Fabryka Czekolady – dokumentacja projektu

Autor: Krzysztof Pietrzak
Nr albumu: 156721
Temat: FABRYKA_CZEKOLADY
Repozytorium: https://github.com/Krisiekk/fabryka-czekolady

---

1. Wstęp

Projekt realizuje temat **„Fabryka czekolady" (Temat 3)**. Symulacja jest wieloprocesowa: dyrektor uruchamia magazyn, dostawców A/B/C/D oraz dwa stanowiska produkcyjne. Dane o stanie magazynu są współdzielone w pamięci dzielonej, a synchronizacja odbywa się semaforami System V. Dodatkowo użyta jest kolejka komunikatów (powiadomienia po PID) i sygnały do sterowania procesami.

---

2. Środowisko i technologie

- **Host:** macOS
- **VM:** Linux (Debian/Ubuntu) uruchomiony w VM
- **Kompilator:** g++ (C++17)
- **Build system:** CMake
- **Testy:** `tests/run_tests.sh` (bash, 7 testów automatycznych)

Skrót uruchomienia:

```
cd build
cmake .. && make
./dyrektor 100
```

Parametr `N` (1–10000) ustala pojemności magazynu i wartość `targetChocolates` w SHM. Produkcja trwa do sygnału `SIGTERM`.

---

3. Architektura systemu (zgodna z kodem)

3.1. Procesy i role

- **Dyrektor** (`src/dyrektor.cpp`): uruchamia procesy, obsługuje polecenia, wykonuje sekwencje zatrzymania, monitoruje SIGSTOP/SIGCONT magazynu.
- **Magazyn** (`src/magazyn.cpp`): właściciel IPC; tworzy/odtwarza SHM i semafory, zapisuje stan do pliku.
- **Dostawcy A/B/C/D** (`src/dostawca.cpp`): produkują składniki, zapis do ring buffera.

- **Stanowiska 1/2** (`src/stanowisko.cpp`): konsumują składniki i „produkują” czekoladę.

3.2. Mechanizmy IPC

- **Pamięć dzielona (SHM)** – parametry magazynu i segmenty danych A/B/C/D.
- **Semafory (19 sztuk)** – sterują wejściem, liczbą wolnych/zajętych miejsc oraz wskaźnikami IN/OUT.
- **Kolejka komunikatów (System V)** – powiadomienia po PID (np. „magazyn zamknięty/otwarty”).
- **Sygnały** – sterowanie procesami (SIGTERM, SIGUSR1, SIGCONT, SIGSTOP).

3.3. Pamięć dzielona i ring buffer

Magazyn przechowuje segmenty danych w jednej SHM. Dla N czekolad na stanowisko: - A: 2*N elementów po 1B - B: 2*N elementów po 1B - C: N elementów po 2B - D: N elementów po 3B

Łączny rozmiar danych = **9*N bajtów **(+ nagłówek). Wskaźniki** IN/OUT są w semaforach** jako offsety bajtowe, nie w SHM.

Przykład dla N=100: 200B (A) + 200B (B) + 200B (C) + 300B (D) = 900B danych + nagłówek (~100B).

Schemat ideowy ring buffera (1 segment):

```
segment (size = capacity * itemSize)
| 0 | 1 | 2 | 3 | ... | capacity-1 |
  ^           ^                   ^
 OUT        IN (następny zapis)  (zawija do 0)

Zapis:  IN = (IN + itemSize) % segmentSize
Odczyt: OUT = (OUT + itemSize) % segmentSize
EMPTY/FULL pilnują, żeby nie nadpisywać ani nie czytać pustych slotów.
```

3.4. Semafory

Łącznie 19 semaforów: - `SEM_MUTEX`, `SEM_RAPORT` - `SEM_EMPTY_X`, `SEM_FULL_X` dla A/B/C/D - `SEM_IN_X`, `SEM_OUT_X` dla A/B/C/D (offsety bajtowe) - `SEM_WAREHOUSE_ON` – bramka magazynu (0 = zamknięty, 1 = otwarty)

Bramka jest przechodzona atomowo (`pass_gate_intr`), aby uniknąć „zabrania” semafora w przypadku SIGSTOP.

3.5. Przepływ danych (w skrócie)

Dostawca: 1. Przejście przez bramkę `SEM_WAREHOUSE_ON`. 2. `P(EMPTY_X)` – czeka na miejsce. 3. Sekcja krytyczna (mutex) → zapis danych → aktualizacja `IN_X`. 4. `V(FULL_X)` – nowy składnik gotowy.

Stanowisko: 1. Przejście przez bramkę `SEM_WAREHOUSE_ON`. 2. `P(FULL_X)` dla wymaganych składników. 3. Sekcja krytyczna (mutex) → odczyt danych → aktualizacja `OUT_X`. 4. `V(EMPTY_X)` – zwolnienie miejsca.

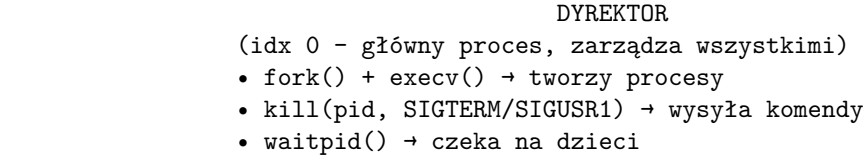
3.6. Kolejka komunikatów

Dyrektor wysyła komunikaty po PID do procesów, informując o stanie magazynu (0/1). Dostawcy i stanowiska mają wątek listenera, który odbiera te komunikaty.

3.7. Struktura repozytorium (skrót)

```
fabryka-czekolady/
  CMakeLists.txt
  docs/
     Krzysztof_Pietrzak_156721_opis_FABRYKA_CZEKOLADY.md
  include/
     common.h
  src/
     dyrektor.cpp
     magazyn.cpp
     dostawca.cpp
     stanowisko.cpp
  tests/
     run_tests.sh
  build/  (binaria)
```

3.8. Diagram architektury



```
MAGAZYN            DOSTAWCY           STANOW.
 (idx 1)           (idx 2-4)          (idx 5-6)
```

```
        PAMIĘĆ DZIELONA (964B)

    Header (80B)
     - targetChocolates, capacities, etc.


    Segment A (200B) - ring buffer

    Segment B (200B) - ring buffer

    Segment C (200B) - ring buffer

    Segment D (300B) - ring buffer
```

```
          SEMAFORY (19)
         0-1:  mutexes
         2-9:  empty/full
        10-17: in/out
        18: WAREHOUSE_ON
```

```
KOMUNIKACJA:
• Dyrektor → Procesy:   SYGNAŁY (SIGTERM, SIGUSR1)
• Procesy → Magazyn:    SEMAFORY (P/V operations)
• Procesy  Dane:        PAMIĘĆ DZIELONA (read/write)
```

---

4. Procedury i sterowanie (zgodnie z kodem)

4.1. Polecenia dyrektora

- **1** – **StopFabryka**: wysyła `SIGTERM` do stanowisk.
- **2** – **StopMagazyn**: ustawia `SEM_WAREHOUSE_ON=0` (zamknięcie bramki).

- **3 – StopDostawcy**: wysyła `SIGTERM` do dostawców.
- **4 – StopAll**: deterministyczna sekwencja zamknięcia:
  1. Zamknięcie bramki + `SIGCONT` do dzieci.
  2. `SIGTERM` do stanowisk → krótki timeout → ewentualnie `SIGKILL`.
  3. `SIGTERM` do dostawców → krótki timeout → ewentualnie `SIGKILL`.
  4. `SIGCONT` + `SIGUSR1` do magazynu → zapis stanu i zakończenie.
- **q – Quit**: kończy pętlę menu; dyrektor wysyła `SIGTERM` do wszystkich w `graceful_shutdown()`.

4.2. Obsługa SIGSTOP/SIGCONT magazynu

- **STOP**: `SEM_WAREHOUSE_ON=0`, powiadomienia do dzieci.
- **CONT**: `SEM_WAREHOUSE_ON=1`, powiadomienia do dzieci.

4.3. Zapis/odtwarzanie stanu

- Zapis następuje po `SIGUSR1` w magazynie.
- **Zapisywane:** `targetChocolates` i liczby FULL dla A/B/C/D.
- **Niezapisywane:** rzeczywiste dane w segmentach oraz liczniki produkcji stanowisk.
- Po starcie, jeśli istnieje plik stanu, magazyn **odtwarza** wartości semaforów i wypełnia segmenty symbolicznie (A/B/C/D), aby stan był spójny.

4.4. Walidacja danych wejściowych

Parametr `N` jest sprawdzany przez `strtol()` i zakres 1–10000. Błędny argument kończy program z komunikatem.

4.5. Główne funkcje w plikach źródłowych (opis działania)

`src/dyrektor.cpp`
- `start_processes()` – uruchamia magazyn, dostawców i stanowiska (fork/exec).
- `monitor_magazyn()` – wątek monitorujący `SIGSTOP/SIGCONT` magazynu; zamyka/otwiera bramkę i wysyła powiadomienia przez msq.
- `wait_for_range()` – czeka na zakończenie wybranego zakresu procesów z timeoutem (używane w StopAll).
- `graceful_shutdown()` – wysyła SIGTERM do wszystkich i domyka procesy, ewentualnie SIGKILL po timeout.
- `cleanup_old_ipcs()` – usuwa stare IPC po poprzednim uruchomieniu (defensywnie).

Pseudokod (StopAll):

```
zamknij_bramke()
SIGCONT_do_dzieci()
SIGTERM -> stanowiska; czekaj; jeśli timeout -> SIGKILL
SIGTERM -> dostawcy; czekaj; jeśli timeout -> SIGKILL
```

`SIGCONT -> magazyn; SIGUSR1 -> magazyn; czekaj; jeśli timeout -> SIGKILL`

src/magazyn.cpp
- `init_ipc()` – tworzy SHM i semafory, inicjalizuje nagłówek i wartości semaforów.
- `load_state_from_file()` – odtwarza stan FULL/EMPTY/IN/OUT z pliku stanu i uzupełnia segmenty symbolicznie.
- `save_state_to_file()` – zapisuje stan (target + FULL A/B/C/D) do pliku.
- `wait_for_shutdown()` – blokuje proces do sygnału lub zamknięcia bramki.
- `cleanup_ipc()` – odłącza SHM i usuwa IPC (gdy magazyn kończy pracę).

Pseudokod (shutdown):

```
while !stop:
  if SEM_WAREHOUSE_ON == 0: break
  pause()  // czekaj na SIGTERM/SIGUSR1
if save_on_exit: save_state_to_file()
cleanup_ipc()
```

src/dostawca.cpp
- `deliver_one()` – pojedyncza dostawa: bramka → P(EMPTY) → zapis → IN → V(FULL).
- `pass_gate_intr()` – atomowe przejście przez bramkę (blokuje gdy magazyn zamknięty).
- Listener msq (wątek) – odbiera komunikaty od dyrektora o stanie magazynu.

Pseudokod (deliver_one):

```
pass_gate_intr()
P(EMPTY_X)
P(MUTEX)
  in = SEM_IN_X
  write(segment[in])
  SEM_IN_X = (in + itemSize) % segmentSize
V(MUTEX)
V(FULL_X)
```

src/stanowisko.cpp
- `produce_one()` – kompletowanie A+B+C lub A+B+D i zwiększenie licznika produkcji.
- `consume_one()` – pobiera składnik z ring buffera: P(FULL) → odczyt → OUT → V(EMPTY).
- Listener msq (wątek) – odbiera komunikaty o stanie magazynu.

Pseudokod (produce_one):

```
pass_gate_intr()
P(FULL_A); consume_one(A)
P(FULL_B); consume_one(B)
P(FULL_C/D); consume_one(C/D)
```

produced++

---

5. Kluczowe fragmenty kodu (wybrane)

5.1 Dyrektor – sekwencja StopAll (fragment z `menu_loop()`)

Gdzie: `src/dyrektor.cpp`, funkcja `menu_loop()` (case 4).
Co robi: zamyka bramkę, zatrzymuje stanowiska i dostawców, a na końcu uruchamia zapis stanu magazynu.
Dlaczego: gwarantuje deterministyczne zakończenie bez utraty stanu.

```
// StopAll - zapis stanu
log_raport(g_semid, "DYREKTOR", "StopAll - zamykam bramkę i wznowię zatrzymane procesy...");
{
    union semun arg; arg.val = 0;
    if (g_semid != -1) semctl(g_semid, SEM_WAREHOUSE_ON, SETVAL, arg); // zamknij bramkę
    send_state_to_children(0);                                          // powiadom dzieci
    send_signal_to_all(SIGCONT);                                        // wznow, jeśli były
}

// 1) Zatrzymaj stanowiska (konsumentów)
send_signal_to_range(SIGTERM, 5, 7);
if (!wait_for_range(5, 7, 3)) {
    std::cout << "[DYREKTOR] Timeout stanowisk - SIGKILL\n";
    for (size_t j = 5; j < 7 && j < g_children.size(); ++j) {
        if (g_children[j] > 0) kill(g_children[j], SIGKILL);
    }
    wait_for_range(5, 7, 1); // ostatnia próba zebrania
}

// 2) Zatrzymaj dostawców (producentów)
log_raport(g_semid, "DYREKTOR", "StopAll - zatrzymuję dostawców...");
send_signal_to_range(SIGCONT, 1, 5);
send_signal_to_range(SIGTERM, 1, 5);
if (!wait_for_range(1, 5, 3)) {
    std::cout << "[DYREKTOR] Timeout dostawców - SIGKILL\n";
    for (size_t j = 1; j < 5 && j < g_children.size(); ++j) {
        if (g_children[j] > 0) kill(g_children[j], SIGKILL);
    }
    wait_for_range(1, 5, 1);
}

// 3) Magazyn zapisuje stan
log_raport(g_semid, "DYREKTOR", "StopAll - zapisuję stan magazynu...");
if (g_children.size() > 0 && g_children[0] > 0) {
```

```
    kill(g_children[0], SIGCONT); // wznowienie jeśli był SIGSTOP
    kill(g_children[0], SIGUSR1); // zapis + zakończenie
    if (!wait_for_range(0, 1, 3)) {
        std::cout << "[DYREKTOR] Timeout magazynu - SIGKILL\n";
        kill(g_children[0], SIGKILL);
        wait_for_range(0, 1, 1);
    }
}
```

5.2 Magazyn – init IPC (fragment z `init_ipc()`)

Gdzie: `src/magazyn.cpp`, funkcja `init_ipc()`.
Co robi: tworzy/łączy SHM i semafory oraz ustawia wartości początkowe (EMPTY/FULL/IN/OUT).
Dlaczego: po starcie system ma spójny stan magazynu i gotowe IPC.

```
key_t key = make_key();
size_t shmSize = calc_shm_size(targetChocolates);

g_shmid = shmget(key, shmSize, IPC_CREAT | IPC_EXCL | 0600);
bool fresh = true;
if (g_shmid == -1) {
    if (errno != EEXIST) die_perror("shmget");
    fresh = false;                          // segment już istnieje
    g_shmid = shmget(key, 0, 0600);         // dołącz bez rozmiaru
    if (g_shmid == -1) die_perror("shmget attach");
}

g_header = static_cast<WarehouseHeader*>(shmat(g_shmid, nullptr, 0)); // mapuj SHM
if (g_header == reinterpret_cast<void*>(-1)) die_perror("shmat");

g_semid = semget(key, SEM_COUNT, IPC_CREAT | 0600);
if (g_semid == -1) die_perror("semget");

if (fresh) {
    std::memset(g_header, 0, shmSize);                    // wyczyść SHM
    init_warehouse_header(g_header, targetChocolates);    // ustaw pojemności

    semun arg{};
    arg.val = 1; semctl(g_semid, SEM_MUTEX, SETVAL, arg);  // mutex
    arg.val = 1; semctl(g_semid, SEM_RAPORT, SETVAL, arg); // mutex dla raportu
    arg.val = g_header->capacityA; semctl(g_semid, SEM_EMPTY_A, SETVAL, arg); // EMPTY = ca
    arg.val = g_header->capacityB; semctl(g_semid, SEM_EMPTY_B, SETVAL, arg);
    arg.val = g_header->capacityC; semctl(g_semid, SEM_EMPTY_C, SETVAL, arg);
    arg.val = g_header->capacityD; semctl(g_semid, SEM_EMPTY_D, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_FULL_A, SETVAL, arg); // FULL = 0
```

```
    arg.val = 0; semctl(g_semid, SEM_FULL_B, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_FULL_C, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_FULL_D, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_IN_A, SETVAL, arg);    // IN = 0
    arg.val = 0; semctl(g_semid, SEM_IN_B, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_IN_C, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_IN_D, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_OUT_A, SETVAL, arg);  // OUT = 0
    arg.val = 0; semctl(g_semid, SEM_OUT_B, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_OUT_C, SETVAL, arg);
    arg.val = 0; semctl(g_semid, SEM_OUT_D, SETVAL, arg);
    arg.val = 1; semctl(g_semid, SEM_WAREHOUSE_ON, SETVAL, arg); // bramka otwarta
}
```

5.3 Magazyn – zapis/odczyt stanu (fragmenty)

Gdzie: `src/magazyn.cpp`, funkcje `save_state_to_file()` i `load_state_from_file()`.
Co robi: zapisuje liczby FULL i `targetChocolates` do pliku, a po restarcie odtwarza semafory.
Dlaczego: umożliwia wznowienie pracy po ponownym uruchomieniu.

```
// save_state_to_file()
int a = semctl(g_semid, SEM_FULL_A, GETVAL);
int b = semctl(g_semid, SEM_FULL_B, GETVAL);
int c = semctl(g_semid, SEM_FULL_C, GETVAL);
int d = semctl(g_semid, SEM_FULL_D, GETVAL);

int fd = open(g_stateFile.c_str(), O_WRONLY | O_CREAT | O_TRUNC, 0600);
if (fd != -1) {
    char buf[128];
    int len = snprintf(buf, sizeof(buf), "%d %d %d %d %d\n",
                       g_header->targetChocolates, a, b, c, d);
    if (len > 0) write(fd, buf, len); // zapis jednej linii stanu
    close(fd);
}
```

5.4 Dostawca – dostawa jednego składnika

Gdzie: `src/dostawca.cpp`, funkcja `deliver_one()`.
Co robi: przechodzi przez bramkę, zajmuje slot (EMPTY), zapisuje dane i zwiększa FULL.
Dlaczego: zapewnia bezpieczny zapis do ring buffera bez nadpisywania.

```
if (pass_gate_intr(g_semid, SEM_WAREHOUSE_ON) == -1) return false;

if (sem_P_intr(g_semid, semEmpty, 1) == -1) {
    if (errno == EINTR) return false;
```

```
    perror("sem_P EMPTY");
    return false;
}

P_mutex(g_semid); // chroni IN i zapis do segmentu
int inOffset = semctl(g_semid, semIn, GETVAL);
int newInOffset = (inOffset + itemSize) % segmentSize;

char *dest = segment + inOffset;
std::memset(dest, static_cast<int>(g_type), itemSize); // wpisz składnik

union semun arg;
arg.val = newInOffset;
if (semctl(g_semid, semIn, SETVAL, arg) == -1) {
    perror("semctl SETVAL IN");
    std::memset(dest, 0, itemSize);
    V_mutex(g_semid);
    sem_V_retry(g_semid, semEmpty, 1);
    return false;
}
V_mutex(g_semid);
sem_V_retry(g_semid, semFull, 1); // informacja: nowy element dostępny
```

5.5 Stanowisko – pobranie i produkcja

Gdzie: `src/stanowisko.cpp`, funkcje `consume_one()` i `produce_one()`.
Co robi: pobiera składniki A/B/C lub A/B/D, czyści sloty i zwiększa licznik produkcji.
Dlaczego: reprezentuje pełny cykl wytworzenia czekolady z magazynu.

```
bool consume_one(char type) {
    char *segment;
    int itemSize, capacity, semEmpty, semOut;
    get_segment_info(type, segment, itemSize, capacity, semEmpty, semOut);
    int segmentSize = capacity * itemSize;
    int semFull = sem_full_for(type);

P_mutex(g_semid);
int outOffset = semctl(g_semid, semOut, GETVAL);
int newOutOffset = (outOffset + itemSize) % segmentSize;
union semun arg; arg.val = newOutOffset;
semctl(g_semid, semOut, SETVAL, arg);
std::memset(segment + outOffset, 0, itemSize); // czyść zajęty slot po pobraniu
V_mutex(g_semid);

sem_V_retry(g_semid, semEmpty, 1);
```

```
return true;
}

bool produce_one() {
    if (pass_gate_intr(g_semid, SEM_WAREHOUSE_ON) == -1) return false;
    char typeC_or_D = (g_workerType == 1) ? 'C' : 'D';
    int semFullC_or_D = (g_workerType == 1) ? SEM_FULL_C : SEM_FULL_D;

    if (sem_P_intr(g_semid, SEM_FULL_A, 1) == -1) return false;
    if (!consume_one('A')) return false; // pobierz A
    if (sem_P_intr(g_semid, SEM_FULL_B, 1) == -1) return false;
    if (!consume_one('B')) return false; // pobierz B
    if (sem_P_intr(g_semid, semFullC_or_D, 1) == -1) return false;
    if (!consume_one(typeC_or_D)) return false; // pobierz C lub D

    g_produced++; // zrobiona jedna czekolada
    return true;
}
```

---

6. Problemy napotkane i rozwiązania

- **Stare IPC po awarii** → przy ponownym starcie semafory miały złe wartości; dodano `cleanup_old_ipcs()` w dyrektorze, który usuwa stare zasoby przed uruchomieniem nowej instancji.
- **Ryzyko blokady bramki przy SIGSTOP** → proces mógł „zabrać" `SEM_WAREHOUSE_ON` i nie oddać; zastąpiono sekwencję P/V atomowym `pass_gate_intr()` (jedno `semop`).
- **Brak reakcji na SIGSTOP/SIGCONT magazynu** → po zatrzymaniu magazynu procesy dalej próbowały pracować; dodano wątek monitorujący w dyrektorze i powiadomienia przez msq (zamknięcie/otwarcie bramki).
- **Race condition przy StopAll** → magazyn dostawał SIGTERM zanim obsłużył SIGUSR1; dodano sekwencję z `wait_for_range()` po SIGUSR1.
- **Potencjalny deadlock mutexu przy awarii procesu** → jeśli proces padał w sekcji krytycznej, mutex zostawał zablokowany; ustawiono `SEM_UNDO` na mutexach.

---

7. Testy

Testy automatyczne są w `tests/run_tests.sh` (łącznie 7). Poniżej opis każdego testu i logi wklejone 1:1.

TEST 1: Poprawne uruchomienie wszystkich procesów

Co sprawdzamy: start dyrektora, magazynu, 4 dostawców i 2 stanowisk oraz zapis stanu po StopAll.
Parametry: N=100, timeout 10s, komenda „4”.

Logi:

```
[20:27:54] MAGAZYN: Start magazynu (target=100 czekolad, pamięć=964 bajtów, A=200 B=200 C=10
[20:27:55] DYREKTOR: StopAll - zatrzymuję stanowiska...
[20:27:55] DOSTAWCA: Dostarczono 1 x C (offset=0, elem=0/100)
[20:27:55] DOSTAWCA: Dostarczono 1 x A (offset=0, elem=0/200)
[20:27:55] DOSTAWCA: Dostarczono 1 x D (offset=0, elem=0/100)
[20:27:55] DOSTAWCA: Dostarczono 1 x B (offset=0, elem=0/200)
[20:27:55] DYREKTOR: StopAll - zatrzymuję dostawców...
[20:27:55] DOSTAWCA: Dostawca A kończy pracę
[20:27:55] DOSTAWCA: Dostawca B kończy pracę
[20:27:55] DOSTAWCA: Dostawca C kończy pracę
[20:27:55] DOSTAWCA: Dostawca D kończy pracę
[20:27:56] DYREKTOR: StopAll - zapisuję stan magazynu...
[20:27:56] MAGAZYN: Magazyn zamknięty (SEM_WAREHOUSE_ON=0)
[20:27:56] MAGAZYN: Zapisuje stan: A=1 B=1 C=1 D=1
```

TEST 2: Mała pojemność magazynu (capacity=5)

Co sprawdzamy: stabilność przy małych buforach i poprawność FULL/EMPTY.
Parametry: N=5, timeout 15s, komenda „4”.

Logi:

```
[20:27:57] MAGAZYN: Start magazynu (target=5 czekolad, pamięć=109 bajtów, A=10 B=10 C=5 D=5
[20:27:58] DYREKTOR: StopAll - zatrzymuję stanowiska...
[20:27:58] DOSTAWCA: Dostarczono 1 x C (offset=0, elem=0/5)
[20:27:58] DOSTAWCA: Dostarczono 1 x B (offset=0, elem=0/10)
[20:27:58] DOSTAWCA: Dostarczono 1 x A (offset=0, elem=0/10)
[20:27:58] DOSTAWCA: Dostarczono 1 x D (offset=0, elem=0/5)
[20:27:58] DYREKTOR: StopAll - zatrzymuję dostawców...
[20:27:58] DOSTAWCA: Dostawca D kończy pracę
[20:27:58] DOSTAWCA: Dostawca A kończy pracę
[20:27:58] DOSTAWCA: Dostawca C kończy pracę
[20:27:58] DOSTAWCA: Dostawca B kończy pracę
[20:27:59] DYREKTOR: StopAll - zapisuję stan magazynu...
[20:27:59] MAGAZYN: Magazyn zamknięty (SEM_WAREHOUSE_ON=0)
[20:27:59] MAGAZYN: Zapisuje stan: A=1 B=1 C=1 D=1
```

TEST 3: Zatrzymanie dostawców (StopDostawcy)

Co sprawdzamy: komenda 3 zatrzymuje dostawców bez ubicia stanowisk.
Parametry: N=100, „3” po 4s, „4” po kolejnych 6s.

Logi:

```
[20:28:03] DOSTAWCA: Dostarczono 1 x C (offset=2, elem=1/100)
[20:28:03] DOSTAWCA: Dostarczono 1 x D (offset=3, elem=1/100)
[20:28:03] DOSTAWCA: Dostarczono 1 x B (offset=2, elem=2/200)
[20:28:04] DYREKTOR: Wysyłam SIGTERM do dostawców
[20:28:04] DOSTAWCA: Dostawca A kończy pracę
[20:28:04] DOSTAWCA: Dostawca B kończy pracę
[20:28:04] DOSTAWCA: Dostawca C kończy pracę
[20:28:04] DOSTAWCA: Dostawca D kończy pracę
[20:28:10] DYREKTOR: StopAll - zatrzymuję stanowiska...
[20:28:10] STANOWISKO: Stanowisko 1 kończy pracę (wyprodukowano 1 czekolad)
[20:28:10] STANOWISKO: Stanowisko 2 kończy pracę (wyprodukowano 1 czekolad)
[20:28:11] DYREKTOR: StopAll - zatrzymuję dostawców...
[20:28:11] DYREKTOR: StopAll - zapisuję stan magazynu...
[20:28:11] MAGAZYN: Magazyn zamknięty (SEM_WAREHOUSE_ON=0)
[20:28:11] MAGAZYN: Zapisuje stan: A=0 B=1 C=1 D=1
```

TEST 4: Wieloprocesowość i synchronizacja

Co sprawdzamy: praca równoległa 7 procesów i brak race conditions.
Parametry: N=100, dłuższy przebieg (8s), komenda „4”.

Logi:

```
[20:28:19] DOSTAWCA: Dostarczono 1 x D (offset=12, elem=4/100)
[20:28:20] DYREKTOR: StopAll - zatrzymuję stanowiska...
[20:28:20] STANOWISKO: Stanowisko 2 kończy pracę (wyprodukowano 2 czekolad)
[20:28:20] STANOWISKO: Stanowisko 1 kończy pracę (wyprodukowano 2 czekolad)
[20:28:20] DOSTAWCA: Dostarczono 1 x A (offset=4, elem=4/200)
[20:28:20] DOSTAWCA: Dostarczono 1 x D (offset=15, elem=5/100)
[20:28:20] DOSTAWCA: Dostarczono 1 x C (offset=10, elem=5/100)
[20:28:21] DYREKTOR: StopAll - zatrzymuję dostawców...
[20:28:21] DOSTAWCA: Dostawca D kończy pracę
[20:28:21] DOSTAWCA: Dostawca B kończy pracę
[20:28:21] DOSTAWCA: Dostawca A kończy pracę
[20:28:21] DOSTAWCA: Dostawca C kończy pracę
[20:28:21] DYREKTOR: StopAll - zapisuję stan magazynu...
[20:28:21] MAGAZYN: Magazyn zamknięty (SEM_WAREHOUSE_ON=0)
[20:28:21] MAGAZYN: Zapisuje stan: A=1 B=0 C=4 D=4
```

TEST 5: Test zakleszczeń (długi przebieg)

Co sprawdzamy: brak deadlocków w długim działaniu.
Parametry: N=10, timeout 30s, komenda „4”.

Logi:

```
[20:28:23] MAGAZYN: Start magazynu (target=10 czekolad, pamięć=154 bajtów, A=20 B=20 C=10 D=
```

```
[20:28:24] DYREKTOR: StopAll - zatrzymuję stanowiska...
[20:28:24] DOSTAWCA: Dostarczono 1 x B (offset=0, elem=0/20)
[20:28:24] DOSTAWCA: Dostarczono 1 x A (offset=0, elem=0/20)
[20:28:24] DOSTAWCA: Dostarczono 1 x C (offset=0, elem=0/10)
[20:28:24] DOSTAWCA: Dostarczono 1 x D (offset=0, elem=0/10)
[20:28:24] DYREKTOR: StopAll - zatrzymuję dostawców...
[20:28:24] DOSTAWCA: Dostawca B kończy pracę
[20:28:24] DOSTAWCA: Dostawca A kończy pracę
[20:28:24] DOSTAWCA: Dostawca C kończy pracę
[20:28:24] DOSTAWCA: Dostawca D kończy pracę
[20:28:25] DYREKTOR: StopAll - zapisuję stan magazynu...
[20:28:25] MAGAZYN: Magazyn zamknięty (SEM_WAREHOUSE_ON=0)
[20:28:25] MAGAZYN: Zapisuje stan: A=1 B=1 C=1 D=1
```

TEST 6: StopMagazyn – brak zapisu stanu

Co sprawdzamy: komenda 2 nie tworzy `magazyn_state.txt`.
Parametry: N=100, „2” po 3s, „q” po kolejnych 2s.

Logi:

```
[20:28:26] MAGAZYN: Start magazynu (target=100 czekolad, pamięć=964 bajtów, A=200 B=200 C=10
[20:28:27] DOSTAWCA: Dostarczono 1 x B (offset=0, elem=0/200)
[20:28:27] DOSTAWCA: Dostarczono 1 x C (offset=0, elem=0/100)
[20:28:27] DOSTAWCA: Dostarczono 1 x A (offset=0, elem=0/200)
[20:28:27] DOSTAWCA: Dostarczono 1 x D (offset=0, elem=0/100)
[20:28:27] STANOWISKO: Stanowisko 2 wyprodukowano czekoladę #1 (A+B+D)
[20:28:28] DOSTAWCA: Dostarczono 1 x C (offset=2, elem=1/100)
[20:28:28] DOSTAWCA: Dostarczono 1 x D (offset=3, elem=1/100)
[20:28:29] DYREKTOR: Wysyłam SIGTERM do magazynu (bez zapisu)
[20:28:29] MAGAZYN: Magazyn zamknięty (SEM_WAREHOUSE_ON=0)
[20:28:29] MAGAZYN: Zakończenie bez zapisu stanu (SIGTERM)
```

TEST 7: Resume – zapis stanu, restart, wczytanie

Co sprawdzamy: poprawny zapis i odtworzenie stanu po restarcie.
Parametry: N=10, dwa uruchomienia (Run#1 zapis, Run#2 odczyt).

Logi:

```
[20:28:35] MAGAZYN: Wczytano stan z pliku (A=1, B=1, C=1, D=1)
[20:28:35] MAGAZYN: Odtworzono stan: A=1 B=1 C=1 D=1
[20:28:36] DYREKTOR: StopAll - zatrzymuję stanowiska...
[20:28:36] DOSTAWCA: Dostarczono 1 x C (offset=2, elem=1/10)
[20:28:36] DOSTAWCA: Dostarczono 1 x A (offset=1, elem=1/20)
[20:28:36] DOSTAWCA: Dostarczono 1 x B (offset=1, elem=1/20)
[20:28:36] DOSTAWCA: Dostarczono 1 x D (offset=3, elem=1/10)
[20:28:37] DYREKTOR: StopAll - zatrzymuję dostawców...
```

```
[20:28:37] DOSTAWCA: Dostawca B kończy pracę
[20:28:37] DOSTAWCA: Dostawca C kończy pracę
[20:28:37] DOSTAWCA: Dostawca A kończy pracę
[20:28:37] DOSTAWCA: Dostawca D kończy pracę
[20:28:37] DYREKTOR: StopAll - zapisuję stan magazynu...
[20:28:37] MAGAZYN: Magazyn zamknięty (SEM_WAREHOUSE_ON=0)
[20:28:37] MAGAZYN: Zapisuje stan: A=2 B=2 C=2 D=2
```

---

8. Wymagania projektu – co zostało użyte

Zrealizowane konstrukcje i mechanizmy: - Procesy: `fork()`, `execv()`, `waitpid()`. - Sygnały: `SIGTERM`, `SIGUSR1`, `SIGCONT`, `SIGSTOP`. - Semafory System V: `ftok()`, `semget()`, `semctl()`, `semop()`. - Pamięć dzielona: `shmget()`, `shmat()`, `shmdt()`, `shmctl()`. - Kolejka komunikatów: `msgget()`, `msgsnd()`, `msgrcv()`, `msgctl()`. - Pliki: `open()`, `read()`, `write()`, `close()`. - Wątki (monitor/odbiornik msq): `std::thread` (backend pthread). - Walidacja wejścia: `strtol()` + zakres. - Minimalne prawa dostępu: 0600 dla IPC i plików stanu (raport także 0600).

Nie użyto (nie były wymagane w tym temacie): - Sockets, pipes/FIFO, `dup/dup2`, mechanizmy gniazd.

---

9. Linki do repozytorium (wymagane fragmenty kodu)

Poniżej linki do kluczowych plików i konkretnych fragmentów **w commicie**

```
Repozytorium: https://github.com/Krisiekk/fabryka-czekolady
Procesy (fork / exec / wait):
- spawn (fork+exec):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- start_processes (uruchamianie procesów):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- monitor_magazyn (waitpid STOP/CONT):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/

Semafory (semget / semctl / semop):
- definicje i helpery (`sem_P_intr`, `sem_V_retry`):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- bramka atomowa (`pass_gate_intr`):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- inicjalizacja semaforów (magazyn):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/

Pamięć dzielona (shmget / shmat / shmdt / shmctl):
```

- tworzenie i mapowanie SHM (magazyn):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- dołączenie i shmat w dostawca:
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- dołączenie i shmat w stanowisko:
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/

Kolejka komunikatów (msgget / msgsnd / msgrcv / msgctl):
- helpery msq (msq_send_pid / msq_recv_pid_intr):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- wysyłanie stanu z dyrektora:
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- nasłuch w dostawca (msgrcv loop):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- nasłuch w stanowisko (msgrcv loop):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/

Sygnały (sigaction / kill / prctl):
- setup handlerów (setup_sigaction):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- parent-death (prctl) w magazyn:
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- parent-death (prctl) w dostawca:
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/
- monitor SIGSTOP/SIGCONT (dyrektor):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/

Pliki (open / read / write / close):
- zapis stanu (magazyn):
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/

Testy:
- skrypt testowy:
https://github.com/Krisiekk/fabryka-czekolady/blob/e86b2b08dd266b5a2d6a3eaee5aeb5627eb37a0e/