

# Ciastkarnia – Opis Projektu

Systemy Operacyjne – Temat 15

Tomasz Pulka

## 1. Opis zadania

Symulacja ciastkarni z samoobsługowym sklepem firmowym. Ciastkarnia produkuje  $P$  różnych produktów ( $P > 10$ ), każdy w innej cenie. Produkty po wypieku trafiają na oddzielne podajniki (FIFO o pojemności  $K_i$ ), a klienci pobierają je w sklepie i opłacą przy kasach.

Symulacja działa na **osobnych procesach** (`fork` + `exec`) komunikujących się przez mechanizmy IPC Systemu V (pamięć dzielona, semaforey, kolejki komunikatów) oraz łącza (pipe, FIFO).

## 2. Procesy i wątki

### Kierownik (`kierownik.c`) – główny proces

- Tworzy wszystkie zasoby IPC (`shm`, semaforey, 3 kolejki, pipe, FIFO).
- Uruchamia dzieci: `fork()` + `execl()` – piekarz (1), kasjerzy (2), klienci ( $N$ ).
- Prowadzi **zegar symulacji** (każda iteracja petli = 1 minuta symulacyjna).
- Co 1-10 minut generuje batch 2-8 nowych klientów.
- Otwiera/zamyka drugą kasę w zależności od liczby klientów.
- Nasłuchuje poleceń z FIFO (inventaryzacja, ewakuacja).
- Na koniec generuje raport i sprząta wszystkie zasoby.

Tworzenie dzieci:

```
fork() -> execl("./piekarz", key_file, ...) -> piekarz
fork() -> execl("./kasjer", key_file, "0", ...) -> kasjer 0
fork() -> execl("./kasjer", key_file, "1", ...) -> kasjer 1
fork() -> execl("./klient", key_file, ...) -> klient N
```

Każde dziecko dostaje ścieżkę do pliku klucza (`ciastkarnia.key`) jako argument i używa `ftok()` do podłączenia się do tych samych zasobów IPC.

### Piekarz (`piekarz.c`)

- 2 wątki produkcyjne (`pthread_create`): wątek 0 – produkty 0-5, wątek 1 – 6-11.
- Petla: losowa partia – `sem_trywait(podajnik)` – `msgsnd()` do kolejki podajników.
- Raportuje produkcję do kierownika przez **pipe** (`write()`).
- Wspólny licznik chroniony `pthread_mutex_t`.

### Kasjer (`kasjer.c`)

- 2 instancje (kasa 0, kasa 1). Każda ma wątek monitora (`pthread_create`, `detached`).

- Monitor co 500ms sprawdza stan kasy – `pthread_cond_signal()` budzi glowny watek.
- Glowna petla: `msgrcv()` z kolejki checkout – skanuje produkty – aktualizuje SHM – `msgsnd()` paragon.
- Dane chronione `pthread_mutex_t` + `pthread_cond_t`.

### Klient (`klient.c`)

1. `sem_trywait(SEM_SHOP_ENTRY)` z `SEM_UNDO` – wejscie do sklepu (maks. N osob).
2. Losuje liste zakupow (2-5 produktow, 1-3 szt. kazdego).
3. `msgrcv()` z kolejki podajnikow (`mtype = product_id + 1`) – pobiera ciastka.
4. Wybiera kase z krotsza kolejka – `msgsnd()` koszyk do checkout.
5. `msgrcv()` paragon (`mtype = getpid()`) – czeka na swoj paragon.
6. `sem_signal(SEM_SHOP_ENTRY)` z `SEM_UNDO` – zwalnia miejsce.
7. Ewakuacja: odklada produkty do kosza w SHM i natychmiast wychodzi.

### Schemat procesow i watkow

kierownik (PID glowny)

```
|
+-- fork+exec -> piekarz
|               +-- pthread -> watek produkcji 0 (produkty 0-5)
|               +-- pthread -> watek produkcji 1 (produkty 6-11)
|
+-- fork+exec -> kasjer 0
|               +-- pthread -> watek monitora (detached)
|
+-- fork+exec -> kasjer 1
|               +-- pthread -> watek monitora (detached)
|
+-- fork+exec -> klient 1
+-- fork+exec -> klient 2
|   ...
+-- fork+exec -> klient N
```

## 3. Mechanizmy IPC

### a) Pamiec dzielona (System V Shared Memory)

Jeden segment (`ftok` z identyfikatorem 'S'), uprawnienia 0660. Zawiera strukture `SharedData` z calym stanem symulacji:

- Konfiguracja: ile produktow, max klientow, skala czasu, godziny otwarcia/zamknienia
- Stan sklepu: klientow w srodku, kasy otwarte, dlugosci kolejek
- Statystyki: sprzedaz na kazdej kasie, produkcja
- Flagi: `simulation_running`, `shop_open`, `evacuation_mode`
- Zegar: `sim_hour`, `sim_min`
- PIDy procesow (piekarz, kasjery)

Wywolania: `shmget()`, `shmat()`, `shmdt()`, `shmctl(IPC_RMID)`.

## b) Semafor (System V)

Jeden zbior semaforow (ftok z 'E'), uprawnienia 0660:

| Indeks | Nazwa               | Init  | Typ        | Zastosowanie                               |
|--------|---------------------|-------|------------|--|
| 0      | SEM_SHM_MUTEX       | 1     | Binarny    | Ochrona dostępu do pamięci dzielonej       |
| 1      | SEM_SHOP_ENTRY      | N     | Zliczający | Kontrola maks. N klientów w sklepie        |
| 2..P+1 | SEM_CONVEYOR_BASE+i | Ki    | Zliczający | Wolne miejsca na podajniku i-tego produktu |
| P+2    | SEM_GUARD_CONVEYOR  | limit | Zliczający | Backpressure kolejki podajników            |
| P+3    | SEM_GUARD_CHECKOUT  | limit | Zliczający | Backpressure kolejki checkout              |
| P+4    | SEM_GUARD_RECEIPT   | limit | Zliczający | Backpressure kolejki paragonów             |

Kluczowe: mutex i shop\_entry używają **SEM\_UNDO** – automatycznie zwalnianie semafor jeśli proces zostanie zabity (kill -9). Zapobiega to trwałemu deadlockowi.

Wywołania: semget(), semctl(SETVAL), semctl(GETVAL), semop(), semctl(IPC\_RMID).

## c) Kolejki komunikatów (System V)

3 kolejki (ftok z 'C', 'K', 'R'), uprawnienia 0660:

| Kolejka   | Kierunek          | mtype           | Treść                 |
|-----------|-------------------|-----------------|-----------------------|
| Podajniki | piekarz -> klient | product_id + 1  | ConveyorMsg (produkt) |
| Checkout  | klient -> kasjer  | register_id + 1 | CheckoutMsg (koszyk)  |
| Paragony  | kasjer -> klient  | customer_pid    | ReceiptMsg (paragon)  |

Filtrowanie msgrcv() przez mtype: klient pobiera konkretne ciastko, kasjer obsługuje swoją kasę, klient czeka na swój paragon po PID.

**Guard semaphores:** każda kolejka ma semafor zliczający inicjalizowany na msg\_qbytes / sizeof(msg). Przed msgsnd() – sem\_wait(guard), po msgrcv() – sem\_signal(guard). Zapobiega to przepelnieniu kolejki i zablokowaniu msgsnd.

Wywołania: msgget(), msgsnd(), msgrcv(), msgctl(IPC\_RMID), msgctl(IPC\_STAT).

## d) Łącze nienazwane (pipe)

```
pipe(fd[2]) -> fd[0] = odczyt, fd[1] = zapis
```

Piekarz -> Kierownik: raporty produkcji w formacie "BATCH:tid:count\n". Pipe tworzony przed fork(), więc obaj mają te same deskryptory.

Wywołania: pipe(), read(), write(), close().

## e) Łącze nazwane (FIFO)

```
mkfifo("/tmp/ciastkarnia_cmd.fifo", 0660)
```

Użytkownik -> Kierownik: polecenia tekstowe inwentaryzacja lub ewakuacja. Kierownik otwiera FIFO jako O\_RDONLY | O\_NONBLOCK i czyta w każdej iteracji petli.

Wywołania: mkfifo(), open(), read(), close(), unlink().

## f) Pliki

- `creat("ciastkarnia.key")` – plik klucza dla `ftok()`
- `open("logs/raport.txt"), write(), close()` – raport końcowy
- `dup2()` – przekierowanie `stderr` dzieci do plików logów
- `popen("date ...")` – pobranie daty systemowej do raportu
- `mkdir("logs")` – tworzenie katalogu logów
- `unlink()` – usuwanie pliku klucza i FIFO przy czyszczeniu

## 4. Sygnały

| Sygnał  | Kto wysyła       | Do kogo                   | Co robi   |
|---------|------------------|---------------------------|---|
| SIGCHLD | kernel           | kierownik                 | Dziecko zakończyło – <code>waitpid(WNOHANG)</code> zbiera |
| SIGINT  | Ctrl+C /<br>test | kierownik                 | Czyste zamknięcie z raportem                              |
| SIGTERM | kierownik        | piekarz, kasjery, klienci | Zakończenie procesu                                       |
| SIGUSR1 | kierownik        | dzieci                    | Inwentaryzacja (sygnał1)                                  |
| SIGUSR2 | kierownik        | dzieci                    | Ewakuacja – natychmiast wyjdź (sygnał2)                   |

Handlersy ustawiane przez `sigaction()` z flagami `SA_RESTART` i `SA_NOCLDSTOP`.

## 5. Synchronizacja – problemy i rozwiązania

## Wyciąg przy dostępie do SHM

`SEM_SHM_MUTEX` (semafor binarny) z `SEM_UNDO`. Każdy dostęp do `SharedData` owinięty w `sem_wait_undo/sem_signal_`

## Za dużo klientów w sklepie

`SEM_SHOP_ENTRY` (semafor zliczający, `init = N`) z `SEM_UNDO`. Klient dekrementuje przy wejściu, inkrementuje przy wyjściu. Slot zwalniany jeśli klient zginie.

## Podajnik pełny

`SEM_CONVEYOR_BASE+i` (`init = Ki`). Piekarz robi `sem_trywait` (nieblokujący) – jeśli 0, podajnik pełny, pomija produkt.

## Zombie procesy

Handler `SIGCHLD` + `waitpid(WNOHANG)` w pętli głównej. Rozróżnia śmierć klienta od śmierci piekarza/kasjera (nie dekrementuje `active_customers` dla nie-klientów).

## Wyciek IPC po crashu

`atexit(atexit_cleanup)` – sprząta IPC nawet przy niespodziewanym `exit()`. Przy starcie `cleanup_all_ipc()` usunąć stale zasoby z poprzedniego uruchomienia.

## Przyspieszenie zegara przez SIGCHLD

`msleep_safe()` – używa `nanosleep()` z pętlą retry na `EINTR`. Zapobiega skróceniu snu przez sygnał.

## 6. Zarządzanie kasami

Kierownik co iterację sprawdza liczbę klientów:

- $\geq N/2$  klientów -> otwiera kasę 1 (`register_open[1] = 1`)
- $< N/2$  klientów -> kasa 1 przestaje przyjmować (dokończy kolejke)
- kolejka kasy 1 = 0 -> kasa 1 zamknięta

Kasa 0 jest **zawsze otwarta**.

## 7. Zamykanie symulacji

Trzy sposoby zamknięcia:

1. **Normalne** – zegar dochodzi do godziny zamknięcia ( $T_k$ )
2. **Ctrl+C / SIGINT** – czyste zamknięcie z raportem
3. **Ewakuacja** – przez FIFO, natychmiastowe opuszczenie

Procedura `shutdown_simulation()`:

1. `simulation_running = 0, shop_open = 0, bakery_open = 0`
2. Czekaj max 5s aż klienci wyjdą sami
3. SIGTERM do piekarza, kasjerów, klientów
4. `waitpid()` zbiera procesy (z timeoutem 5s)
5. Jeśli ktoś nie odpowiedział -> SIGKILL jako ostateczność
6. Zbierz ostatnie zombie
7. Generuj raport
8. Usun wszystkie IPC (`cleanup_all_ipc()`)

## 8. Testy

Automatyczne (`make test`)

| #  | Skrypt                                  | Opis  |
|----|---|---|
| 01 | <code>test_01_startup_cleanup.sh</code> | Start -> procesy widoczne -> IPC istnieje -> zamknięcie -> IPC czyste |
| 02 | <code>test_02_no_zombies.sh</code>      | Monitorowanie zombie co 0.5s -> 0 zombie                              |
| 03 | <code>test_03_stress_capacity.sh</code> | $N=4$ , szybki czas -> <code>customers_in_shop</code> nigdy $> N$     |
| 04 | <code>test_04_evacuation.sh</code>      | Ewakuacja przez FIFO -> raport z ewakuacją                            |
| 05 | <code>test_05_sigint_cleanup.sh</code>  | SIGINT -> raport -> 0 zombie -> IPC czyste                            |

Dodatkowy: `test_kill.sh` – `kill` i `kill -9` na procesach -> symulacja kontynuuje dzięki `SEM_UNDO`.

## 9. Obsługa błędów

Dedykowany moduł `error_handler.c` z funkcjami:

- `handle_error(msg)` – `perror()` + wypisanie `errno` + `exit(EXIT_FAILURE)` – błąd krytyczny
- `handle_warning(msg)` – `perror()` bez przerywania – ostrzeżenie
- `check_sys_call(ret, msg, fatal)` – wrapper na sprawdzanie wartości zwracanej
- `validate_int_range(val, min, max, name)` – walidacja parametrów z czytelnym komunikatem

Kazde wywołanie systemowe (`fork`, `shmget`, `semget`, `msgget`, `pipe`, `mkfifo`, `open`, `execl`, `pthread_create`) jest sprawdzane. Błędy `semop` z `EINTR` są powtarzane, z `EIDRM/EINVAL` – ignorowane (shutdown).

## 10. Funkcje wymagane przez projekt (gdzie szukać)

- **Pliki:** `creat()`, `open()`, `close()`, `read()`, `write()`, `unlink()` – `kierownik.c`, `ipc_utils.c`
- **Procesy:** `fork()`, `execl()`, `exit()`, `waitpid()` – `kierownik.c`
- **Watki:** `pthread_create()`, `pthread_join()`, `pthread_detach()`, `pthread_mutex_lock/unlock()`, `pthread_cond_wait/signal/broadcast()` – `piekarz.c`, `kasjer.c`
- **Sygnały:** `kill()`, `sigaction()` – `kierownik.c`, `piekarz.c`, `kasjer.c`, `klient.c`
- **Semaforey:** `ftok()`, `semget()`, `semctl()`, `semop()` – `ipc_utils.c`
- **Lacza:** `mkfifo()`, `pipe()`, `dup2()`, `popen()` – `ipc_utils.c`, `kierownik.c`
- **Pamięć dzielona:** `ftok()`, `shmget()`, `shmat()`, `shmdt()`, `shmctl()` – `ipc_utils.c`
- **Kolejki komunikatów:** `ftok()`, `msgget()`, `msgsnd()`, `msgrcv()`, `msgctl()` – `ipc_utils.c`

## 11. Struktura projektu

```
os-bakery-simulator/
+-- Makefile
+-- README.md
+-- docs/
|   +-- opis_projektu.md      <- ten plik (zrodlo dla PDF)
+-- src/
|   +-- common.h              Stale, struktury, definicje IPC
|   +-- error_handler.h/.c    Obsluga bledow (perror, walidacja)
|   +-- ipc_utils.h/.c        Narzedzia IPC (shm, sem, msg, pipe, fifo)
|   +-- logger.h/.c           Kolorowe logowanie z zegarem
|   +-- kierownik.c           Glowny proces (manager)
|   +-- piekarz.c             Piekarz (2 watki produkcyjne)
|   +-- kasjer.c              Kasjer (2 instancje, watek monitora)
|   +-- klient.c              Klient (zakupy, kasa, wyjscie)
|   +-- check_shm.c           Narzedzie diagnostyczne SHM
+-- tests/
|   +-- run_tests.sh          Runner testow
|   +-- test_01_startup_cleanup.sh
|   +-- test_02_no_zombies.sh
|   +-- test_03_stress_capacity.sh
|   +-- test_04_evacuation.sh
|   +-- test_05_sigint_cleanup.sh
|   +-- test_kill.sh          Test odpornosci na kill
+-- logs/                     Generowany automatycznie
    +-- raport.txt
    +-- piekarz.log
    +-- kasjer_0.log
    +-- kasjer_1.log
```