Ciastkarnia – Opis Projektu

Systemy Operacyjne – Temat 15

Tomasz Pulka

1. Opis zadania

Symulacja ciastkarni z samoobslugowym sklepem firmowym. Ciastkarnia produkuje P roznych produktow (P > 10), kazdy w innej cenie. Produkty po wypieku trafiaja na oddzielne podajniki (FIFO o pojemnosci Ki), a klienci pobieraja je w sklepie i oplaca przy kasach.

Symulacja dziala na **osobnych procesach** (fork + exec) komunikujacych sie przez mechanizmy IPC Systemu V (pamiec dzielona, semafory, kolejki komunikatow) oraz lacza (pipe, FIFO).

2. Procesy i watki

Kierownik (`kierownik.c`) – glowny 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 klientow.
- Otwiera/zamyka druga kase w zaleznosci od liczby klientow.
- Nasluchuje polecen z FIFO (inwentaryzacja, ewakuacja).
- Na koniec generuje raport i sprzata 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
```

Kazde dziecko dostaje sciezke do pliku klucza (`ciastkarnia.key`) jako argument i uzywa `ftok()` do podlaczenia sie do tych samych zasobow IPC.

Piekarz (`piekarz.c`)

- 2 watki produkcyjne (`pthread_create`): watek 0 – produkty 0-5, watek 1 – 6-11.
- Petla: losowa partia – `sem_trywait(podajnik)` – `msgsnd()` do kolejki podajnikow.
- Raportuje produkcje do kierownika przez **pipe** (`write()`).
- Wspolny licznik chroniony `pthread_mutex_t`.

Kasjer (`kasjer.c`)

- 2 instancje (kasa 0, kasa 1). Kazda ma watek 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/zamkniecia
- 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) Semafory (System V)

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

Indeks	Nazwa	Init	Typ	Zastosowanie
0	`SEM_SHM_MUTEX`	1	Binarny	Ochrona dostepu do pamieci dzielonej
1	`SEM_SHOP_ENTRY`	N	Zliczajacy	Kontrola maks. N klientow w sklepie
2..P+1	`SEM_CONVEYOR_BASE+i`	Ki	Zliczajacy	Wolne miejsca na podajniku i-tego produktu
P+2	`SEM_GUARD_CONVEYOR`	limit	Zliczajacy	Backpressure kolejki podajnikow
P+3	`SEM_GUARD_CHECKOUT`	limit	Zliczajacy	Backpressure kolejki checkout
P+4	`SEM_GUARD_RECEIPT`	limit	Zliczajacy	Backpressure kolejki paragonow

Kluczowe: mutex i shop_entry uzywaja **`SEM_UNDO`** – automatycznie zwalnianie semafor jesli proces zostanie zabity (`kill -9`). Zapobiega to trwalemu deadlockowi.

Wywolania: `semget()`, `semctl(SETVAL)`, `semctl(GETVAL)`, `semop()`, `semctl(IPC_RMID)`.

c) Kolejki komunikatow (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 `msgrcv()` przez `mtype`: klient pobiera konkretne ciastko, kasjer obsluguje swoja kase, klient czeka na swoj paragon po PID.

Guard semaphores: kazda kolejka ma semafor zliczajacy inicjalizowany na `msg_qbytes / sizeof(msg)`. Przed `msgsnd()` – `sem_wait(guard)`, po `msgrcv()` – `sem_signal(guard)`. Zapobiega to przepelnieniu kolejki i zablokowaniu `msgsnd`.

Wywolania: `msgget()`, `msgsnd()`, `msgrcv()`, `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()`, wiec obaj maja te same deskryptory.

Wywolania: `pipe()`, `read()`, `write()`, `close()`.

e) Lacze nazwane (FIFO)

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

Uzytkownik -> Kierownik: polecenia tekstowe `inwentaryzacja` lub `ewakuacja`. Kierownik otwiera FIFO jako `O_RDONLY | O_NONBLOCK` i czyta w kazdej iteracji petli.

Wywolania: `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 – `waitpid(WNOHANG)` 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)` – sprzata 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:

- `>= N/2 klientow` -> otwiera kase 1 (`register_open[1] = 1`)
- `< N/2 klientow` -> 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`)

Testy skupiaja sie na **komunikacji miedzy procesami (IPC)** i **edge case'ach**. Kazdy test:

1. Opisuje **cel** — jaki mechanizm IPC jest testowany
2. Ustawia **parametry** wymuszajace edge case (np. maly sklep, zabicie procesu)
3. Formuje **wnioski** — czy IPC dziala poprawnie w ekstremalnych warunkach

#	Skrypt	IPC testowane	Edge case
01	`test_01_piekarz_...`	msg queue podajnikow	Zabicie piekarza
02	`test_02_klient_...`	msg queue paragonow	Filtrowanie mtype=PID
03	`test_03_msgqueue_...`	msg queue (kontencja)	1 produkt, wielu klientow
04	`test_04_pipe_...`	pipe()	Duzo write() na pipe
05	`test_05_sem_undo_...`	semafory SEM_UNDO	kill -9 klienta

Test 01: Piekarz → Klient – brak podazy

Parametry: `-t 15 -s 30 -n 8 -o 8 -c 12`

Testuje kolejke komunikatow podajnikow (`msgsnd`/`msgrcv` z `mtype = product_id + 1`). Zabijamy piekarza w trakcie symulacji — klienci probuja pobrac produkty z pustej kolejki (`msgrcv` z `IPC_NOWAIT`

zwraca `ENOMSG`). Weryfikacja: klienci nie zakleszczaja sie, wychodzą ze sklepu jako "nieobsluzeni", symulacja konczy sie normalnie.

Test 02: Klient → Kasjer – paragony (mtype = PID)

Parametry: `-t 12 -s 20 -n 10 -o 8 -c 14`

Testuje dwa typy kolejek: checkout (`klient → kasjer`, `mtype = register_id + 1`) oraz paragony (`kasjer → klient`, `mtype = customer_pid`). Przy duzym ruchu wielu klientow jednoczesnie placi — kazdy musi dostac SWOJ paragon (filtrowanie `msgrcv` po PID). Weryfikacja: `customers_served + customers_not_served == total_entered` (brak zgubionych).

Test 03: Piekarz → Klienci – rywalizacja o msgrcv na jednym mtype

Parametry: `-t 12 -s 25 -n 10 -p 1 -o 8 -c 12`

Testuje kolejke komunikatow gdy wielu klientow rywalizuje o ten sam produkt (`msgrcv` z `mtype = 1`). Tylko 1 produkt (Bulka), kazdy klient chce kupic 1-3 szt. Kazda bulka to 1 `msgsnd()`, kazdy zakup to 1 `msgrcv()` — wysoka kontencja. Weryfikacja: `served <= produced` (brak duplikacji wiadomosci), `served > 0` (msgrcv dziala mimo rywalizacji), brak zakleszczenia.

Test 04: Pipe – raporty produkcji

Parametry: `-t 10 -s 15 -n 6 -o 8 -c 12`

Testuje lacze nienazwane (pipe) miedzy piekarzem a kierownikiem. Piekarz wysyla raporty produkcji (`"BATCH:tid:count\n"`) — kierownik odczytuje je i aktualizuje `baker_produced[]` w SHM. Szybka produkcja (skala 15ms) wymusza duzo `write()`. Weryfikacja: `baker_produced` rosnie w czasie, pipe nie blokuje (brak deadlocka `write()`).

Test 05: SEM_UNDO – odpornosc na kill -9

Parametry: `-t 20 -s 30 -n 3 -o 8 -c 14`

Testuje mechanizm `SEM_UNDO` w semaforach System V. Klient w sklepie trzyma slot `SEM_SHOP_ENTRY` — `kill -9` go zabija. Bez `SEM_UNDO` slot bylby stracony i nowi klienci nie mogliby wejsc (deadlock). Z `SEM_UNDO` kernel automatycznie cofa operacje semafora. Weryfikacja: `sem_shop_entry` wraca do wyzszej wartosci, nowi klienci wchodza.

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`, `execl`, `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()`, `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
- **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_piekarz_klient_brak_podazy.sh
|   +-- test_02_klient_kasjer_paragony.sh
|   +-- test_03_msgqueue_kontencja_mtype.sh
|   +-- test_04_pipe_raporty_produkcji.sh
|   +-- test_05_sem_undo_kill.sh
+-- logs/                       Generowany automatycznie
    +-- raport.txt
    +-- piekarz.log
    +-- kasjer_0.log
    +-- kasjer_1.log
```