11 Monitory

Semafor nie jest mechanizmem strukturalnym. Aplikacje pisane z użyciem semaforów są podatne na błędy. Np. brak operacji sem_post blokuje aplikację.

Monitor (Brinch Hansen, Hoare) jest strukturalnym narzędziem synchronizacji procesów lub wątków.

Jest to obiekt którego metody mogą być bezpiecznie używane przez wiele procesów / wątków, gdyż wzajemne wykluczanie jest automatycznie zapewniane. Zapewnione jest także oczekiwanie na dostęp do jakiegoś zasobu które powoduje odblokowanie na ten czas monitora.

Monitory zaimplementowane w językach:
Java, Modula-3, Concurrent Pascal, Concurrent Euclid.

Definicja monitora

1. Zmienne i działające na nich procedury zebrane są w jednym module. Dostęp do zmiennych monitora możliwy jest tylko za pomocą procedur monitora.

2. W danej chwili tylko jeden proces wykonywać może procedury monitora. Gdy inny proces wywoła procedurę monitora będzie on zablokowany do chwili opuszczenia monitora przez pierwszy proces.

3. Istnieje możliwość wstrzymania i wznowienia procedur monitora za pomocą zmiennych warunkowych (*ang. conditional variables*). Na zmiennych warunkowych można wykonywać operacje **`Wait, Signal, Noempty`**.

`Wait(c)` - Wstrzymanie procesu bieżącego wykonującego procedurę monitora i wstawienie go na koniec kolejki związanej ze zmienną warunkową c. Jeżeli jakieś procesy czekają na wejście do monitora to jeden z nich będzie wpuszczony.

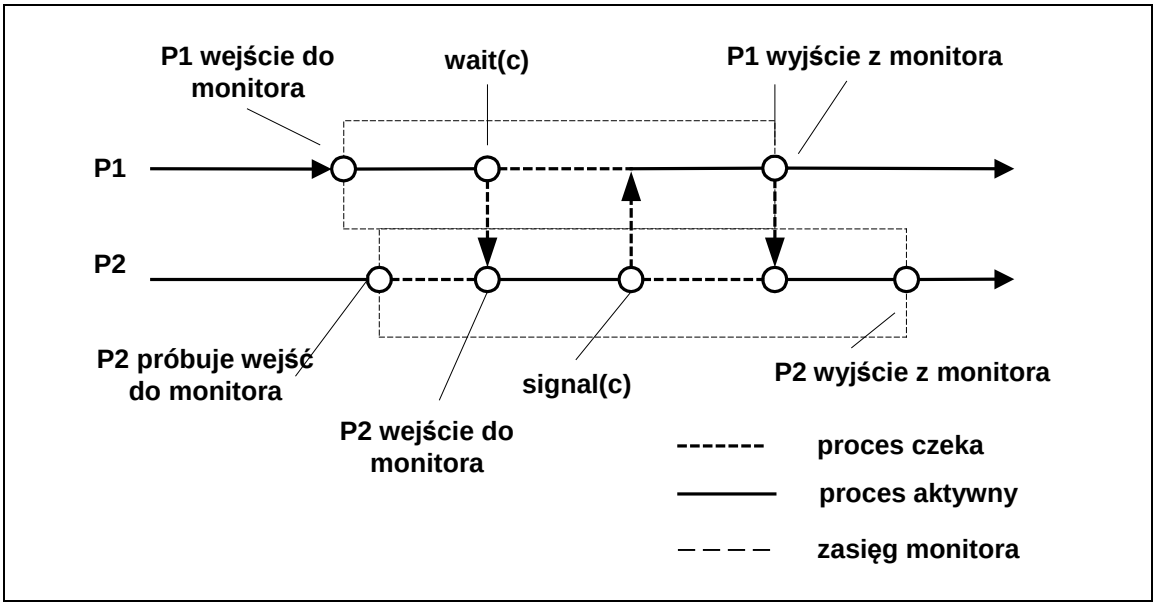
`Signal(c)` – Odblokowanie jednego z procesów czekających na zmiennej warunkowej c. Gdy brak czekających procesów operacja nie daje efektów. Operacja **`Signal`** jest bezpamięciowa (nie posiada licznika).

`Notempty(c)` – Funkcja zwraca true gdy kolejka c jest niepusta, false gdy pusta.

Jeśli **`Signal`** nie jest ostatnią instrukcją procedury monitora to proces wykonujący tę operację jest wstrzymany do chwili gdy wznowiony przezeń proces zwolni monitor. Wstrzymany tak proces może przebywać w:

- wejściowej kolejce procesów oczekujących na wejście do monitora
- kolejce uprzywilejowanej

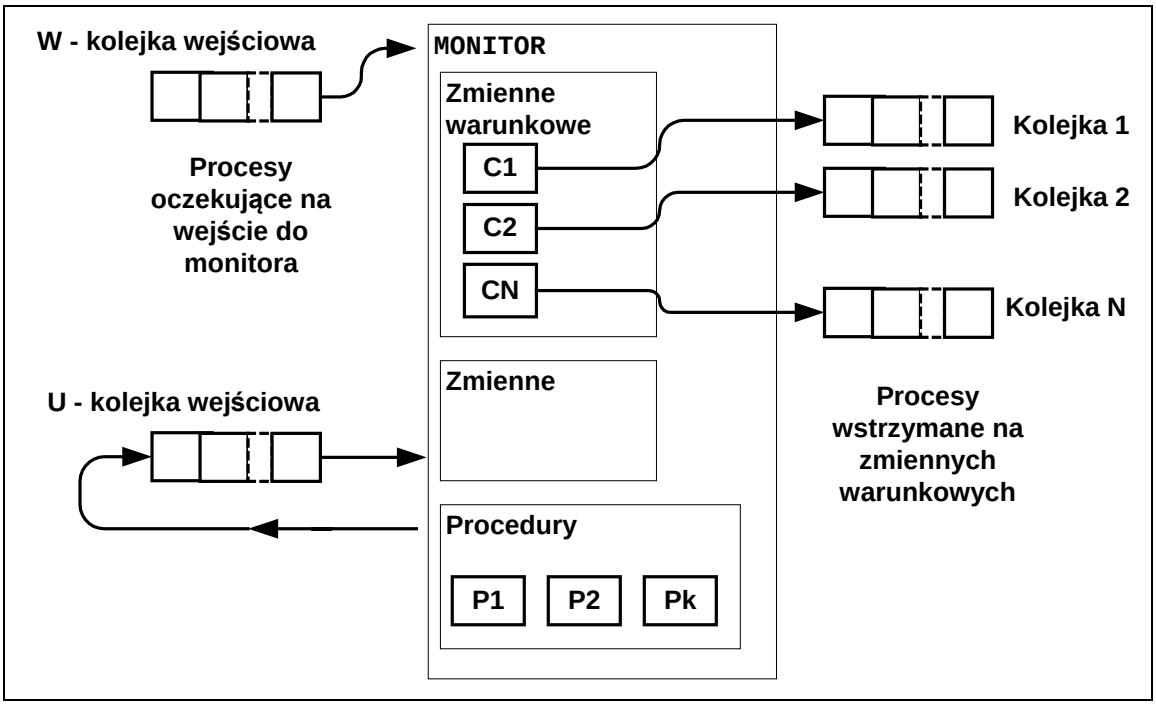
Które rozwiązanie zastosowano zależne jest od implementacji.



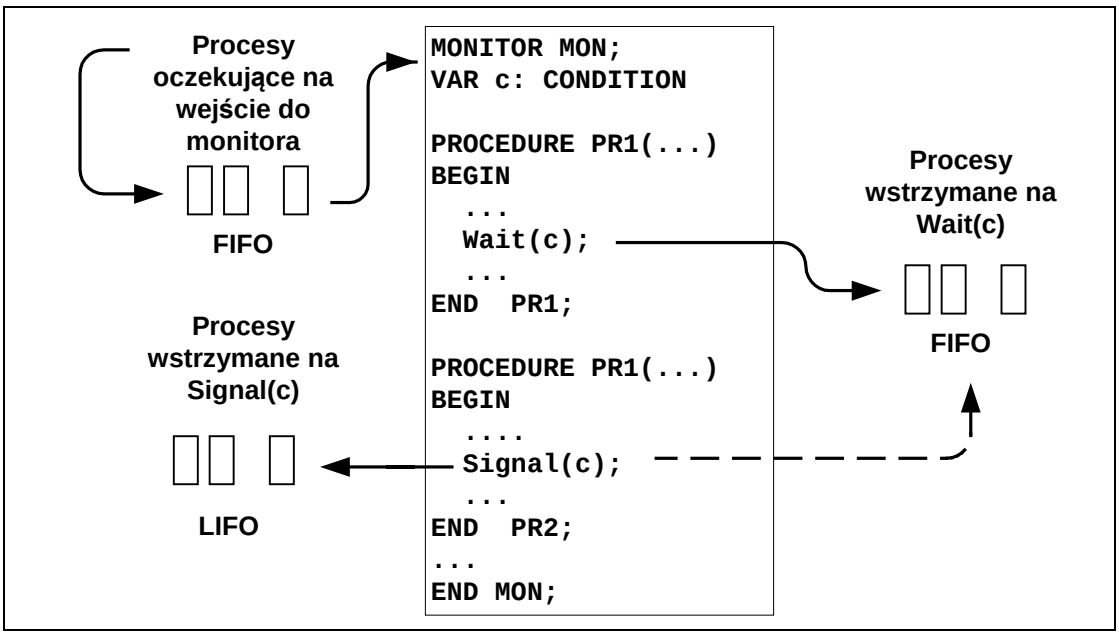
Rys. 11-1 Wewnątrz monitora znajduje się tylko jeden proces/wątek

Kolejki monitora:

- W - kolejka wejściowa
- U - kolejka uprzywilejowana
- C - kolejka zmiennej warunkowej, jest ich tyle ile zmiennych warunkowych



Rys. 11-2 Kolejki monitora



Rys. 11-3 Przykład monitora

Wejście do monitora

```
Gdy monitor jest wolny
    Zajmij monitor
Gdy monitor jest zajęty
    Dodaj bieżący wątek do kolejki W,
    Zablokuj bieżący wątek.
```

Wait(C)

```
Umieść bieżący wątek w kolejce C
Zablokuj bieżący wątek
Gdy w kolejce uprzywilejowanej U jest jakiś
wątek, usuń go z U i wznów,
Gdy w kolejce wejściowej W jest jakiś watek, usuń
go z W i wznów,
Gdy w żadnej z kolejek U i W nie ma wątków oznacz
monitor jako wolny
```

Signal(C)

```
Jeżeli jest jakiś wątek W czekający w kolejce C
Usuń W z kolejki C
Dodaj bieżący wątek do kolejki U
Zablokuj wątek bieżący
Odblokuj wątek  W
```

Wyjście z monitora

```
Gdy w kolejce U jest jakiś wątek, usuń go z U i
wznów,
Gdy w kolejce W jest jakiś watek, usuń go z W i
wznów,
Gdy w żadnej z kolejek U i W nie ma wątków oznacz
monitor jako wolny
```

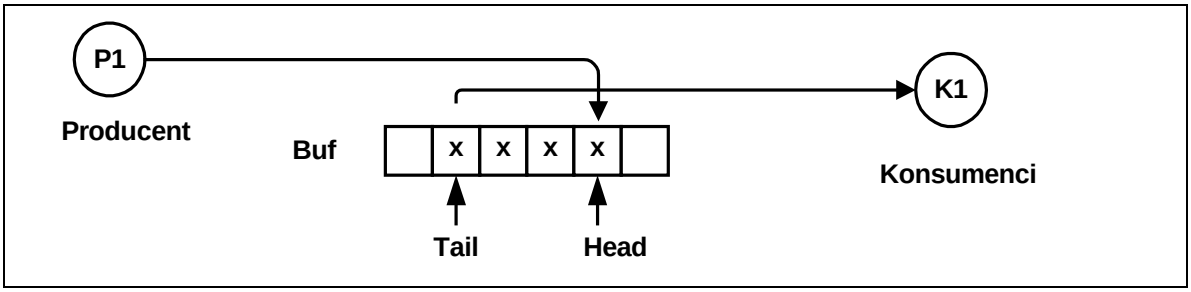
11.1.1 Rozwiązanie problemu wzajemnego wykluczania za pomocą monitorów

```
monitor WzajemneWykluczanie
int z1,z2;

Pisz(int x1, int x2){
  z1 = x1;
  z2 = x2;
};

Czytaj(int *x1, int *x2){
 *x1 := z1;
 *x2 := z2;
};
```

Przykład 11-1 Rozwiązanie problemu wzajemnego wykluczania za pomocą monitora

Procedury Czytaj lub Pisz wykonane będą w sposób wyłączny co wynika z definicji monitora.

11.2 Rozwiązanie problemu producenta i konsumenta za pomocą monitorów



Rys. 11-4 Problem producenta – konsumenta, bufor cykliczny

```
monitor ProducentKonsument
#define N 8
Record Buf[N];   // Bufor na rekordy
int Ile;         // Ile rekordów w buforze
int Head,Tail;   // Indeks wejściowy i wyjściowy
CONDITION Prod;  // Czekający producenci
CONDITION Cons;  // Czekający konsumenci

Wstaw(Record x ) {
  if(Ile == N) {
    Wait(Prod);
  }
  Head := (Head+1) % N;
  Buf(Head) = x;
  Ile ++;
  Signal(Cons);
 }
};

Pobierz(Record *x ) {
  if(Ile == 0) {
    Wait(Cons);
  }
  Tail := (Tail+1) % N;
  *x = Buf(Head)x;
  Ile --;
  Signal(Prod);
 }
};
```

Przykład 11-2 Rozwiązanie problemu producenta / konsumenta za pomocą monitora Hoare'a (monitor ze wznawianiem).

11.3 Rozwiązanie problemu czytelników i pisarzy za pomocą monitorów

```
monitor CzytelnicyPisarze
condition Pisanie, Czytanie;// Kolejka pisarzy i czytelników
int ReaderCount = 0;        // Liczba czytelników w czytelni
Boolean zajety = false;     // true gdy w czytelni jest pisarz

procedure StartRead()  {
    if (zajety)         // Gdy czytelnia zajęta - blokada
      Wait(Czytanie);
    ReaderCount++;
    Signal(Czytanie);  // Wpuść następnego czytelnika
}

EndRead()  {
     ReaderCount-- ;
     if(ReaderCount == 0) // Ostatni Czyt. wpuszcza pisarzy
        Signal(Pisanie);
}

StartWrite() {
    // Czekaj gdy w czytelni jest pisarz lub  czytelnicy
    if(zajety || ReaderCount != 0)
       Wait(Pisanie);
    zajety = true;
}

EndWrite() {
    zajety = false;
    // Gdy czekają czytelnicy to ich wpuść
    // Gdy czytelnicy nie czekają – wpuść pisarzy
    if (Notempty(Czytanie))  Signal(Czytanie);
    else                     Signal(Pisanie);
}
```

```
Reader() {
   while (true)  {
        StartRead();
        Czytaj(); // Wykonaj procedurę czytania
        EndRead();
   }
}
Writer() {
     while (true) {
        StartWrite();
        Pisz();  // Wykonaj procedurę pisania
        EndWrite();
     }
  }
```

Przykład 11-3 Rozwiązanie problemu czytelników i pisarzy za pomocą monitora.

11.4 Monitory z powiadamianiem i rozgłaszaniem

W omawianych poprzednio monitorach Hoara'a wystąpienie operacji **`Signal(c)`** musi spowodować natychmiastowe wznowienie procesu oczekującego na zmiennej warunkowej c.
Jest to trudne do realizacji ze względu na implementację szeregowania.

Monitory z powiadamianiem

Inne rozwiązanie stosowane jest w monitorach z powiadamianiem (wprowadzone -Lampson, Redell w języku Mesa). Zdefiniowana tam operacja **`Notify(c)`** powoduje odblokowanie jednego z procesów związanej ze zmienną warunkową **`c`** ale proces ten nie musi być natychmiast zaszeregowany.

Monitory z rozgłaszaniem

Dodana funkcja **`NotifyAll(c)`** która powoduje odblokowanie wszystkich z procesów związanej ze zmienną warunkową c. Proces te będą konkurowały o dostęp do monitora. Będą zaszeregowane gdy będzie to możliwe.

Nie występuję tu kolejka uprzywilejowana U, system operacyjny może stosować standardowe szeregowanie.

Wejście do monitora

Gdy monitor jest wolny
Zajmij monitor
Gdy monitor jest zajęty
Dodaj bieżący wątek do kolejki W,
Zablokuj bieżący wątek

Wait(C)

Umieść bieżący wątek w kolejce C,
Zablokuj bieżący wątek,
Gdy w kolejce wejściowej W jest jakiś wątek, usuń go z W i wznów,
Gdy w nie oznacz monitor jako wolny

Notify(C)

Jeżeli jest jakiś wątek P czekający w kolejce C
Usuń go z kolejki C i dodaj do kolejki wejściowej W.

NotifyAll(c)

Jeżeli są jakie wątki czekające w kolejce C
Usuń je wszystkie z kolejki C i dodaj do kolejki wejściowej W

Wyjście z monitora

Gdy w kolejce W jest jakiś wątek, usuń go z W i wznów, gdy nie oznacz monitor jako wolny.

```
monitor ProducentKonsument
#define N 8
Record Buf[N]; // Bufor na rekordy
int Ile;           // Ile rekordów w buforze
int Head,Tail;     // Indeks wejściowy i wyjściowy
CONDITION Prod;    // Czekający producenci
CONDITION Cons;  // Czekający konsumenci

Wstaw(Record x ) {
 while(Ile == N)  Wait(Prod);
 Head := (Head+1) % N;
 Buf(Head) = x;
 Ile ++;
 Notify(Cons);
};

Pobierz(Record *x ) {
 while(Ile == 0) Wait(Cons);
 Tail := (Tail+1) % N;
 *x = Buf(Head)x;
 Ile --;
 Notify (Prod);
};
```

Przykład 11-4 Rozwiązanie problemu producenta / konsumenta za pomocą monitora z powiadamianiem

11.5 Monitory w Javie

Monitory z powiadamianiem zaimplementowane są w języku Java jako obiekty synchronizowane. Poprzedza się je słowem kluczowym **`synchronized`**.

- Każde wejście do monitora oznaczone jest słowem kluczowym **`synchronized`**.
- Monitor związany jest z obiektem a nie klasą.

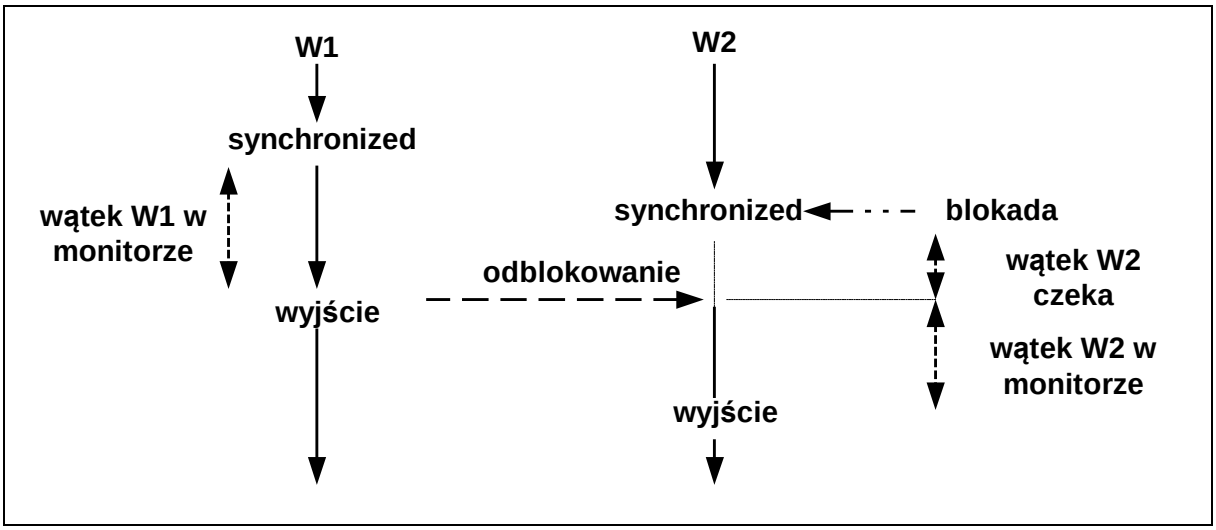
Gdy wątek przechodzi instrukcje oznaczone jako **`synchronized`** to tak jakby zamykał drzwi do monitora.

Synchronizacja w Javie może być wykonana na poziomie:

1. Metod – słowo kluczowe **`synchronized`** występuje przy definiowaniu metody – np.:
   ```
   public synchronized int get() {...}
   ```

2. Obiektów i bloków - słowo kluczowe **`synchronized`** występuje przy definiowaniu bloku instrukcji – np.

   ```
   synchronized(p) {
      safeX = p.x;
      safeY = p.y;
   }

   ...
   synchronized(this) {
   ...
    }
   ```



Rys. 11-5 Dostęp do obiektu synchronizowany przez monitor

Zapewnienie wzajemnego wykluczania metod synchronizowanych nie wystarczy do rozwiązania wielu zagadnień synchronizacji. Co zrobić gdy wewnątrz synchronizowanej metody nastąpi konieczność oczekiwania na pewien zasób?.

Jakby zastosować aktywne czekanie to inne wątki miały by zablokowany dostęp do monitora. Konieczna jest implementacja metody wstrzymywania i wznawiania wątku wewnątrz monitora.

Java nie udostępnia jawnie zmiennych warunkowych. Dla obiektu synchronizowanego automatycznie tworzona jest jedna zmienna warunkowa dostępna przez metody **`wait`**, **`notify`** i **`notifyAll`**. Kolejne mogą być wprowadzone przez obiekty klasy **`Lock`**.

Metoda **wait()**

```
public final void wait();
public final void wait(long timeout);
public final void wait(long timeout,int nanos)
                throws InterruptedException
```

Wykonanie metody powoduje zawieszenie bieżącego wątku do czasu gdy inny watek nie wykona metody **notify**() lub **notifyAll**() odnoszącej się do wątku w który wykonał **wait**().

Uwaga!
Wątek wykonujący **wait(...)** musi być w posiadaniu monitora dotyczącego synchronizowanego obiektu.

Wykonanie **wait(...)** powoduje zwolnienie monitora.

Metoda **notify()**

```
public final void notify()
```

Metoda powoduje odblokowanie jednego z wątków zablokowanych na monitorze pewnego obiektu poprzez **wait()**. Który z czekających wątków będzie odblokowany nie jest w definicji metody określone.

Uwaga!
Odblokowany wątek nie będzie natychmiast wykonywany – musi on jeszcze zaczekać, aż zwolniona będzie przez bieżący wątek blokada monitora. Odblokowany wątek będzie konkurował z innymi o nabycie blokady monitora.

Uwaga!
Metoda **notify()** i **notifyAll** może być wykonana tylko przez wątek który jest właścicielem monitora obiektu.

Wątek staje się właścicielem monitora obiektu gdy:
- Wykona synchronizowaną metodę instancji tego obiektu.
- Wykona synchronizowane wyrażenia tego obiektu.

Gdy wątek nie jest w posiadaniu monitora obiektu generowany jest wyjątek. **`IllegalMonitorStateException`**.

Metoda **`notifyAll()`**

```
public final void notifyAll()
```

Metoda powoduje odblokowanie wszystkich wątków zablokowanych na monitorze pewnego obiektu poprzez uprzednie wykonanie **`wait()`**.

Uwaga!
Wątki będą jednak czekały aż do chwili gdy wątek bieżący zwolni blokady monitora. Odblokowane wątki będą konkurowały o nabycie blokady monitora.

Literatura:
[1] http://pl.wikipedia.org/wiki/Monitor_programowanie_wsp