Ada 95 — współbieżność

Współpraca (synchronizacja i komunikacja) zadań za pomocą spotkań asymetrycznych i obiektów chronionych

Współbieżność w Adzie — zadania

- Zadania są jednostkami strukturalizacji programów współbieżnych.
- Zadanie jest obiektem typu zadaniowego. W programie można utworzyć wiele zadań na podstawie tego samego typu zadaniowego. Wszystkie obiekty danego typu mają jednakową specyfikację (interfejs) i tę samą implementację.
- Obiekty zadaniowe są tworzone tak jak inne obiekty języka: mogą być deklarowane statycznie lub kreowane dynamicznie.

Komunikacja pomiędzy zadaniami

- komunikacja synchroniczna między parą zadań — mechanizm spotkań asymetrycznych (ang. rendez-vous),
- komunikacja asynchroniczna między wieloma zadaniami — obiekty chronione

Specyfikacja zadania

```
task Zad is
   ...
   entry E1(p1: in t1; p2: out t2;);
   ...
private
   entry E2(...);
   ...
end Zad;
```

Implementacja zadania

```
task body Zad is
    ...
accept E1(p1: in t1; p2: out t2;)
    do
       ...
    end E1;
    ...
accept E2(...)
    do
       ...
    end E2;
    ...
end Zad;
```

Specyfikacja typu zadaniowego

```
task type T is
   ...
   entry E1(...);
   ...
end T;
```

Przykład zastosowania typu zadaniowego

```
tablica: array(1..5) of T;

type R is
record
   Zadanie: T;
   ...
end record;
...
R1, R2 : R;
```

Spotkania asymetryczne

- W komunikacji między zadaniami z wykorzystaniem mechanizmu spotkań istotne jest wyróżnienie roli, jaką może w danej chwili pełnić zadanie — asymetria.
- W różnych momentach czasu zadanie może być bierne (serwer) — jeśli udostępnia lub jest gotowe udostępnić usługi identyfikowane przez nazwy wejść – lub czynne (klient) — jeśli wywołuje właśnie wejście jakiegoś serwera.

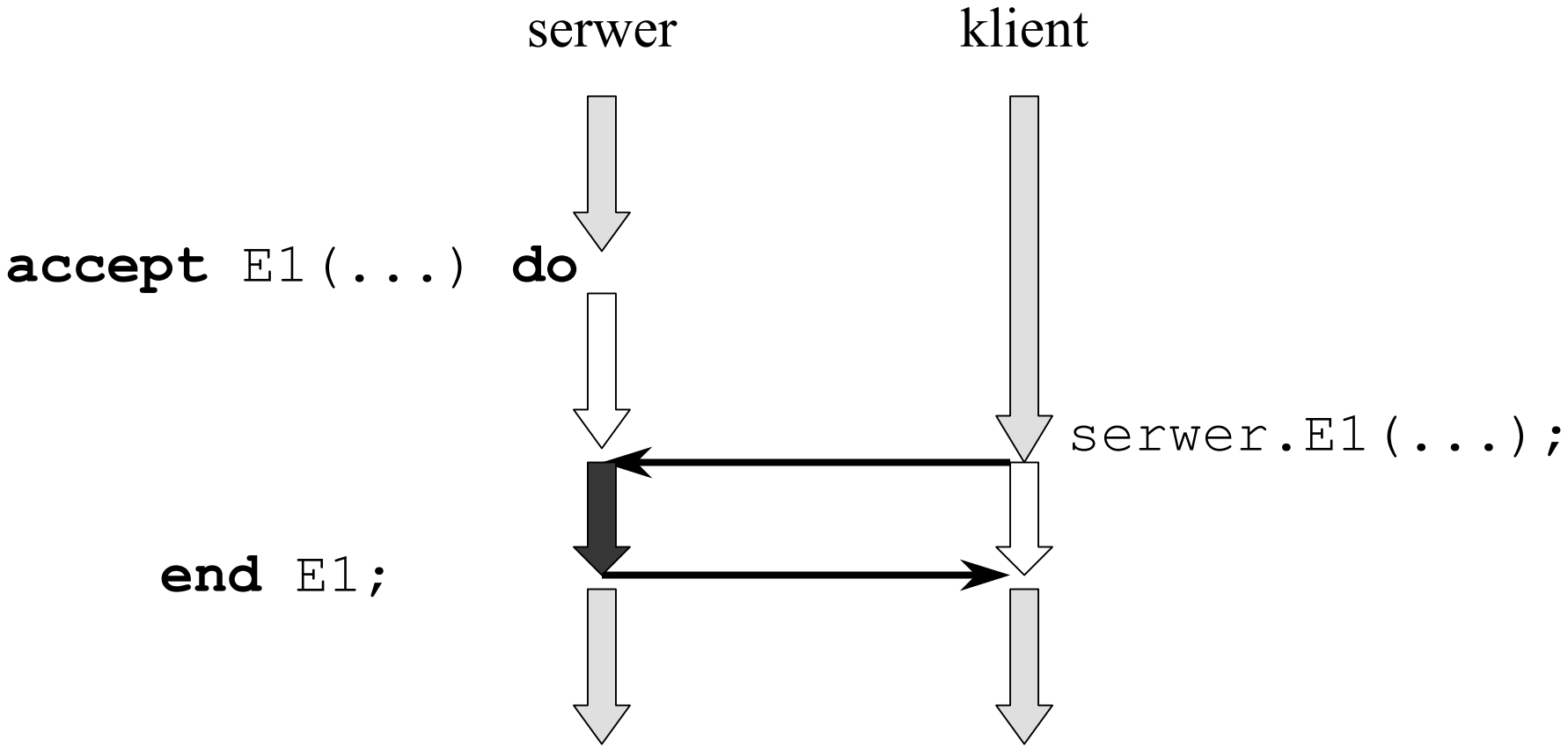
Wywołanie wejścia

```
Nazwa_zadania.wejście(param,...);
```

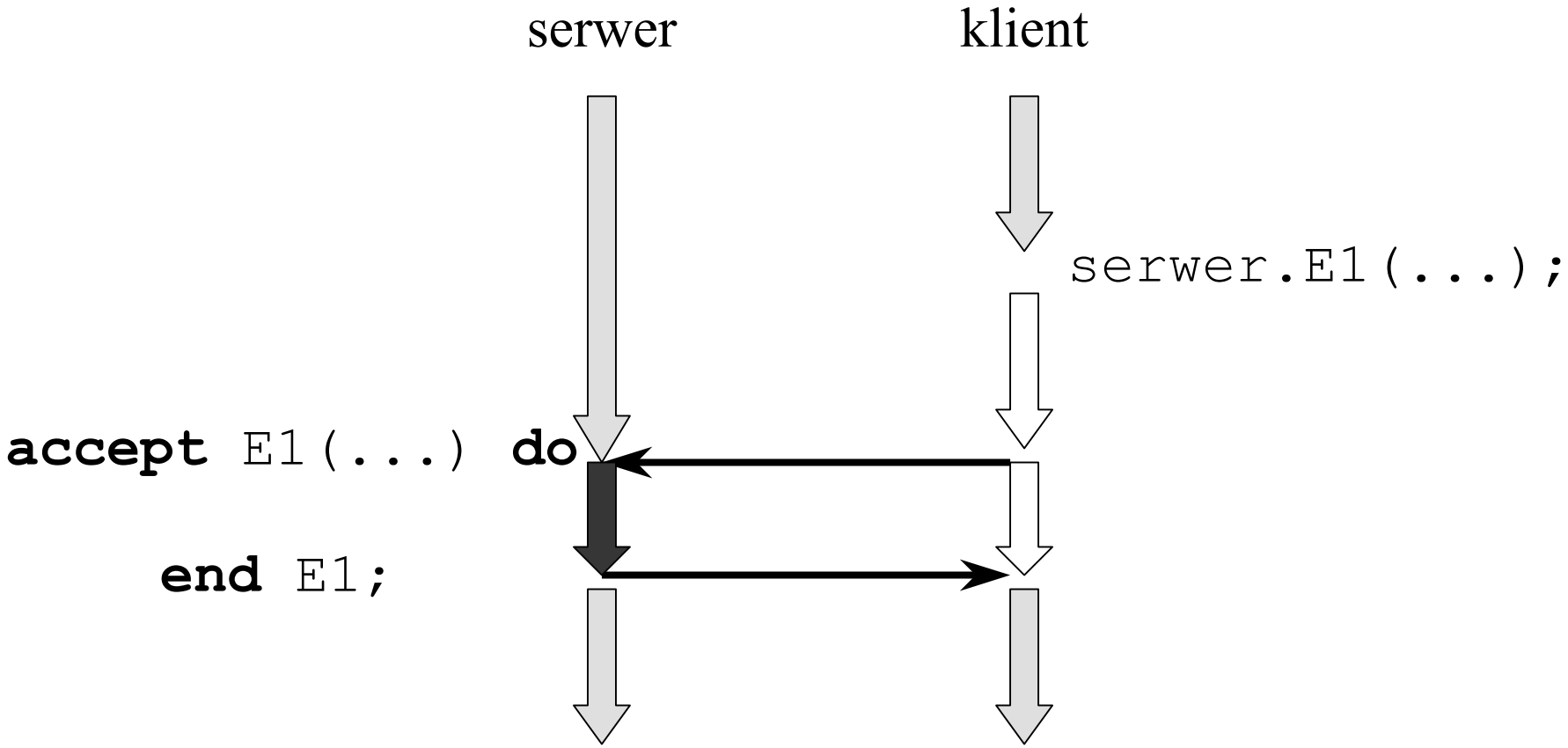
Obsługa wejścia

```
accept wejście(param: typ) do
    ...
end wejście;
```

Spotkanie — oczekiwanie serwera



Spotkanie — oczekiwanie klienta



Instrukcja `select`

- Oczekiwanie selektywne
- Terminowe wywołanie wejścia
- Warunkowe wywołanie wejścia
- Asynchroniczna zmiana wątku sterowania

Oczekiwanie selektywne

Umożliwia po **stronie serwera**:

- oczekiwanie na więcej niż jedno spotkanie — alternatywa,
- oczekiwanie na rozpoczęcie spotkania w ustalonym odcinku czasu,
- wycofanie oferty spotkania, jeżeli nie może ono nastąpić natychmiast,
- zakończenie istnienia zadania, jeżeli nie istnieją klienci, którzy wywołują jego wejścia

Alternatywa

```
task body T is
begin
  loop
    select
      accept E1(...) do
        ...
      end;
    or
      accept E2(...) do
        ...
      end;
    end select;
  end loop;
end T;
```

Alternatywa z gałęziami dozorowanymi

```
task body T is
begin
  loop
    select
      when i > 0 => accept E1(...) do
        ...
      end;
    or
      when i = 0 => accept E2(...) do
        ...
      end;
    end select;
  end loop;
end T;
```

Obsługa kolejki żądań

- Przy braku pragm przyjmowana jest strategia FIFO obsługi kolejki żądań.
- Jeśli w momencie osiągnięcia instrukcji **`select`** zadania-klienci czekają w kolejkach kilku wejść, to wybór kolejki jest **niederministyczny**.

Dozory (1)

- Każda gałąź **`accept`** instrukcji **`select`** może być poprzedzona wyrażeniem logicznym, nazywanym dozorem (guard).
- Gałęzie bez dozorów równoważne są gałęziom z dozorami `True`.
- Gałęzie, dla których dozór jest spełniony, są nazywane gałęziami otwartymi (pozostałe – zamkniętymi).

Dozory (2)

- Wykonanie instrukcji **`select`** rozpoczyna się od obliczenia dozorów wszystkich gałęzi.
- Tylko gałęzie otwarte brane są pod uwagę w trakcie dalszego wykonania instrukcji **`select`**.
- Wartości wszystkich dozorów obliczane są każdorazowo tylko raz na początku wykonania instrukcji **`select`**.
- Jeśli wszystkie gałęzie chronione są dozorami, a ich wartości są równe `False`, to generowany jest wyjątek `Program_Error`.

Przeterminowanie spotkań

```
task body T is
begin
  loop
    select
      accept E1(...) do
        ...
      end;
    or
      delay 5.0; -- możliwe również delay until
      exit; -- wyjście z pętli
    end select;
  end loop;
end T;
```

Gałąź `else`

```
task body T is
begin
  loop
    select
      accept E1(...) do
        ...
      end;
    else
      exit; -- wyjście
    end select;
  end loop;
end T;
```

Gałąź **`else`** pozwala serwerowi wycofać ofertę spotkania, gdy brak jest zadań-klientów już oczekujących na spotkanie. Gałąź **`else`** może wystąpić instrukcji **`select`** tylko raz i nie może być chroniona dozorem.

Gałąź `terminate`

```
task body T is
begin
  loop
    select
      accept E1(...) do
        ...
      end;
    or
      terminate;
    end select;
  end loop;
end T;
```

Gałąź **`terminate`** jest wybierana gdy jednostka macierzysta zadania zakończyła się i wszystkie zadania potomne albo zakończyły się, albo gotowe są wybrać gałąź **`terminate`**. Gałąź **`terminate`** może być poprzedzona dozorem. Nie może występować jednocześnie z gałęzią **`delay`** lub **`else`**.

Terminowe wywołanie wejścia

umożliwia po **stronie klienta** oczekiwanie na rozpoczęcie spotkania w ustalonym odcinku czasu.

```
select
   server.E1(...);
   -- tu mogą być jeszcze jakieś instrukcje
or
   delay 2.0; -- ewent. delay until
   -- tu mogą być jeszcze jakieś instrukcje
end select;
```

Warunkowe wywołanie wejścia

umożliwia po **stronie klienta** wycofanie oferty spotkania, jeżeli nie może ono nastąpić natychmiast.

```
select
   server.E1(...);
   --  tu mogą być jeszcze jakieś instrukcje
else
   --  tu mogą być jeszcze jakieś instrukcje
end select;
```

Asynchroniczna zmiana wątku sterowania

umożliwia przerwanie wykonywania programu po zakończeniu instrukcji wyzwalającej (po ustalonym czasie lub zakończeniu wywołania wejścia itp.).

```
select
   delay 5.0;
   Put_line (”Czas minął”);
then abort
   obliczaj(...);
end select;
```

Obiekty chronione

- Obiekt chroniony jest jednostką programową, która organizuje dostęp zadań do grupowanych przez siebie danych współdzielonych.
- Budowa obiektu chronionego jest podobna do budowy pakietu i zadania – składa się ze specyfikacji i treści implementującej obiekt.
- Możliwe jest definiowanie typów chronionych.

Struktura obiektu chronionego

- Specyfikacja obiektu (oraz typu) chronionego zawsze zawiera część publiczną i część prywatną.
- Część publiczną tworzą deklaracje funkcji, procedur oraz wejść. W części prywatnej występują deklaracje zmiennych współdzielonych i — opcjonalnie — deklaracje wewnętrznych funkcji, procedur i wejść.

Dostępu do obiektu chronionego

- Dostęp do obiektu chronionego jest możliwy tylko poprzez wywołania funkcji, procedur i wejść publicznych i odbywa się on zgodnie z zasadą wzajemnego wykluczania.
- Wywołania funkcji pozwalają tylko na odczyt danych współdzielonych (podanych w części private), a wywołania procedur i wejść – na ich modyfikowanie.

Blokady

- W momencie, gdy zadanie wywołuje wejście lub procedurę obiektu chronionego, ten może być zajęty obsługą innego wywołania (zablokowany).
- Z każdym obiektem chronionym związane są dwie blokady:
  - do czytania (shared read lock) — aktywna gdy obiekt chroniony obsługuje wywołanie swojej funkcji
  - do pisania (exclusive read/write lock) — gdy obiekt obsługuje wywołanie procedury lub wejścia.

Synchronizacja dostępu do obiektu chronionego (1)

1. Jeżeli obiekt chroniony ma założoną blokadę read i wywoływana jest jego funkcja, to funkcja ta zostaje wykonana.
2. Jeżeli obiekt chroniony ma założoną blokadę read i wywoływane jest jego wejście lub procedura, to wywołanie jest opóźniane, dopóki są zadania aktywne wewnątrz obiektu chronionego.

Synchronizacja dostępu do obiektu chronionego (2)

3. Jeżeli obiekt chroniony ma założoną blokadę read/write, to wywołanie jest opóźniane, dopóki są zadania aktywne wewnątrz obiektu chronionego.
4. Jeżeli nadchodzi kolej wykonania wywoływanego wejścia obiektu chronionego lecz bariera ma wartość False, to wywołanie jest ustawiane w kolejce związanej z tą barierą czekając na spełnienie warunku bariery; obiekt nie zostaje jeszcze zablokowany.

Przykład specyfikacji obiektu chronionego

```
protected zmienna_chroniona is
   function czytaj return zapis;
   procedure pisz (x: in zapis);
private
   element: zapis;     -- zmienna
   współdzielona
end zmienna_chroniona;
```

Przykład implementacji obiektu chronionego

```
protected body zmienna_chroniona is
   function czytaj return zapis is
   begin
      return element;
   end czytaj;
   procedure pisz (x: in zapis) is
   begin
      element:=x;
   end pisz;
end zmienna_chroniona;
```

Wejścia obiektu chronionego

- Wejścia podobnie jak procedury umożliwiają modyfikację obiektu.
- W części implementacyjnej, z każdym z zadeklarowanych wejść jest związany warunek wykonania wejścia, nazywany barierą (*barrier*).
- Treść wołanego wejścia jest wykonywana tylko jeśli warunek bariery jest spełniony.

Przykład specyfikacji typu chronionego z wejściami

```
subtype Rozmiar is Integer range 1..Rozmiar_MAX;

protected type Bufor_cykliczny(N: Rozmiar:=100) is
    entry Put(x: in Wartość);
    entry Get(x: out Wartość);
private
    bufor: array(0..N-1) of Wartość;-- bufor N-elem.
    put_ptr: Integer :=0; -- indeks miejsca wstaw.
    get_ptr: Integer :=0; -- indeks miejsca pob.
    licznik: Integer range 0..N :=0 -- zajętość buf.
end Bufor_cykliczny;
```

Rekolejkowanie

Żądanie wejścia może zostać przekazane do kojki związanej z innym wejściem (zadeklarowanym np. w części prywatnej). Instrukcja **`requeue`** może pojawić się w obsłudze wejścia zadania lub obiektu chronionego.

```
requeue E1;
```

Zadania

1. Implementacja semafora binarnego
2. Implementacja problemu ograniczonego buforowania (producenta i konsumenta) z buforem jednoelementowym
3. Implementacja problemu ograniczonego buforowania (producenta i konsumenta) z buforem wieloelementowym
4. Implementacja semafora ogólnego