

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 oplacają przy kasach.

Symulacja działa na **osobnych procesach** (fork + exec) komunikujących się przez mechanizmy IPC Systemu V (pamiec dzielona, semafory, kolejki komunikatów) oraz lacza (pipe, FIFO).

2. Procesy i wątki

Kierownik (`kierownik.c`) – główny proces

- Tworzy wszystkie zasoby IPC (shm, semafory, 3 kolejki, pipe, FIFO).
- Uruchamia dzieci: `fork() + execl()` – piekarz (1), kasjerzy (2), klienci (N).
- Prowadzi **zegar symulacji** (kazda iteracja petli = 1 minuta symulacyjna).
- Co 1-10 minut generuje batch 2-8 nowych klientów.
- Otwiera/zamyka druga kasa w zależności od liczby klientów.
- Nasłuchiwa polecen z FIFO (inwentaryzacja, 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 główny wątek.
- Główna pętla: `msgrecv()` 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` – wejście do sklepu (maks. N osób).
2. Losuje liste zakupow (2-5 produktów, 1-3 szt. kazdego).
3. `msgrecv()` z kolejki podajnikow (`mtype = product_id + 1`) – pobiera ciastka.
4. Wybiera kase z krótsza kolejka – `msgsnd()` koszyk do checkout.
5. `msgrecv()` paragon (`mtype = getpid()`) – czeka na swój paragon.
6. `sem_signal(SEM_SHOP_ENTRY)` z `SEM_UNDO` – zwalnia miejsce.
7. Ewakuacja: odkłada produkty do kosza w SHM i natychmiast wychodzi.

Schemat procesów i wątków

```

kierownik (PID główny)
|
+-- 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) Pamięć dzielona (System V Shared Memory)

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

- Konfiguracja: ile produktów, max klientów, skala czasu, godziny otwarcia/zamknięcia
- Stan sklepu: klientów w środku, kasły otwarte, długości kolejek
- Statystyki: sprzedaż na każdej kasie, produkcja
- Flagi: `simulation_running`, `shop_open`, `evacuation_mode`
- Zegar: `sim_hour`, `sim_min`
- PIDy procesów (piekarz, kasjerzy)

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

b) Semafora (System V)

Jeden zbiór semaforów (`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 używa **SEM_UNDO** – automatycznie zwalnianie semafora jeśli proces zostanie zabity (`kill -9`). Zapobiega to trwalemu 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	Tresc
Podajniki	piekarz -> klient	product_id + 1	ConveyorMsg (produkt)
Checkout	klient -> kasjer	register_id + 1	CheckoutMsg (koszyk)
Paragony	kasjer -> klient	customer_pid	ReceiptMsg (paragon)

Filtrowanie `msgrecv()` przez `mtype`: klient pobiera konkretne ciasto, kasjer obsługuje swoja 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 `msgrecv()` – `sem_signal(guard)`. Zapobiega to przepelnieniu kolejki i zablokowaniu `msgsnd`.

Wywołania: `msgget()`, `msgsnd()`, `msgrecv()`, `msgctl(IPC_RMID)`, `msgctl(IPC_STAT)`.

d) Lacze 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) Lacze 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 koncowy
- `dup2()` – przekierowanie stderr dzieci do plikow logow
- `popen("date ...")` – pobranie daty systemowej do raportu
- `mkdir("logs")` – tworzenie katalogu logow
- `unlink()` – usuwanie pliku klucza i FIFO przy czyszczeniu

4. Sygnaly

Sygnal	Kto wysyla	Do kogo	Co robi
SIGCHLD	kernel	kierownik	Dziecko zakonczylo – <code>waitpid(WNOHANG)</code> zbiera
SIGINT	Ctrl+C / test	kierownik	Czyste zamkniecie z raportem
SIGTERM	kierownik	piekarz, kasjery, klienci	Zakonczenie procesu
SIGUSR1	kierownik	dzieci	Inwentaryzacja (sygnal1)
SIGUSR2	kierownik	dzieci	Ewakuacja – natychmiast wyjdz (sygnal2)

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

5. Synchronizacja – problemy i rozwiazania

Wyscig przy dostepie do SHM

`SEM_SHM_MUTEX` (semafor binarny) z `SEM_UNDO`. Kazdy dostep do `SharedData` owiniety w `sem_wait_undo/sem_signal_`

Za duzo klientow w sklepie

`SEM_SHOP_ENTRY` (semafor zliczajacy, init = N) z `SEM_UNDO`. Klient dekrementuje przy wejsciu, inkrementuje przy wyjsciu. Slot zwalniany jesli klient zginie.

Podajnik pelny

`SEM_CONVEYOR_BASE+i` (init = Ki). Piekarz robi `sem_trywait` (nieblokujacy) – jesli 0, podajnik pelny, pomija produkt.

Zombie procesy

Handler `SIGCHLD + waitpid(WNOHANG)` w petli glownej. Rozroznia smierc klienta od smierci piekarza/kasjera (nie dekrementuje `active_customers` dla nie-klientow).

Wyciek IPC po crashu

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

Przyspieszenie zegara przez SIGCHLD

`msleep_safe()` – uzywa `nanosleep()` z petla retry na EINTR. Zapobiega skroceniu snu przez sygnal.

6. Zarzadzanie kasami

Kierownik co iteracje sprawdza liczbe klientow:

- $\geq N/2$ klientów -> otwiera kase 1 (`register_open[1] = 1`)
- $< N/2$ klientów -> kasa 1 przestaje przyjmowac (dokonczy kolejke)
- kolejka kasy 1 = 0 -> kasa 1 zamknieta

Kasa 0 jest **zawsze otwarta**.

7. Zamykanie symulacji

Trzy sposoby zamkniecia:

1. **Normalne** – zegar dochodzi do godziny zamkniecia (Tk)
2. **Ctrl+C / SIGINT** – czyste zamkniecie 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 az klienci wyjda sami
3. SIGTERM do piekarza, kasjerow, klientow
4. `waitpid()` zbiera procesy (z timeoutem 5s)
5. Jesli ktos nie odpowiedzial -> SIGKILL jako ostatecznosc
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 -> zamkniecie -> 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 dzieki SEM_UNDO.

9. Obsluga bledow

Dedykowany modul `error_handler.c` z funkcjami:

- `handle_error(msg)` – perror() + wypisanie errno + exit(EXIT_FAILURE) – blad krytyczny
- `handle_warning(msg)` – perror() bez przerwania – ostrzezenie
- `check_sys_call(ret, msg, fatal)` – wrapper na sprawdzanie wartosci zwracanej
- `validate_int_range(val, min, max, name)` – walidacja parametrow z czytelnym komunikatem

Kazde wywolanie systemowe (`fork`, `shmget`, `semget`, `msgget`, `pipe`, `mkfifo`, `open`, `exec1`, `pthread_create`) jest sprawdzane. Bledy `semop` z EINTR sa powtarzane, z EIDRM/EINVAL – ignorowane (shutdown).

10. Funkcje wymagane przez projekt (gdzie szukac)

- **Pliki:** `creat()`, `open()`, `close()`, `read()`, `write()`, `unlink()` – kierownik.c, ipc_utils.c
- **Procesy:** `fork()`, `exec1()`, `exit()`, `waitpid()` – kierownik.c
- **Watki:** `pthread_create()`, `pthread_join()`, `pthread_detach()`, `pthread_mutex_lock/unlock()`, `pthread_cond_wait/signal/broadcast()` – piekarz.c, kasjer.c
- **Sygnaly:** `kill()`, `sigaction()` – kierownik.c, piekarz.c, kasjer.c, klient.c
- **Semafory:** `ftok()`, `semget()`, `semctl()`, `semop()` – ipc_utils.c
- **Lacza:** `mkfifo()`, `pipe()`, `dup2()`, `popen()` – ipc_utils.c, kierownik.c
- **Pamiec dzielona:** `ftok()`, `shmget()`, `shmat()`, `shmdt()`, `shmctl()` – ipc_utils.c
- **Kolejki komunikatow:** `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/
    +-- raport.txt            Generowany automatycznie
    +-- piekarz.log
    +-- kasjer_0.log
    +-- kasjer_1.log

```