Scenariusz obrony – zadanie „pierogi” (lab3, semafory)

Wstęp i cel

Są cztery bufory o długości `N` : **ciasto**, **mięso**, **ser**, **kapusta**. Czterech producentów uzupełnia odpowiednie bufory. Trzech konsumentów składa trzy typy pierogów (mięso/ser/kapusta), każdy pieróg wymaga jednocześnie 1 porcji ciasta i 1 porcji nadzienia. Celem jest zsynchronizowanie pracy semaforami tak, by nie gubić danych, nie przepełniać buforów i nie wchodzić w deadlock.

Dane wejściowe programu (przykładowe parametry)

- `N` – pojemność każdego bufora (liczba porcji).
- `P_C` , `P_M` , `P_S` , `P_K` – liczba producentów: ciasto/mięso/ser/kapusta.
- `C_M` , `C_S` , `C_K` – liczba konsumentów: pierogi z mięsem/serem/kapustą.
- Opcjonalnie czasy (ms) produkcji i konsumpcji: `t_prod_*` , `t_cons_*` .

Przykładowe uruchomienia:

```
./pierogi 10 2 2 2 2 2 2           # N=10, po 2 producentów i konsumentów
./pierogi 5 1 1 1 1 3 1 1          # mniejszy bufor, więcej konsumentów mięsnych
./pierogi 8 2 1 1 1 1 3 1 t_prod=50 t_cons=80
```

Struktura i główne elementy kodu

1. **Bufory** (tablice lub kolejki cykliczne) dla: ciasta, mięsa, sera, kapusty. Długość = `N` .
2. **Semafory per bufor**:
   - `mutex_X` (binarny) – sekcja krytyczna dostępu do bufora X.
   - `empty_X` (liczący, start `N`) – wolne miejsca w buforze X.
   - `full_X` (liczący, start `0`) – zajęte miejsca w buforze X.
3. **Wątki producentów** (4 typy), przykład dla ciasta:
   - `wait(empty_ciasto)`

- `wait(mutex_ciasto)`
- wstaw porcję do bufora ciasta
- `signal(mutex_ciasto)`
- `signal(full_ciasto)`
- (mięso/ser/kapusta: analogicznie z własnymi semaforami)

4. **Wątki konsumentów** (3 typy pierogów):
   - Pieróg z mięsem (ciasto + mięso):
     - `wait(full_ciasto)` , `wait(full_mieso)`
     - `wait(mutex_ciasto)` → zdejmij ciasto → `signal(mutex_ciasto)` → `signal(empty_ciasto)`
     - `wait(mutex_mieso)` → zdejmij mięso → `signal(mutex_mieso)` → `signal(empty_mieso)`
     - „Lepienie" (sekcja lokalna)
   - Pieróg z serem: ciasto + ser (`full_ciasto` , `full_ser` , `mutex_ciasto` , `mutex_ser`).
   - Pieróg z kapustą: ciasto + kapusta (`full_ciasto` , `full_kapusta` , `mutex_ciasto` , `mutex_kapusta`).
5. **Kolejność blokad** dla konsumenta: najpierw rezerwujemy zasoby (`full_ciasto` → `full_nadzienie`). Dopiero potem sięgamy po mutexy (np. `mutex_ciasto` → `mutex_mieso`), co zmniejsza ryzyko zakleszczeń.

Przepływ działania (krok po kroku)

1. **Start**: inicjalizacja semaforów: `empty_* = N` , `full_* = 0` , `mutex_* = 1` .
2. **Producent** (dowolny typ X):
   1. `wait(empty_X)` – czeka na wolne miejsce w buforze X.
   2. `wait(mutex_X)` – wchodzi do sekcji krytycznej bufora X.
   3. Dodaje porcję X (np. odkłada numer partii).
   4. `signal(mutex_X)` , `signal(full_X)` – zwalnia dostęp, sygnalizuje dostępność porcji.
3. **Konsument** (np. pieróg z mięsem):
   1. `wait(full_ciasto)` , `wait(full_mieso)` – rezerwuje potrzebne porcje.
   2. `wait(mutex_ciasto)` → zdejmij ciasto → `signal(mutex_ciasto)` , `signal(empty_ciasto)` .
   3. `wait(mutex_mieso)` → zdejmij mięso → `signal(mutex_mieso)` , `signal(empty_mieso)` .
   4. „Lepienie" pieroga (sekcja lokalna; można dodać `sleep` /czas symulacji).
4. **Zakończenie**: wątki kończą po wytworzeniu zadanej liczby porcji lub po sygnale stop; main czeka na wątki (`join`).

Kluczowe fragmenty (pseudokod)

Producent (dla bufora X):

```
while (run) {
    wait(empty_X);
    wait(mutex_X);
    put(X_buffer, item);
    signal(mutex_X);
    signal(full_X);
}
```

Konsument pieroga z mięsem:

```
while (run) {
    wait(full_ciasto);
    wait(full_mieso);
    wait(mutex_ciasto);
    ciasto = get(ciasto_buf);
    signal(mutex_ciasto);
    signal(empty_ciasto);

    wait(mutex_mieso);
    mieso = get(mieso_buf);
    signal(mutex_mieso);
    signal(empty_mieso);

    lep_pieroga(ciasto, mieso);
}
```

Argumentacja synchronizacji

- **Brak przepełnienia**: `empty_X` gwarantuje, że producent nie zapisze, gdy bufor pełny.
- **Brak podbioru z pustego**: `full_X` blokuje konsumenta, gdy brak porcji.
- **Spójność danych**: `mutex_X` chroni pojedynczą strukturę bufora przed równoczesnym zapisem/odczytem.
- **Unikanie deadlocków**: stała kolejność rezerwacji zasobów (ciasto → nadzienie) i szybkie zwalnianie zaraz po użyciu (jak w pseudokodzie), bez odwracania kolejności.

- **Przerwanie czekania**: zamiast sztucznego `signal(empty_*)` lepiej użyć osobnego sygnału stop (flaga + warunek/condition variable) albo timeoutów, aby wątki wybudzić bez fałszowania dostępności bufora.

Możliwe rozszerzenia testów

- Małe `N` (np. 2) + wielu konsumentów mięsa, aby sprawdzić blokowanie na `full_mieso`.
- Mało producentów ciasta vs. wielu konsumentów wszystkich typów → widać blokowanie na `full_ciasto`.
- Asymetryczne czasy produkcji/konsumpcji (np. `t_prod_mieso` duży) pokazują kumulację w innych buforach.
- Test zakończenia: licznik docelowych pierogów na konsumenta i sygnał do zakończenia producentów po spełnieniu planu.

Bezpieczne zakończenie (propozycja)

- Ustal licznik docelowych pierogów na każdy wątek konsumenta. Gdy konsument osiągnie cel, wychodzi z pętli i sygnalizuje (np. zmienną atomową lub semafor „done").
- Gdy wszystkie konsumenty zakończą, wątek główny ustawia `run = false` (flaga współdzielona/atomowa) i sygnalizuje stop przez osobny prymityw (np. condition variable lub dedykowany semafor „stop"), bez modyfikowania semaforów buforów.
- `main` wykonuje `join` na wszystkich wątkach, aby domknąć program bez osieroconych wątków.