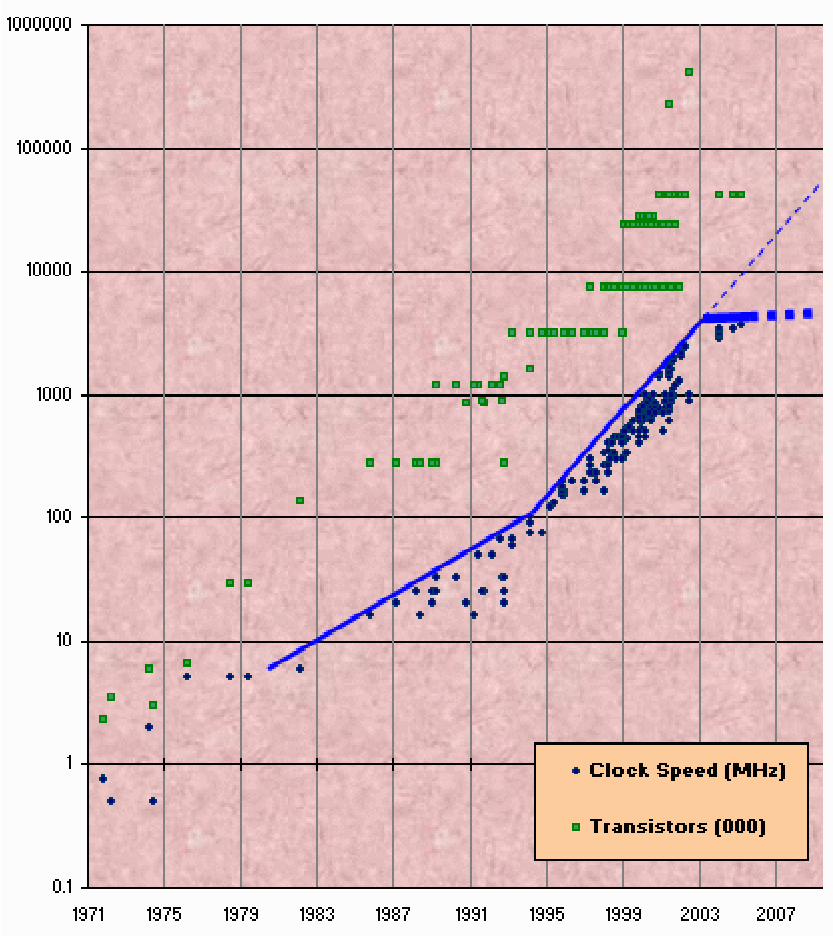
1 Podstawowe definicje i pojęcia współbieżności

1.1 Dlaczego zajmujemy się współbieżnością ?

W ciągu ostatnich 30 lat wzrost mocy przetwarzania osiągano przez:

- Zwiększenie częstotliwości zegara
- Optymalizację wykonywania operacji
- Zastosowanie pamięci podręcznych

Prawo Moor'a mówi że ekonomicznie optymalna liczba tranzystorów w układzie scalonym podwaja się co 20 miesięcy. Jest ono ekstrapolowane na moc obliczeniową komputerów. Prawo to nie sprawdza się w odniesieniu do częstotliwości zegara. Od około roku 2003 obserwuje się jego stabilizację.



Prawo Moore'a napotka bariery związane z prawami fizyki:

- Tranzystor nie może być mniejszy od atomu
- Prędkość światła jest ograniczona

Przewiduje się że w ciągu najbliższych kilku lat wzrost mocy przetwarzania będzie osiągana przez:

- Hyperthreading
- Procesory wielordzeniowe
- Zastosowanie pamięci podręcznych

W dalszej perspektywie głównym motorem wzrostu mocy przetwarzania komputerów będzie zwiększanie stopnia równoległości przetwarzania. Nie da się jednak tego osiągnąć bez radykalnej zmiany w oprogramowaniu.

Wnioski:

- *Aby wykorzystać możliwości sprzętu w dziedzinie przetwarzania równoległego należy dostosować do tego oprogramowanie.*
- *Aplikacje będą musiały być projektowane jako w coraz większym stopniu współbieżne aby wykorzystać ciągle rosnącą moc sprzętu*

Współczesne systemy i aplikacje charakteryzują się coraz większą złożonością. Typowe aplikacje:

1. Wiele procesów wykonywanych w środowisku systemu pracującego z podziałem czasu
2. Wiele procesów wykonywanych na komputerze wieloprocesowym
3. Wiele procesów wykonywanych na wielu powiązanych w jeden system maszynach (*ang. Cluster*).

W systemach takich stosuje się podział zadania na procesy gdyż zapewnia to określone korzyści:

1. Polepszenie wykorzystania systemu,
2. Zmniejszenie czasu przetwarzania
3. Ułatwienia w projektowaniu oprogramowania.

1.2 Czym jest proces ?

Proces jest czymś innym niż program. Program jest zapisem algorytmu wraz ze strukturami danych na których algorytm ten operuje. Algorytm zapisany bywa zwykle w jednym z wielu języków programowania.

Za Wirthem możemy podać definicję programu:

Program = algorytm + struktury danych

Program jest więc strukturą statyczną zapisaną na jakimś nośniku. Natomiast proces jest wykonującym się programem.

Proces - wykonujący się program

Proces jest więc aktywną strukturą dynamiczną istniejącą tylko w środowisku działającego komputera.

1.3 Podstawowe definicje współbieżności

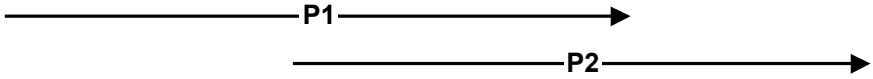
Procesy sekwencyjne (*ang. Sequential processes*)

Procesy są sekwencyjne jeżeli następny proces ze zbioru procesów rozpoczyna się po zakończeniu procesu poprzedniego.

Rys. 1-1 Procesy P1 i P2 wykonywane są sekwencyjnie

Procesy współbieżne (*ang. Concurrent processes*)

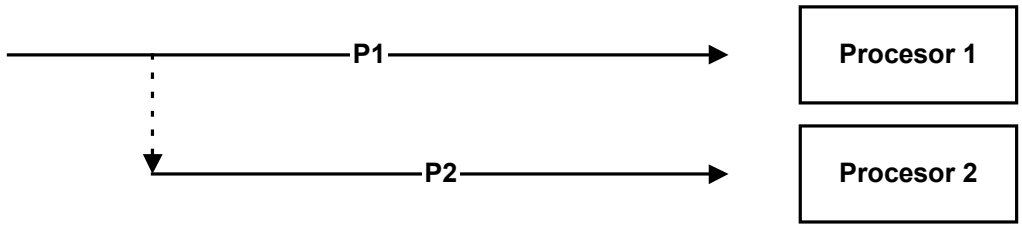
Dwa procesy są współbieżne jeżeli jeden z nich rozpoczyna się przed zakończeniem drugiego.



Rys. 1-2 Procesy P1 i P2 wykonywane są współbieżnie

Procesy równoległe (*ang.Paralell processes*)

Dwa procesy są równoległe jeżeli jeden z nich rozpoczyna się przed zakończeniem drugiego i wykonywane są jednocześnie na oddzielnych procesorach.



Rys. 1-3 Procesy P1 i P2 wykonywane są równolegle.

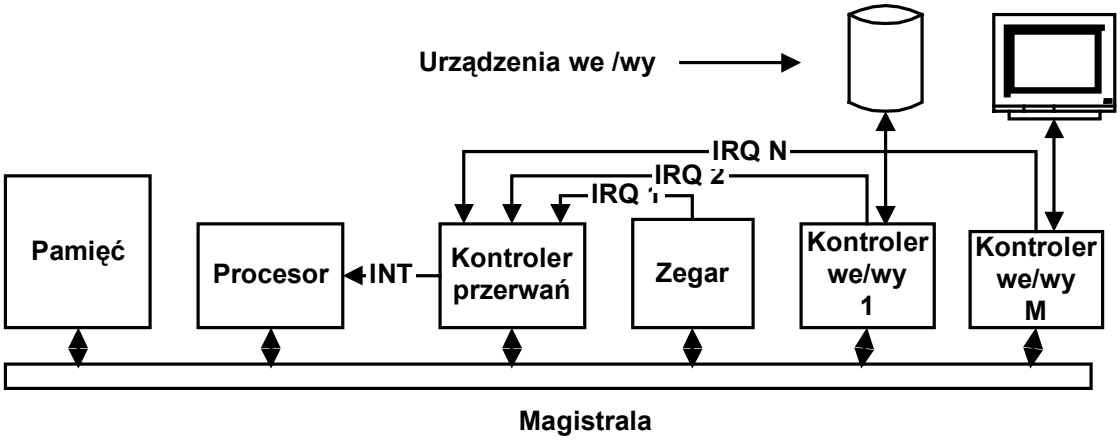
Rodzaje współbieżności

- Współbieżność konkurencyjna – procesy nie współpracują ze sobą. Ich oddziaływanie polega tylko na konkurowaniu o dostęp do zasobów których potrzebują.
- Współbieżność kooperacyjna – procesy współpracują ze sobą działając w ramach aplikacji jednej współbieżnej. Komunikują i synchronizują się ze sobą w celu wykonania pewnego zadania.

1.4 Sprzętowe podstawy współbieżności

Niezbędnym minimum sprzętowym potrzebnym do implementacji takiego systemu jest:

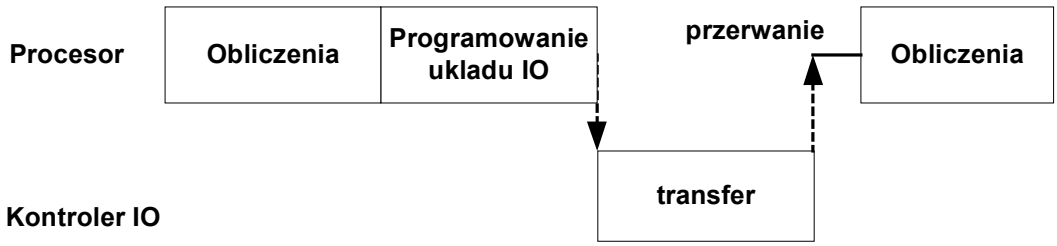
- System przerwań
- Układ zegarowy generujący impulsy które są zamieniane w przerwania.



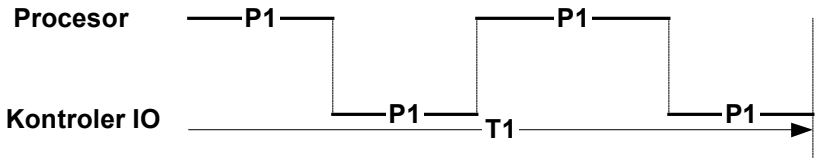
Rys. 1-4 Uproszczony schemat komputera mogącego wykonywać współbieżnie wiele procesów.

Zdarzenia w systemie komputerowym:

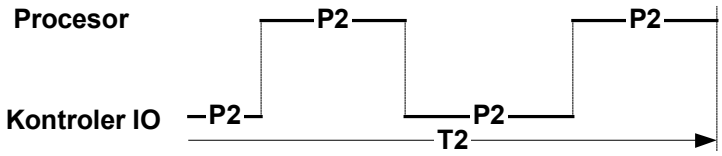
- Układ zegarowy cyklicznie generuje żądania przerwań IRQ0.
- Kontrolery urządzeń wejścia / wyjścia generują żądania IRQ1 - IRQN gdy wystąpi w nich pewne zdarzenie.
- Żądania przerwań IRQ0 – IRQN doprowadzane są do kontrolera przerwań.
- Kontroler wysyła do procesora przerwanie INT.



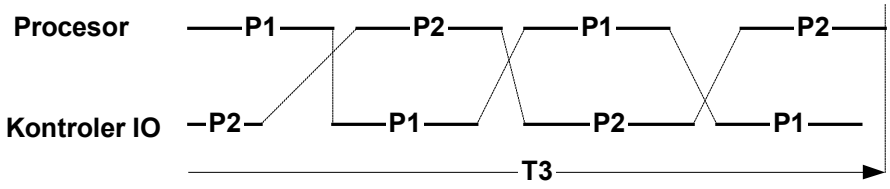
Rys. 1-5 Przebieg operacji wejścia wyjścia wykonywanej przez kontroler wejścia wyjścia



Rys. 1-6 Proces P1 wykonywany w trybie wyłącznym



Rys. 1-7 Proces P2 wykonywany w trybie wyłącznym



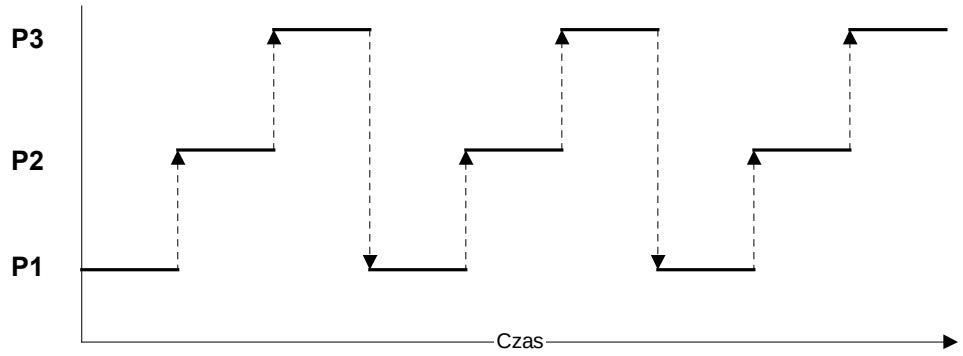
Rys. 1-8 Procesy P1 i P2 wykonywane w trybie wieloprogramowym T3 < T1 + T2

Gdy procesy P1 i P2 wykonywane są w trybie wieloprogramowym ich łączny czas wykonania T3 jest nie większy niż suma czasów wykonania w trybie wyłącznym: T3 <= T1 + T2

1.5 Przełączanie procesów, wieloprogramowość

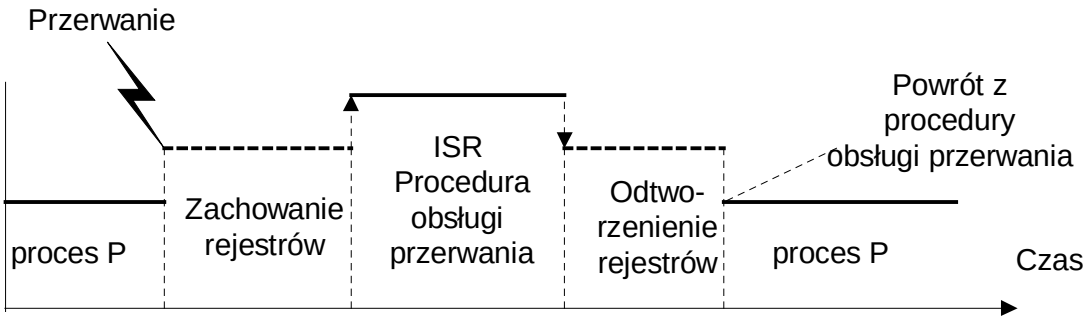
Współczesne komputery są na tyle wydajne że bez trudności mogą wykonywać wiele procesów które współdzielą czas procesora.

Procesy zgodnie z kolejnością wyznaczoną przez procedurę szeregującą (*ang. scheduler*) wykonywane są kolejno przez zadany kwant czasu (*ang. time slice*).



Rys. 1-9 Procesy P1, P2, P3 wykonujące się w trybie podziału czasu (*ang. time scharing*)

Podstawowym mechanizmem umożliwiającym taki tryb pracy są przerwania.



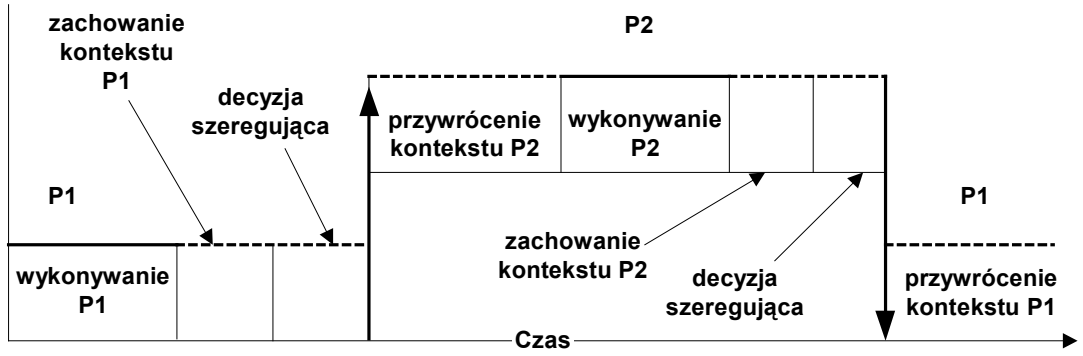
Rys. 1-10 Obsługa zdarzenia poprzez procedurę obsługi przerwania

Obsługa przerwania - chwilowe wstrzymanie aktualnie wykonywanego procesu i wykonanie procedury przypisanej zdarzeniu powodującemu przerwanie po zakończeniu której następuje powrót do przerwanego procesu.

Pojedynczy przełączenie składa się z trzech faz:

1. Zachowania kontekstu procesu dotychczas wykonywanego.
2. Podjęcie decyzji który z procesów wznowić.
3. Przywrócenie kontekstu nowego procesu.

Kontekst procesu – wszystkie informacje potrzebne do wznowienia zawieszonego wcześniej procesu.



Rys. 1-11 Zachowanie kontekstu, wykonywanie i przywrócenie kontekstu procesu

Przełączenia procesów mają miejsce w następujących sytuacjach:

- Wystąpiło **przerwanie zegarowe** i system stwierdził że wykonywany proces wyczerpał już swój kwant czasu.
- Wystąpiło **przerwanie zewnętrzne** np. od kontrolera wejścia / wyjścia pewnego urządzenia sygnalizujące zakończenie się zleconej wcześniej operacji.
- Proces bieżący wykonał pewną niedozwoloną operację polegającą na naruszeniu systemu ochrony zasobów procesora Zdarzenie takie powoduje **przerwanie wewnętrzne** procesora .
- Wykonywany proces wykonał **wywołanie systemowe** zmieniające status gotowości przynajmniej jednego procesu.

Przełączenia procesów występują w nie dających się przewidzieć momentach czasu. Stąd nie można czynić założeń że pewien ciąg instrukcji danego procesu nie zostanie przerwany.

1.6 Przeplot i operacje atomowe

W systemach z jednym procesorem procesy współbieżne muszą się wykonywać z wykorzystaniem podziału czasu procesora (przeplot).

Program jest zazwyczaj pisany w języku wysokiego poziomu.
Wykonywane są natomiast instrukcje kodu maszynowego będące wynikiem kompilacji programu źródłowego przez kompilator.

Operacja atomowa
Operacją atomową nazywamy taką operację która wykonywana jest przez procesor niepodzielnie. Znaczy to że o ile się rozpocznie, musi być w trybie wyłącznym wykonana i zakończona.

Pojedyncza instrukcja kodu maszynowego jest operacją atomową.
Trudno jest stwierdzić jakie operacje zapisane w języku wyższego poziomu będą operacjami atomowymi.

Wyodrębnienie instrukcji atomowych jest istotne gdy dwa lub więcej procesy korzystają ze wspólnego obszaru pamięci.

Przykład 1 - hazard

Zmienna wspólna	X = 0
Proces 1	Proces 2
X++	X++

Dwa procesy inkrementują wspólna zmienną

Instrukcja ta może być przetłumaczona przez kompilator co najmniej na dwa sposoby:

Sposób 1	Sposób 2

INC X	MOV A,X
	ADD A, 1
	MOV X, A

Proces	Instrukcja	Wartość X
		0
P1	INC X	1
P2	INC X	2

Przypadek 1

Proces	Instrukcja	Wartość X
		0
P1	MOV A, X	0
P1	ADD A,1	0
P2	MOV A, X	0
P2	ADD A,1	0
P1	MOV X, A	1
P2	MOV X, A	1

Przypadek 2

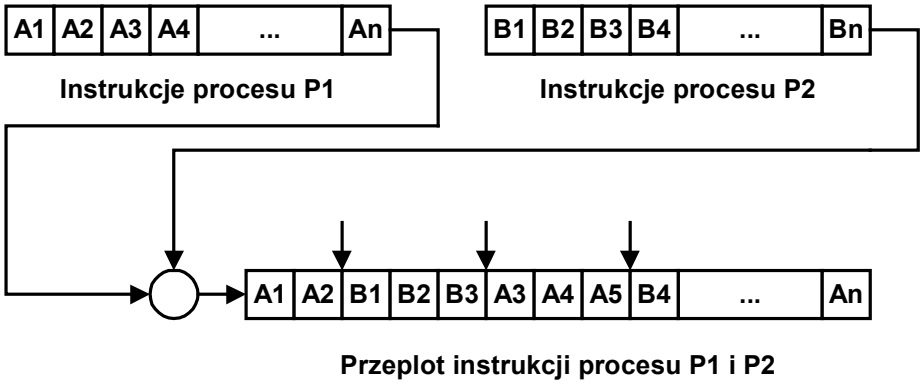
Przykład 2 - wyścigi

Aplikacja składa się z dwu procesów P1 i P2 mających dostęp do wspólnej zmiennej i.

Proces P1	Proces P2

i = 0;
while (i < 10) {
  i = i +1;
}
printf(„P1 – wygrał”);
	i = 0;
	while (i > - 10) {
	  i = i - 1;
	}
	printf(„P2 – wygrał”);

W powyższym przykładzie instrukcje atomowe kodu procesów P1 i P2 są przeplatane.



Rys. 1-12 Instrukcje procesów P1 i P2 wykonywane w trybie przeplotu

- Nie możemy poczynić żadnych założeń dotyczących momentów przełączenia procesów P1 i P2
- Nie da się określić wyniku działania powyższych procesów.

Wynik działania aplikacji współbieżnej nie może być uzależniony od sposobu przełączania procesów. Musi być prawidłowy dla wszystkich możliwych przeplotów.

Wzajemne wykluczanie

Wzajemne wykluczanie musi być zapewnione gdy kilka procesów ma dostęp do wspólnego obszaru pamięci i przynajmniej jeden z nich modyfikuje ten obszar.

1.7 Poprawność aplikacji współbieżnych

Rodzaje aplikacji:

1. Aplikacje transformacyjne – Procesy skończone które wykonują obliczenie czyli pobierają dane które mają przekształcić w wyniki. Kryterium poprawności: przekształcenie danych zgodnie ze specyfikacją w skończonym czasie

2. Aplikacja reaktywne – Wykonują się dowolnie długo (być może w nieskończoność) i ich celem jest interakcja z otoczeniem (wymiana danych). Kryterium poprawności: Prawidłowa interakcja z otoczeniem - czasowa i dotycząca przekształcania danych.

Kryterium poprawności aplikacji transformacyjnej

Aplikacja transformacyjnej jest poprawna jeżeli:
1. Zatrzymuje się
2. Jeżeli się zatrzyma to da poprawne wyniki

Typowe aplikacje reaktywne to:
- systemy operacyjne,
- aplikacje sterujące obiektami,
- serwery baz danych
- serwery WWW.

Aplikacje te nie kończą się, a nawet jeżeli, to nie jest ich zadaniem przedstawienie jakiegoś końcowego wyniku. Dla tego typu aplikacji ważniejsze są własności dynamiczne to znaczy zachowanie się aplikacji w czasie.

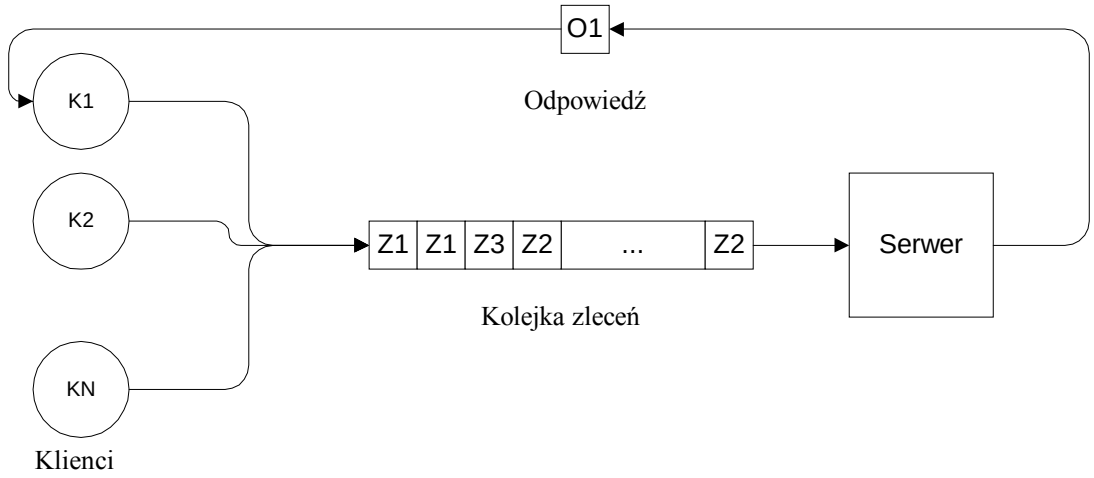
Ważne jest aby system operacyjny prawidłowo sterował komputerem i nie zawieszał się, program sterujący utrzymywał obiekt w pożądanym stanie a serwer bazy danych odpowiadał na zlecenia prawidłowo i w rozsądnym czasie.

Dla określenia prawidłowości aplikacji reaktywnych używa się pojęć:
- Bezpieczeństwa,
- Żywotności
- Uczciwości.

Bezpieczeństwo

Aplikacja jest bezpieczna jeżeli utrzymuje system w pożądanym stanie.

Aplikacja nie jest bezpieczna jeżeli:
- Da nieprawidłowe wyniki – np. nie jest zachowany warunek wzajemnego wykluczania
- Nie będzie wykonywał pożądanych działań - ulegnie blokadzie



Rys. 1-13 Model przetwarzania typu klient - serwer

W odniesieniu do modelu klient – serwer bezpieczeństwo oznacza że klienci są obsługiwani w zadowalający sposób.
1. Serwer nie zaprzestał obsługi zleceń.
2. Na zlecenia odpowiadał w prawidłowy sposób.

Blokada
Każdy z zablokowanych procesów oczekuje na zdarzenie które może być wygenerowane tylko przez któryś z zablokowanych procesów. Blokada zwana też zakleszczeniem jest typowym zagrożeniem aplikacji współbieżnych.

Zagłodzenie
Zagłodzenie występuje gdy procesowi cały czas odmawia się dostępu do zasobów których ten potrzebuje by wykonać zlecone mu zadanie.

Dla aplikacji reaktywnych nie formułuje się warunku zakończenia. Odpowiednikiem jest żywotność.

Żywotność
Aplikacja jest żywotna jeżeli każde pożądane zdarzenie w końcu zajdzie.

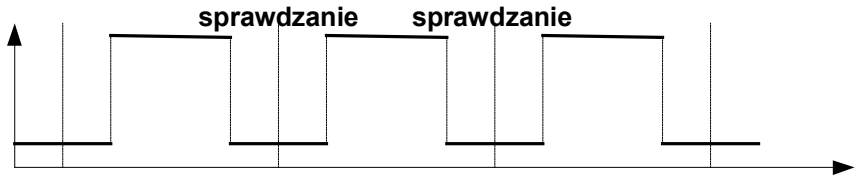
W modelu klient – serwer żywotność oznacza że każdy klient zostanie w końcu obsłużony.

Uczciwość
Aplikacja jest uczciwa jeżeli żądające obsługi procesy są traktowane jednakowo lub zgodnie ze swoimi priorytetami.

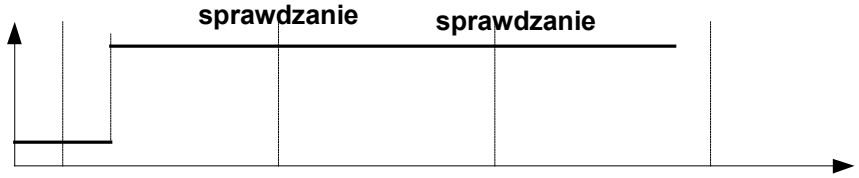
W modelu klient – serwer uczciwość oznacza że każdy klient zostanie obsłużony zgodnie z kolejnością zgłoszeń lub priorytetem.

Rodzaje uczciwości:

1. Uczciwość słaba – jeżeli proces nieprzerwanie zgłasza żądanie to kiedyś będzie ono obsłużone.
2. Uczciwość mocna – jeśli proces zgłasza żądanie nieskończenie wiele razy to w końcu zostanie ono obsłużone.
3. Uczciwość liniowa – jeśli proces zgłasza żądanie będzie ono obsłużone zanim dowolny inny proces będzie obsłużony więcej niż raz.
4. Uczciwość typu FIFO – żądania procesów są obsługiwane zgodnie z kolejnością ich zgłaszania. (FIFO – *ang. First-In First-Out)*



Uczciwość mocna – proces zgłasza żądanie nieskończoną ilość



Uczciwość słaba – proces musi zgłaszać żądanie nieprzerwanie

1.8 Skutki stosowania współbieżności

Korzyści wynikające z zastosowania współbieżności:

1. Polepszenie wykorzystania zasobów. Gdy jakiś proces czeka na niedostępny w danej chwili zasób, procesor może wykonywać inny proces.
2. Podział zadania na procesy umożliwia wykonywanie ich na oddzielnych maszynach. Prowadzi to do zrównoleglenia przetwarzania.
3. Podział dużego zadanie na wiele mniejszych komunikujących się procesów prowadzi do dekompozycji problemu. Przez co ułatwia ich implementację, uruchamianie i testowanie przez wielu niezależnych programistów.

Trudności powstające przy implementacji aplikacji współbieżnych:

- problem sekcji krytycznej
- problem synchronizacji procesów
- problem zakleszczenia

Procesy tworzące aplikację nie działają w izolacji. Muszą jakoś ze sobą współpracować co prowadzi do:

- Konieczności wzajemnej wymiany informacji - komunikacja międzyprocesowa.
- Zapewnienia określonej kolejności wykonania pewnych akcji - problem synchronizacji.

Przedmiot programowania współbieżnego

Metodologia tworzenia aplikacji składających się z wielu komunikujących się i dzielących zasoby procesów współbieżnych.