

Systemy operacyjne

## Systemowe mechanizmy synchronizacji procesów

Dariusz Wawrzyniak



Systemy operacyjne

Semafor



- Semafor jest zmienną całkowitą nieujemną lub — w przypadku semaforów binarnych — zmienną typu logicznego.
- Na semaforze można wykonywać dwa rodzaje operacji:
  - P — opuszczanie semafora (hol. proberen)
  - V — podnoszenie semafora (hol. verhoggen)
- Synchronizacja polega na blokowaniu procesu w operacji opuszczania semafora, jeśli semafor jest już opuszczony.

Systemowe mechanizmy synchronizacji procesów (3)

Systemy operacyjne

Plan wykładu



- Definicja semafora
- Klasyfikacja semaforów
- Implementacja semaforów
- Zamki
- Zmienne warunkowe
- Klasyczne problemy synchronizacji

Systemowe mechanizmy synchronizacji procesów (2)

Systemy operacyjne

Rodzaje semaforów <sup>(1)</sup>

- Semafor binarny — zmienna semaforowa przyjmuje tylko wartości true (stan podniesienia, otwarcia) lub false (stan opuszczenia, zamknięcia).
- Semafor ogólny (zliczający) — zmienna semaforowa przyjmuje wartości całkowite nieujemne, a jej bieżąca wartość jest zmniejszana lub zwiększana o 1 w wyniku wykonania odpowiednio operacji opuszczenia lub podniesienia semafora.

Systemowe mechanizmy synchronizacji procesów (4)

Systemy operacyjne

Rodzaje semaforów (2)

- Semafor uogólniony — semafor zliczający, w przypadku którego zmienną semaforową można zwiększać lub zmniejszać o dowolną wartość, podaną jako argument operacji.
- Semafor dwustronnie ograniczony — semafor ogólny, w przypadku którego zmienna semaforowa, oprócz dolnego ograniczenia wartością 0, ma górne ograniczenie, podane przy definicji semafora.

Systemowe mechanizmy synchronizacji procesów (5)

Systemy operacyjne

Implementacja semafora binarnego na poziomie maszynowym

- Opuszczanie semafora
 

```
procedure P(var s: Binary_Semaphore)
begin
  while not s do nic; } instrukcje muszą być
  s := false;         } wykonane atomowo
end;
```
- Podnoszenie semafora
 

```
procedure V(var s: Binary_Semaphore)
begin
  s := true;          } instrukcja musi być
end;                  } wykonana atomowo
```

Systemowe mechanizmy synchronizacji procesów (7)

Systemy operacyjne

Implementacja semafora ogólnego na poziomie maszynowym

- Opuszczanie semafora
 

```
procedure P(var s: Semaphore)
begin
  while s = 0 do nic; } instrukcje muszą być
  s := s - 1;         } wykonane atomowo
end;
```
- Podnoszenie semafora
 

```
procedure V(var s: Semaphore)
begin
  s := s + 1;        } instrukcja musi być
end;                 } wykonana atomowo
```

Systemowe mechanizmy synchronizacji procesów (6)

Systemy operacyjne

Implementacja semafora ogólnego na poziomie systemu operacyjnego (1)


- Struktury danych
 

```
type Semaphore = record
  wartość: Integer;
  L: list of Proces;
end;
```
- Opuszczanie semafora
 

```
procedure P(var s: Semaphore) begin
  s.wartość := s.wartość - 1;
  if s.wartość < 0 then begin
    dołącz dany proces do s.L
    zmień stan danego procesu na „oczekujący”
  end;
end;
```

Systemowe mechanizmy synchronizacji procesów (8)

Systemy operacyjne



**Implementacja semafora ogólnego na poziomie systemu operacyjnego** (2)

- Podnoszenie semafora


```

procedure V (var s: Semaphore)
  begin
    s.wartość := s.wartość + 1;
    if s.wartość ≤ 0 then begin
      wybierz i usuń jakiś/kolejny proces z kolejki s.L
      zmień stan wybranego procesu na „gotowy”
    end;
  end;

```

Systemowe mechanizmy synchronizacji procesów (9)

Systemy operacyjne




**Mechanizmy synchronizacji POSIX — zmienne synchronizujące**

- Rodzaje zmiennych synchronizujących:
  - zamek — umożliwiającą implementację wzajemnego wykluczenia,
  - zmienna warunkowa — umożliwia usypianie i budzenie wątków.
- Zmienne synchronizujące muszą być współdzielone przez synchronizowane wątki.
- Zanim zmienna zostanie wykorzystana do synchronizacji musi zostać zainicjalizowana.

Systemowe mechanizmy synchronizacji procesów (11)

Systemy operacyjne



**Wzajemne wykluczenie z użyciem semaforów**


```

shared mutex: Semaphore := 1;
P(mutex);      ← sekcja wejściowa
sekcja krytyczna;
V(mutex);      ← sekcja wyjściowa
reszta;

```

Systemowe mechanizmy synchronizacji procesów (10)

Systemy operacyjne




**Operacje na zmiennych synchronizujących**

- Zamek — umożliwia implementację wzajemnego wykluczenia. Operacje:
  - lock — zajęcie (zaryglowanie) zamka
  - unlock — zwolnienie (odryglowanie) zamka
  - trylock — nieblokująca próba zajęcia zamka
- Zmienna warunkowa — umożliwia usypianie i budzenie wątków. Operacje:
  - wait — usypianie wątku,
  - signal — obudzenie jednego z uspiących wątków
  - broadcast — obudzenie wszystkich uspiących wątków

Systemowe mechanizmy synchronizacji procesów (12)

Systemy operacyjne




**Zamek — interfejs**

- Typ: `pthread_mutex_t`
- Operacje:
  - `pthread_mutex_lock(pthread_mutex_t *m)`  
— zajęcie zamka
  - `pthread_mutex_unlock(pthread_mutex_t *m)`  
— zwolnienie zamka
  - `pthread_mutex_trylock(pthread_mutex_t *m)` — próba zajęcia zamka w sposób nie blokujący wątku w przypadku niepowodzenia

Systemowe mechanizmy synchronizacji procesów (13)

Systemy operacyjne




**Zmienna warunkowa — interfejs**

- Typ: `pthread_cond_t`
- Operacje:
  - `pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m)` — uśpienie wątku na zmiennej warunkowej,
  - `pthread_cond_signal(pthread_cond_t *c)` — obudzenie jednego z wątków uśpionych na zmiennej warunkowej,
  - `pthread_cond_broadcast(pthread_cond_t *c)` — obudzenie wszystkich wątków uśpionych na zmiennej warunkowej.

Systemowe mechanizmy synchronizacji procesów (15)

Systemy operacyjne




**Zamek — implementacja**

- `pthread_mutex_lock`
  - zajęcie zamka, jeśli jest wolny
  - ustawienie stanu wątku na oczekujący i umieszczenie w kolejce, jeśli zamek jest zajęty
- `pthread_mutex_unlock`
  - ustawienie zamka na wolny, jeśli kolejka oczekujących wątków jest pusta
  - wybranie wątku z niepustej kolejki wątków oczekujących i ustawienie jego stanu na gotowy.
- `pthread_mutex_trylock`
  - zajęcie zamka lub kontynuacja przetwarzania

Systemowe mechanizmy synchronizacji procesów (14)

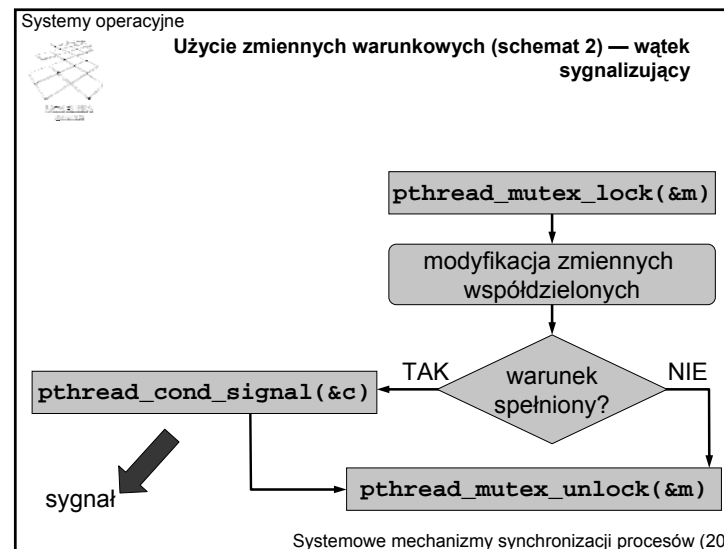
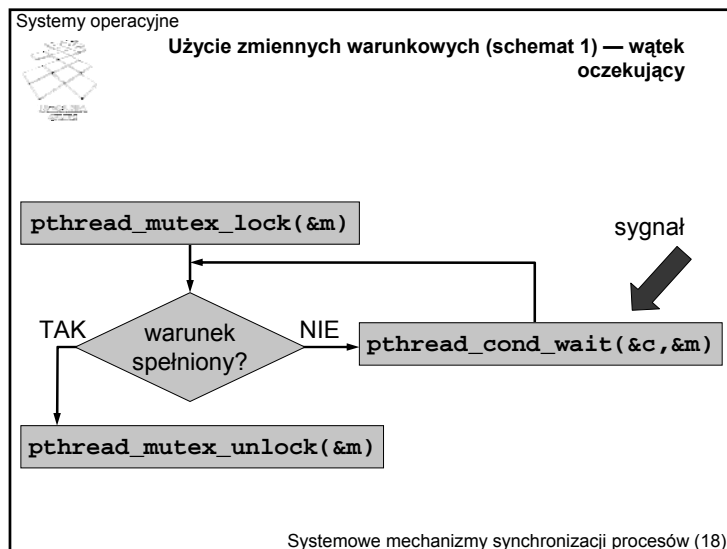
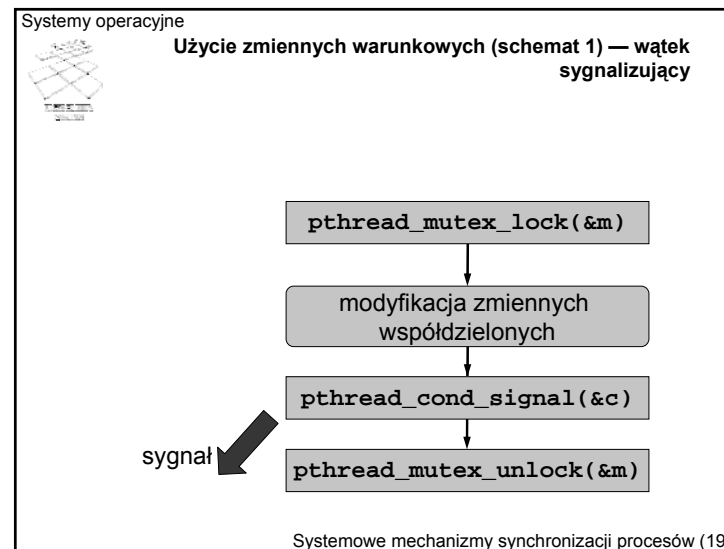
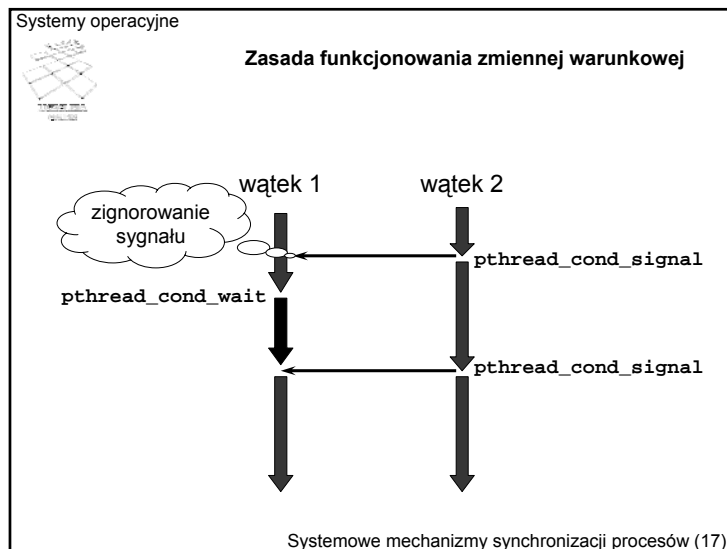
Systemy operacyjne

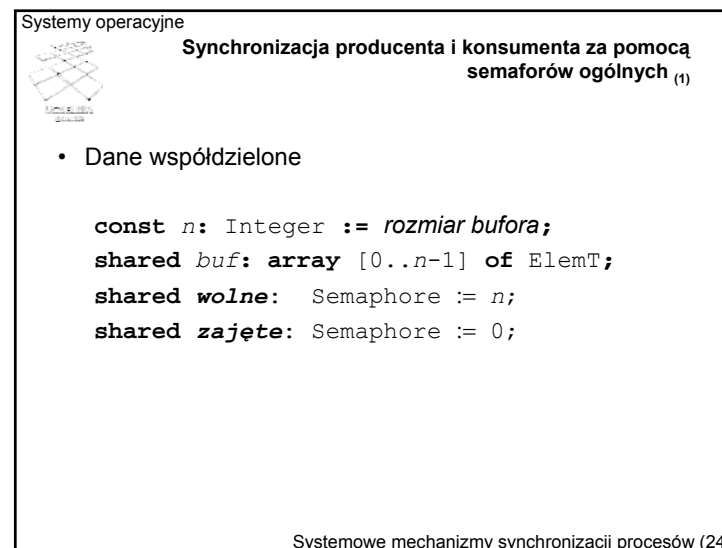
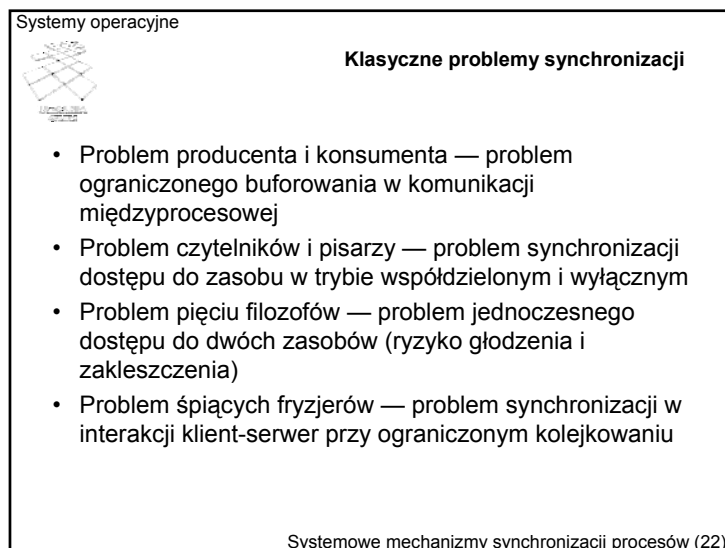
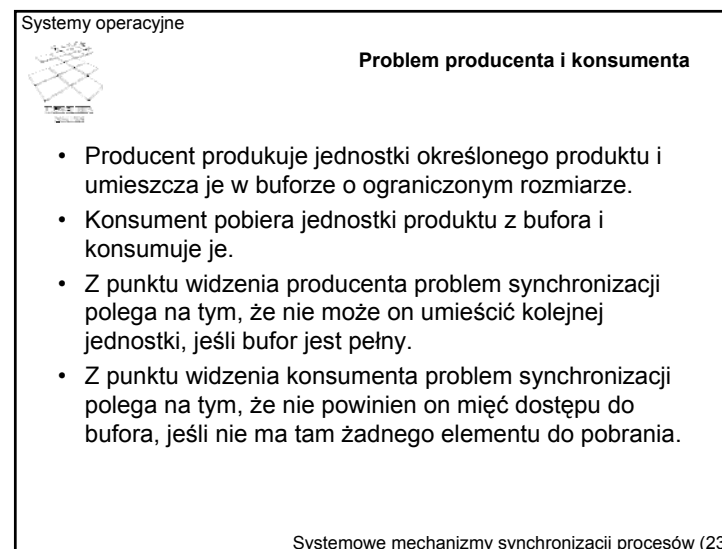
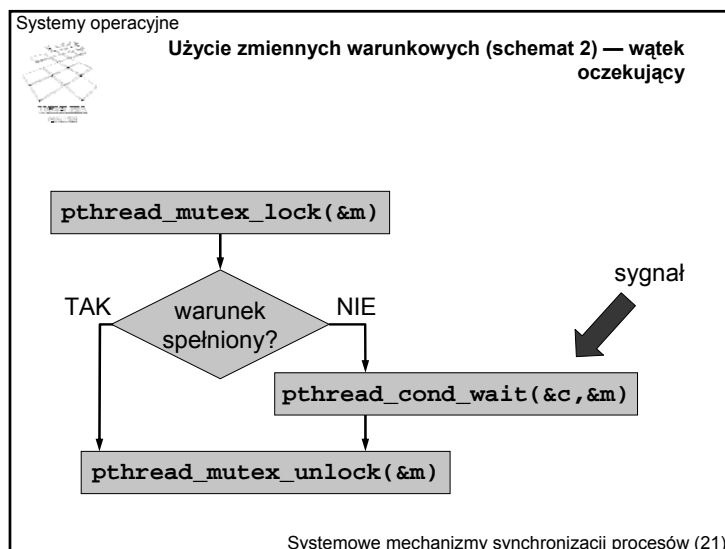


**Zmienna warunkowa — implementacja**


- `pthread_cond_wait`
  - ustawienie stanu wątku na oczekujący i umieszczenie go w kolejce
- `pthread_cond_signal`
  - wybranie jednego wątku z kolejki i postępowanie takie, jak przy zajęciu zamka
  - zignorowanie sygnału, jeśli kolejka jest pusta
- `pthread_cond_broadcast`
  - ustawienie wszystkich wątków oczekujących na zmiennej warunkowej w kolejce do zajęcia zamka, a jeśli zamek jest wolny zmiana stanu jednego z nich na gotowy.

Systemowe mechanizmy synchronizacji procesów (16)





Systemy operacyjne



**Synchronizacja producenta i konsumenta za pomocą semaforów ogólnych** <sup>(2)</sup>

- Producent


```

local i: Integer := 0;
local elem: ElemT;
while ... do begin
  produkuj(elem);
  P(wolne);
  buf[i] := elem;
  i := (i+1) mod n;
  V(zajete);
end;

```

Systemowe mechanizmy synchronizacji procesów (25)

Systemy operacyjne




**Problem czytelników i pisarzy**

- Dwa rodzaje użytkowników — czytelnicy i pisarze — korzystają ze wspólnego zasobu — czytelnicy.
- Czytelnicy korzystają z czytelnicy w trybie współdzielonym, tzn. w czytelnicy może przebywać w tym samym czasie wielu czytelników.
- Pisarze korzystają z czytelnicy w trybie wyłącznym, tzn. w czasie, gdy w czytelnicy przebywa pisarz, nie może z niej korzystać inny użytkownik (ani czytelnik, ani inny pisarz).
- Synchronizacja polega na blokowaniu użytkowników przy wejściu do czytelnicy, gdy wymaga tego tryb dostępu.

Systemowe mechanizmy synchronizacji procesów (27)

Systemy operacyjne



**Synchronizacja producenta i konsumenta za pomocą semaforów ogólnych** <sup>(3)</sup>

- Konsument


```

local i: Integer := 0;
local elem: ElemT;
while ... do begin
  P(zajete);
  elem := buf[i];
  i := (i+1) mod n;
  V(wolne);
  konsumuj(elem);
end;

```

Systemowe mechanizmy synchronizacji procesów (26)

Systemy operacyjne



**Synchronizacja czytelników i pisarzy za pomocą semaforów binarnych** <sup>(1)</sup>

- Dane współdzielone


```

shared l_czyt: Integer := 0;
shared mutex_r: Binary_Semaphore := true;
shared mutex_w: Binary_Semaphore := true;

```

Systemowe mechanizmy synchronizacji procesów (28)

Systemy operacyjne



**Synchronizacja czytelników i pisarzy za pomocą semaforów binarnych** <sup>(2)</sup>

- Czytelnik


```

while ... do begin
  P(mutex_r);
  l_czyt := l_czyt + 1;
  if l_czyt = 1 then P(mutex_w);
  V(mutex_r);
  czytanie;
  P(mutex_r);
  l_czyt := l_czyt - 1;
  if l_czyt = 0 then V(mutex_w);
  V(mutex_r);
end;

```

Systemowe mechanizmy synchronizacji procesów (29)

Systemy operacyjne




**Problem pięciu filozofów**

- Przy okrągłym stole siedzi pięciu filozofów, którzy na przemian myślą (filozofują) i jedzą makaron ze wspólnej miski.
- Żeby coś zjeść, filozof musi zdobyć dwa widelce, z których każdy współdzieli ze swoim sąsiadem.
- Widelec dostępny jest w trybie wyłącznym — może być używany w danej chwili przez jednego filozofa.
- Należy zsynchronizować filozofów tak, aby każdy mógł się w końcu najeść przy zachowaniu reguł dostępu do widelców oraz przy możliwie dużej przepustowości w spożywaniu posiłków.

Systemowe mechanizmy synchronizacji procesów (31)

Systemy operacyjne



**Synchronizacja czytelników i pisarzy za pomocą semaforów binarnych** <sup>(3)</sup>

- Pisarz


```

while ... do
  P(mutex_w);
  pisanie;
  V(mutex_w);
end;

```

Systemowe mechanizmy synchronizacji procesów (30)

Systemy operacyjne



**Synchronizacja 5 filozofów za pomocą semaforów binarnych** <sup>(1)</sup>


```

shared dopuść: Semaphore := 4;
shared widelec: array[0..4] of
  Binary_Semaphore := true;

```

Systemowe mechanizmy synchronizacji procesów (32)

Systemy operacyjne



**Synchronizacja 5 filozofów za pomocą semaforów binarnych** <sup>(2)</sup>


- Filozof nr  $i$  ( $i = 0 \dots 4$ )
 

```

while ... do begin
    myślenie;
    P(dopusć);
    P(widelec[i]);
    P(widelec[(i+1) mod 5]);
    jedzenie;
    V(widelec[i]);
    V(widelec[(i+1) mod 5]);
    V(dopusć);
end;
      
```

Systemowe mechanizmy synchronizacji procesów (33)

Systemy operacyjne



**Synchronizacja śpiących fryzjerów za pomocą semaforów** <sup>(1)</sup>


- Dane współdzielone
 

```

const p: Integer := pojemność poczekalni;
const n: Integer := liczba foteli;
shared l_czek: Integer := 0;
shared mutex: Binary_Semaphore := true;
shared klient: Semaphore := 0;
shared fryzjer: Semaphore := 0;
shared fotel: Semaphore := n;
      
```

Systemowe mechanizmy synchronizacji procesów (35)

Systemy operacyjne




**Problem śpiących fryzjerów**

- W salonie fryzjerskim jest poczekalnia z  $p$  miejscami oraz  $n$  foteli, obsługiwanych przez fryzjerów.
- Do salonu przychodzi klient, budzi fryzjera, po czym fryzjer znajduje wolny fotel i obsługuje klienta.
- Jeśli nie ma wolnego fotela, klient zajmuje jedno z wolnych miejsc w poczekalni.
- Jeśli nie ma miejsca w poczekalni, klient odchodzi.
- Problem polega na zsynchronizowaniu fryzjerów oraz klientów w taki sposób, aby jeden fryzjer w danej chwili obsługiwał jednego klienta i w tym samym czasie klient był obsługiwany przez jednego fryzjera.

Systemowe mechanizmy synchronizacji procesów (34)

Systemy operacyjne



**Synchronizacja śpiących fryzjerów za pomocą semaforów** <sup>(2)</sup>


- Klient
 

```

while ... do begin
    P(mutex);
    if l_czek < p then begin
        l_czek := l_czek + 1;
        V(klient);
        V(mutex);
        P(fryzjer);
        strzyżenie;
    end
    else V(mutex);
end;
      
```

Systemowe mechanizmy synchronizacji procesów (36)

Systemy operacyjne



**Synchronizacja śpiących fryzjerów za pomocą semaforów** (3)

- Fryzjer


```

while ... do begin
  P(klient);
  P(fotel);
  P(mutex);
  l_czek := l_czek - 1;
  V(fryzjer);
  V(mutex);
  strzyżenie;
  V(fotel);
end;

```

Systemowe mechanizmy synchronizacji procesów (37)

Systemy operacyjne



**Ogólny schemat definicji monitora**


```

type nazwa_monitora = monitor
  deklaracje_zmiennych
  procedure entry proc_1(...);
    begin ... end;
  :
  procedure entry proc_n(...);
    begin ... end;
begin
  kod_inicjalizujący
end.

```

Systemowe mechanizmy synchronizacji procesów (39)

Systemy operacyjne




**Monitory**

- Monitory jest strukturalnym mechanizmem synchronizacji, którego definicja monitora obejmuje:
  - hermetycznie zamkniętą definicję zmiennych (pól),
  - definicję wejść, czyli procedur umożliwiających wykonywanie operacji na polach monitora.
- Wewnątrz monitora może być aktywny co najwyżej jeden proces.
- Proces może zostać uśpiony wewnątrz monitora na zmiennej warunkowej.
- Uśpiony proces może zostać obudzony przez wysłanie sygnału związanego z daną zmienną warunkową.

Systemowe mechanizmy synchronizacji procesów (38)

Systemy operacyjne



**Ograniczony bufor cykliczny — definicja oparta na monitorze** (1)


```

type Buffer = monitor
  pula : array [0..n-1] of ElemT;
  wej, wyj, licz : Integer;
  pusty, pełny : Condition;

```

Systemowe mechanizmy synchronizacji procesów (40)

Systemy operacyjne



**Ograniczony bufor cykliczny — definicja oparta na monitorze** (2)


```

procedure entry wstaw(elem: ElemT);
begin
  if licz = n then pełny.wait;
  pula[wej] := elem;
  wej := (wej + 1) mod n;
  licz := licz + 1;
  pusty.signal;
end;

```

Systemowe mechanizmy synchronizacji procesów (41)

Systemy operacyjne



**Ograniczony bufor cykliczny — definicja oparta na monitorze** (4)


```

begin
  wej := 0;
  wyj := 0;
  licz := 0;
end.

```

Systemowe mechanizmy synchronizacji procesów (43)

Systemy operacyjne



**Ograniczony bufor cykliczny — definicja oparta na monitorze** (3)


```

procedure entry pobierz(var elem: ElemT);
begin
  if licz = 0 then pusty.wait;
  elem := pula[wyj];
  wyj := (wyj + 1) mod n;
  licz := licz - 1;
  pełny.signal;
end;

```

Systemowe mechanizmy synchronizacji procesów (42)

Systemy operacyjne



**Synchronizacja producenta i konsumenta za pomocą monitora**

- Dane współdzielone

```

shared buf: Buffer;

```

- Producent

```

local
  elem: ElemType;
while ... do begin
  produkuj(elem);
  buf.wstaw(elem);
end;

```

- Konsument


```

local
  elem: ElemType;
while ... do begin
  buf.pobierz(elem);
  konsumuj(elem);
end;

```

Systemowe mechanizmy synchronizacji procesów (44)

Systemy operacyjne




**Synchronizacja czytelników i pisarzy za pomocą monitora**

- Dane współdzielone  
`shared czytelnia: Monitor;`
- Czytelnik  
`czytelnia.wejście_czytelnika;`  
`czytanie;`  
`czytelnia.wyjście_czytelnika;`
- Pisarz  
`czytelnia.wejście_pisarza;`  
`pisanie;`  
`czytelnia.wyjście_pisarza;`

Systemowe mechanizmy synchronizacji procesów (45)

Systemy operacyjne




**Synchronizacja producenta i konsumenta za pomocą regionu krytycznego <sup>(1)</sup>**

- Dane współdzielone

```
shared buf: record
  pula : array [0..n-1] of ElemT;
  wej, wyj, licz : Integer;
end;
```

Systemowe mechanizmy synchronizacji procesów (47)

Systemy operacyjne




**Regiony krytyczne**

- Region krytyczny jest fragmentem programu — oznaczonym jako  $S$ , wykonywanym przy wyłącznym dostępie do pewnej zmiennej współdzielonej, wskazanej w jego definicji — oznaczonej jako  $v$ .
- Wykonanie regionu krytycznego uzależnione jest od wyrażenia logicznego —  $B$ , a przetwarzanie blokowane jest do momentu, aż wyrażenie będzie prawdziwe.

```
shared v: T;
region v when B do S;
```

Systemowe mechanizmy synchronizacji procesów (46)

Systemy operacyjne




**Synchronizacja producenta i konsumenta za pomocą regionu krytycznego <sup>(2)</sup>**

- Producent

```
local elem: ElemT;
while ... do
  begin
    produkuj(elem);
    region buf when buf.licz < n do
      begin
        buf.pula[wej] := elem;
        buf.wej := (buf.wej+1) mod n;
        buf.licz := buf.licz + 1;
      end;
    end;
  end;
```

Systemowe mechanizmy synchronizacji procesów (48)

Systemy operacyjne



**Synchronizacja producenta i konsumenta za pomocą regionu krytycznego** <sup>(3)</sup>

- Konsument

```
local elem: ElemT;
while ... do
begin
  region buf when buf.licz > 0 do
  begin
    elem := buf.pula[wyj];
    buf.wyj := (buf.wyj+1) mod n;
    buf.licz := buf.licz - 1;
  end;
  konsumuj(elem);
end;
```

Systemowe mechanizmy synchronizacji procesów (49)