

# Temat 16 - Dyskont [Raport Projektu]

---

**Kamil Karpieł**

Grupa 1 | Studia stacjonarne | III semestr | Nr albumu 155184

**UWAGA:** Symulacja jest nieskończona. Zdefiniowana jest określona pula klientów, a po ich obsłudze proces czeka: albo na pojawienie się nowego klienta, albo na manualną ewakuację (zakończenie programu) przez kierownika.

**Uwaga dotycząca logów:** W symulacji tag `[BLAD]` jest używany nie tylko do zgłaszania awarii programu, ale również do obsługi zdarzeń takich jak np. **weryfikacji wieku** (gdy niepełnoletni klient próbuje kupić alkohol). W takim przypadku tag `[BLAD]` oznacza odmowę sprzedaży.

## 0. Środowisko i kompilacja

Projekt został przetestowany w środowisku Linux (Torus) przy użyciu następujących narzędzi:

- **GCC:** 8.5.0
- **GNU Make:** 4.3

### Kompilacja

Aby skompilować projekt, należy wykonać polecenie:

```
make
```

### Uruchomienie

Składnia polecenia:

```
./dyskont.out <pula_klientow> <max_klientow_sklep> <nr_testu*>
```

### Argumenty:

- `<pula_klientow>`: Całkowita liczba klientów do stworzenia w symulacji.
- `<max_klientow_sklep>`: Maksymalna liczba klientów przebywających jednocześnie w sklepie.
- `<nr_testu*>`: (Opcjonalne) Tryb testu: `0` = normalny (symulacja czasu zakupów), `1` = test obciążeniowy (bez opóźnień/sleepów). Domyślnie `0`.

### Wyświetlanie paragonów (Tryb Cichy)

Domyślnie program uruchamia klientów w trybie cichym (flaga `-quiet` w kodzie), aby nie zaśmiecać konsoli tysiącami paragonów. Aby włączyć wyświetlanie paragonów dla każdego klienta należy:

1. Otworzyć plik `main.c`.

2. Znaleźć linię uruchamiającą proces **klient** (ok. linii 351).
3. Usunąć argument **"-quiet"** z funkcji **exec1**.
4. Skompilować projekt ponownie (**make**).

## Obsługa Panelu Kierownika

Po uruchomieniu programu w osobnej konsoli (np. drugiej karcie terminala) można uruchomić panel sterowania:

```
./kierownik
```

Panel umożliwia interaktywne zarządzanie sklepem:

```
=== PANEL KIEROWNIKA ===
1. Otworz kase stacjonarna 2
2. Zamknij kase stacjonarna 1
3. Zamknij kase stacjonarna 2
4. EWAKUACJA (SIGTERM)
5. Pokaz status kolejek
0. Wyjście
Wybor:
```

## 1. Założenia projektu

Celem projektu było stworzenie symulacji działania **dyskontu**, opartej na **architekturze wieloprotocowej** (proces: *klienta*, *pracownika*, *kasjera*, *kasy samoobsługowej* oraz *sam dyskontu*), a także na **wątkach** (*logów* podpiętych do procesu głównego). Komunikacja pomiędzy procesami jak i wątkami została zrealizowana z wykorzystaniem szeregu różnych **mechanizmów komunikacji międzyprocesowej (IPC)** wbudowanych w **system Linux**.

Szczegółowe wymagania dotyczące realizacji projektu znajdują się w pliku:  
<https://github.com/MrKamkar/Dyskont/blob/main/README.md>

## 2. Ogólny opis plików

Projekt został podzielony na moduły funkcjonalne:

- **main.c**: Główny proces zarządczy. Inicjalizuje zasoby IPC, a następnie uruchamia procesy-managery. Obsługuje sygnały systemowe i w przypadku ewakuacji uruchamia osobny **wątek sprzątający** (**WatekSprzatajacy**), który asynchronicznie zamyka procesy potomne, pozwalając procesowi głównemu na finalne zwolnienie zasobów IPC po zakończeniu wszystkich dzieci.
- **pamiec\_wspoldzielona.c / .h**: Zawiera definicję struktury **StanSklepu** oraz funkcje zarządzające segmentem pamięci współdzielonej. Przechowuje ona tablicę statusów kas, liczniki klientów, flagi ewakuacji oraz magazyn produktów, umożliwiając wszystkim procesom dostęp do wspólnego stanu symulacji w czasie rzeczywistym.

- **semafor.c / .h**: Biblioteka do operacji na semaforach Systemu V. Implementuje między innymi semafor zliczający wolne miejsca w kolejkach komunikatów, co zapobiega systemowemu blokowaniu procesów przy przepełnieniu buforów kernela.
- **logi.c / .h**: System logowania oparty na dedykowanym wątku i kolejce komunikatów. Procesy przesyłają logi do kolejki, a wątek loggera (działający w procesie głównym) odbiera je, formatuje z użyciem kolorów ANSI i zapisuje do rotowanego pliku w katalogu **logs/**.
- **klient.c / .h**: Proces **Generatora Klientów**, który w pętli **fork**-uje właściwe procesy klientów. Każdy klient jest autonomicznym procesem. Klienci oczekujący w kolejce do kasy samoobsługowej uruchamiają pomocniczy **wątek sprawdzający**, który po czasie **T** monitoruje dostępność kas stacjonarnych i umożliwia migrację, jeśli te są wolne.
- **kasjer.c / .h**: Proces **Managera Kas Stacjonarnych**. Obsługuje Kasę 1 bezpośrednio, a w razie zapotrzebowania **fork**-uje proces Kasy 2. Posiada **wątek zarządzający**, który monitoruje wspólną kolejkę wejściową i rozdziela klientów do kolejek prywatnych konkretnych kas na podstawie ich długości i stanu.
- **kasa\_samoobslugowa.c / .h**: Proces **Managera Kas Samoobsługowych**. Zarządza pulą procesów-robotów (Kasy 1-5). Posiada **wątek skalujący**, który dynamicznie tworzy nowe kasy (fork) lub wysyła sygnał **SIGUSR1** do istniejących w celu ich łagodnego zamknięcia, bazując na liczbie klientów w sklepie.
- **pracownik\_obsługi.c / .h**: Proces **Pracownika**, który nasłuchuje na dedykowanej kolejce zadań. Odpowiada za losowe odblokowywanie zaciętych kas samoobsługowych oraz weryfikację wieku klientów kupujących alkohol (na podstawie wieku przekazanego w komunikacie).
- **kierownik.c / .h**: Interaktywny panel sterowania. Pozwala użytkownikowi na ręczne wysyłanie sygnałów **SIGUSR1/SIGUSR2** do procesu głównego (sterowanie kasami) oraz **SIGTERM** (ewakuacja).
- **kolejki.c / .h**: Warstwa abstrakcji nad IPC Message Queues. Implementuje bezpieczne wysyłanie komunikatów z użyciem semaforów oraz mechanizm **VIP (odpowiedzi)**, który gwarantuje dostarczenie komunikatu zwrotnego do klienta nawet przy pełnych kolejkach wejściowych.

### 3. Co udało się zrealizować?

- **Pełna symulacja wieloprotokowa**: Udało się stworzyć stabilnie działający system symulujący pracę dyskontu, w którym każdy podmiot (kasjer, klient, kierownik, kasa samoobsługowa) jest osobnym procesem.
- **Bezpieczna obsługa sygnałów**: Wdrożono mechanizm obsługi sygnałów, gdzie handler przekazuje sygnały do procesów potomnych (np. ewakuacja) lub inicjuje wątek czyszczący, a właściwa logika sprzątnięcia wykonywana jest poza handlerem.
- **Wątek loggera**: Zastosowanie osobnego wątku do logowania operacji (z kolejką komunikatów) znacząco odciążało główne procesy, eliminując opóźnienia związane z operacjami wejścia/wyjścia.
- **Inteligentne zarządzanie kolejkami**: Klienci dynamicznie wybierają kasę (krótsza kolejka, stan kasy) oraz potrafią migrować do nowo otwartej kasy na sygnał kierownika.
- **Bezpieczne zamykanie aplikacji i czyszczenie zasobów IPC**: System poprawnie reaguje na sygnały różne zamykania aplikacji, rozpoczynając procedurę ewakuacji klientów, a następnie bezpiecznie kończy

wszystkie procesy potomne i zwalnia zasoby IPC.

## Elementy wyróżniające (Dodatkowe funkcjonalności)

- **Kolorowanie wyjścia terminala:** Wątek loggera dynamicznie koloruje logi w zależności od ich wag (czerwone błędy, żółte ostrzeżenia, zielone info), co znacząco poprawia czytelność symulacji w czasie rzeczywistym.

## 4. Z czym były problemy?

- **Procesy zombie:** Występował problem z niepoprawnym zliczaniem procesów potomnych. Rozwiązano to poprzez użycie dedykowanych wątków lub pętli głównej z funkcją `wait` / `waitpid`, co pozwala na bieżące odbieranie statusów zakończonych procesów potomnych bez blokowania głównej logiki symulacji.
- **Deadlocki:** Zdarzały się, gdy procesy blokowały się na operacjach IPC podczas zamykania. Rozwiązano to poprzez zastosowanie **mechanizmu bezpiecznej obsługi sygnałów** oraz wdrożenie procedury, która przy zamykaniu aplikacji (SIGTERM) aktywnie przerywa oczekiwanie i sprząta zasoby.
- **Problemy z synchronizacją:** Początkowo projekt opierał się na kolejkach zaimplementowanych ręcznie w pamięci współdzielonej i semaforach, co prowadziło do trudnych asynchronicznych błędów w odczycie/zapisie kolejek. Problem rozwiązano poprzez całkowite przejście na **kolejki komunikatów**, które zapewniają spójność operacji na danych.
- **Aktywne oczekiwanie (Polling):** Wstępna wersja projektu wykorzystywała pętle z `usleep()`, co niepotrzebnie obciążało procesor. Cały kod został przebudowany na **metody blokujące** (operacje na semaforach, oczekiwanie na komunikat w kolejce), dzięki czemu procesy czekają na zdarzenie, zamiast aktywnie sprawdzać ich stan.
- **Przepełnienie kolejek:** Przy dużej liczbie równoczesnych klientów dochodziło do zapychania się kolejek komunikatów, ponieważ brakowało obsługi tzw. komunikatu awaryjnego na odpowiedź. Procesy blokowały się, nie mogąc wysłać potwierdzenia do klienta. Rozwiązano to, dodając obsługę priorytetowych odpowiedzi (VIP), które gwarantują dostarczenie wiadomości z powodu braku ograniczeń na semaforach.

## 5. Mechanizmy synchronizacji i IPC

- **Pamięć dzielona (`shmget` / `shmat`):** Obiekt `StanSklepu` jest mapowany przez każdy proces. Zawiera:

- Tablice struktur `Kasa` (statusy, liczniki kolejek).
- Liczniki globalne (liczba klientów w sklepie, flagi ewakuacji).
- Magazyn produktów.

- **Semaforey (`semop`):** Zestaw semaforów kontroluje dostęp do zasobów, zapobiegając niesynchronizowanemu dostępowi do danych:

- `MUTEX_PAMIEC_WSPOLDZIELONA`: Chroni modyfikację stanu sklepu (blokuje dostęp do pamięci współdzielonej).
- `MUTEX_KOLEJKI_VIP`: Synchronizuje operacje podnoszenia limitu kolejki dla odpowiedzi VIP.
- `SEM_NOWY_KLIENT`: Sygnalizuje nowego klienta dla wątku skalującego kas samoobsługowych.

## - Kolejki komunikatów (`msgrcv/msgsnd`):

Umożliwiają dwukierunkową wymianę danych między procesami:

- **Osobne kolejki + Wspólna Kolejka:** Klienci decydujący się na kasę stacjonarną wysyłają zgłoszenie do `ID_IPC_KASA_WSPOLNA`. **Wątek Zarządzający** (w procesie `kasjer`) odbiera te zgłoszenia i przekierowuje je do prywatnych kolejek kas (`ID_IPC_KASA_1` lub `ID_IPC_KASA_2`) w zależności od ich obciążenia i dostępności.
- **Informacja zwrotna:** Klient nasłuchuje na kanale `MSG_RES_STACJONARNA_BASE + id_klienta` (dla kas stacjonarnych) lub `MSG_RES_SAMOOSLUGA_BASE + id_klienta` (dla kas samoobsługowych). Dzięki temu odbiera komunikat skierowany wyłącznie do niego. W odpowiedzi otrzymuje ID kasy, która go obsługiwała.
- **Mechanizm VIP:** Funkcja `WyslijKomunikatVIP` tymczasowo podnosi limit kolejki, by zagwarantować miejsce na odpowiedź nawet przy pełnej kolejce.
- **Logowanie asynchroniczne:** Procesy wysyłają logi do dedykowanej kolejki komunikatów, która jest opróżniana przez osobny wątek loggera. Zapis na dysk odbywa się w tym osobnym wątku, minimalizując wpływ operacji I/O na płynność symulacji.

## 6. Kluczowe Pseudokody

### 1. main.c (Proces Główny)

```

Inicjalizacja:
    utworz_pamiec_wspoldzielona()
    utworz_semafony()
    utworz_kolejki_komunikatow()
    uruchom_watek_loggera()

Uruchamianie Managerów:
    fork() -> exec("kasjer")           // Manager Kas Stacjonarnych + Kasa 1
    fork() -> exec("kasa_samoobslugowa") // Manager Kas Samoobsługowych + Kasa 0
    fork() -> exec("pracownik")        // Pracownik Obsługi
    fork() -> exec("klient")           // Generator Klientów

Pętla główna:
    wait() // Czekanie na zakończenie procesów (lub sygnały)

Obsługa Sygnałów:
    SIGTERM -> Rozpocznij procedurę ewakuacji (kill SIGTERM do process group)
    SIGUSR1/2 -> Przekaż sygnał do Managera Kas Stacjonarnych

```

### 2. klient.c (Generator + Klient)

```

Generator (Proces Główny):
    WHILE pula_klientow > 0:
        fork() -> Proces Klienta

```

```
        sleep(losowy_czas)
        czekaj_na_wejscie(SEM_WEJSCIE_DO_SKLEPU)

Proces Klienta:
    zrob_zakupy()

    IF samoobsługa:
        StartWatekSprawdzajacy(Timeout -> Idz do stacjonarnej)
        Wysluj(MSG_SAMOBSLUGA)
        Odbierz(MSG_RES_SAMO + ID)
        ZabijWatekSprawdzajacy()

    IF stacjonarna OR przekierowany:
        Wysluj(MSG_WSPOLNA) // Wspólna kolejka
        Odbierz(MSG_RES_STACJONARNA + ID)

Wyjście:
    Zwolnij(SEM_WEJSCIE)
    exit()
```

### 3. kasjer.c (Manager Kas Stacjonarnych)

```
Proces Główny (Kasa 1):
    StartWatekZarządzajacy()
    WHILE TRUE:
        Odbierz(MSG_KASA_1) // Prywatna kolejka
        ObsluzKlienta()
        WyslujVIP(Odpowiedz)

Wątek Zarządzający:
    WHILE TRUE:
        Msg = Odbierz(MSG_WSPOLNA)

        IF (K1 zamknięta AND duża_kolejka):
            OtworzKase1()

        Decyzja = WybierzKrotszaKolejke(K1, K2)
        Przekieruj(Msg -> Decyzja)

        Migracja() // Opcjonalnie przenieś z K1 do K2 jeśli K2 pusta
```

### 4. kasa\_samoobslugowa.c (Manager Kas Samoobsługowych)

```
Proces Główny (Kasa 0):
    StartWatekSkalujacy()
    WHILE TRUE:
        ObsluzKlienta(0)

Wątek Skalujący:
```

```
WHILE TRUE:
    Czekaj(SEM_NOWY_KLIENT)
    Wymagane = ObliczLiczbeKas(Klienci)

    IF Aktywne < Wymagane:
        fork() -> Nowa Kasa
    ELSE IF ZaDuzoKas:
        Wysluj(SIGUSR1 -> Kasa) // łagodne zamykanie

    ZbierzZombie()
```

## 7. Wykonane testy

Poniżej przedstawiono zbiór testów weryfikujących poprawność działania mechanizmów IPC w symulacji.

### Test 1 – Test Obciążeniowy (Brak Deadlocków)

- **Polecenie:** `./dyskont.out 10000 1000 1`
- **Opis:** Weryfikacja stabilności systemu przy ekstremalnie szybkiej generacji i obsłudze procesów (tryb bez `usleepów`). Test weryfikuje odporność na deadlocki, poprawne czyszczenie procesów zombie oraz poprawność działania algorytmu skalowania kas (+/- 1 kasa na każde K klientów, minimum 3).
- **Oczekiwany wynik:**
  1. Kasa swoje kończy działanie automatycznie po obsłużeniu wszystkich 10 000 klientów i upłynięciu czasu bezczynności.
  2. Brak procesów zombie (wszystkie procesy klientów są poprawnie odebrane przez `wait` w procesie głównym generatora, co można potwierdzić brakiem logów o zombie).
  3. Weryfikacja logiki skalowania: Gdy liczba klientów w sklepie spadnie do 0, system dynamicznie przez wątek skalujący redukuje liczbę kas samoobsługowych do poziomu 3, zgodnie z wzorem  $\$klientów < K \cdot (N-3)$ .
  4. Brak komunikatów błędów IPC na standardowym wyjściu błędów.
  5. `kierownik` (opcja 5) pokazuje na koniec 0 klientów w sklepie oraz powrót do 3 czynnych kas samoobsługowych.

#### Obserwacja przy ekstremalnym obciążeniu (2000+ procesów):

Przy uruchomieniu symulacji z bardzo dużą liczbą równoczesnych klientów (np. 2000), można zaobserwować kilkusekundowe opóźnienie w uruchamianiu dodatkowych kas (np. stan 5/6 przez 5 sekund). Jest to zjawisko naturalne, wynikające z kolejkowania procesów przez planer systemu operacyjnego. Proces managera musi konkurować o czas procesora z tysiącami procesów klientów, co wydłuża czas wykonania operacji `fork()` potrzebnej do utworzenia nowego procesu kasy.

```

22:10:05] [INFO] Klient [PID: 1325372] zakonczyl zakupy pomyslnie (status: 0)
22:10:05] [INFO] Kasa samoobslugowa [6]: Odblokowana
22:10:05] [INFO] Kasa samoobslugowa [6]: Weryfikacja wieku..
22:10:05] [INFO] Pracownik: Odblokowanie kasy 5
22:10:05] [DEBUG] Kasa samoobslugowa [3]: Czekam na klienta..
22:10:05] [INFO] Pracownik: Weryfikacja wieku OK (lat: 90) dla kasy 6
22:10:05] [INFO] Kasa samoobslugowa [2]: Odblokowana
22:10:05] [INFO] Kasa samoobslugowa [2]: Weryfikacja wieku..
22:10:05] [INFO] Kasa samoobslugowa [6]: Klient [1325377] zaplacil 45.00 PLN
22:10:05] [INFO] Pracownik: Weryfikacja wieku NIEUDANA (lat: 7) dla kasy 2
22:10:05] [DEBUG] Kasa samoobslugowa [1]: Czekam na klienta..
22:10:05] [BLAD] Kasa samoobslugowa [2]: Klient niepelnoletni!
22:10:05] [INFO] Kasa samoobslugowa [5]: Odblokowana
22:10:05] [OSTRZ] Kasa samoobslugowa [5]: BLOKADA!
22:10:05] [DEBUG] Klient [ID: 1325377] otrzymal odpowiedz od kasy samoobslugowej [6]
22:10:05] [DEBUG] Klient [ID: 1325381] otrzymal odpowiedz od kasy samoobslugowej [2]
22:10:05] [BLAD] Klient [ID: 1325381] niepelnoletni (probował kupic alkohol). Opuszcza sklep bez zakupow
22:10:05] [INFO] Pracownik: Odblokowanie kasy 5
22:10:05] [DEBUG] Kasa samoobslugowa [6]: Czekam na klienta..
22:10:05] [DEBUG] Kasa samoobslugowa [2]: Czekam na klienta..
22:10:05] [INFO] Kasa samoobslugowa [PID: 1315315]: Konczy dzialanie (SIGUSR1 - bezczynna)
22:10:05] [INFO] Klient [PID: 1325380] zakonczyl zakupy pomyslnie (status: 0)
22:10:05] [INFO] Klient [PID: 1325383] zakonczyl zakupy pomyslnie (status: 0)
22:10:05] [INFO] Zamknieto kase 3
22:10:05] [INFO] Kasa samoobslugowa [5]: Odblokowana
22:10:05] [INFO] Kasa samoobslugowa [5]: Weryfikacja wieku..
22:10:05] [INFO] Pracownik: Weryfikacja wieku OK (lat: 33) dla kasy 5
22:10:05] [INFO] Kasa samoobslugowa [5]: Klient [1325384] zaplacil 101.50 PLN
22:10:05] [BLAD] Klient [PID: 1325381] odlozyl zakupy i wyszedl ze sklepu. (status: 1)
22:10:05] [DEBUG] Klient [ID: 1325384] otrzymal odpowiedz od kasy samoobslugowej [5]
22:10:05] [INFO] Kasa samoobslugowa [PID: 1325461]: Konczy dzialanie (SIGUSR1 - bezczynna)
22:10:05] [INFO] Klient [PID: 1325377] zakonczyl zakupy pomyslnie (status: 0)
22:10:05] [INFO] Zamknieto kase 6
22:10:05] [DEBUG] Kasa samoobslugowa [5]: Czekam na klienta..
22:10:05] [INFO] Klient [PID: 1325384] zakonczyl zakupy pomyslnie (status: 0)
22:10:34] [INFO] Kasa 1: Zamkniete (Limit czasu)

```

Zrzut 1: Kasa kończy działanie po czasie bezczynności

```

22:10:05] [BLAD] Klient [PID: 1325381] odlozyl zakupy i wyszedl ze sklepu. (status: 1)
22:10:05] [DEBUG] Klient [ID: 1325384] otrzymal odpowiedz od kasy samoobslugowej [5]
22:10:05] [INFO] Kasa samoobslugowa [PID: 1325461]: Konczy dzialanie (SIGUSR1 - bezczynna)
22:10:05] [INFO] Klient [PID: 1325377] zakonczyl zakupy pomyslnie (status: 0)
22:10:05] [INFO] Zamknieto kase 6
22:10:05] [DEBUG] Kasa samoobslugowa [5]: Czekam na klienta..
22:10:05] [INFO] Klient [PID: 1325384] zakonczyl zakupy pomyslnie (status: 0)
22:10:34] [INFO] Kasa 1: Zamkniete (Limit czasu)
^Z
[1]+  Stopped                  ./dyskont.out 10000 1000 1
karpieł.kamil.155184@torus:~/SO_Projekty/Dyskont$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
karpieł+ 1108016  0.0  0.0  11788  8216 pts/9    Ss   21:29   0:00 -bash
karpieł+ 1315308 20.7  0.0  76312   560 pts/9    TL   22:09   0:19 ./dyskont.out 10000 1000 1
karpieł+ 1315310  0.4  0.0  10776   560 pts/9    Sl   22:09   0:00 kasjer
karpieł+ 1315311  4.7  0.0  10772   564 pts/9    Sl   22:09   0:04 kasa_samoobslugowa
karpieł+ 1315312  2.0  0.0   2444   560 pts/9    S    22:09   0:01 pracownik
karpieł+ 1315314  1.8  0.0   2444   104 pts/9    S    22:09   0:01 kasa_samoobslugowa
karpieł+ 1315378  1.8  0.0  10772   116 pts/9    S    22:09   0:01 kasa_samoobslugowa
karpieł+ 1334911  0.0  0.0   13256  6412 pts/9    R+   22:11   0:00 ps -u
karpieł.kamil.155184@torus:~/SO_Projekty/Dyskont$ |

```

Zrzut 2: Poprawne skalowanie kas (3 procesy) i brak procesów zombie

```
--- STATUS KOLEJEK ---
Wspolna kolejka do kas stacjonarnych: 0 osob
Kolejka Kasa 1: 0 osob
Kolejka Kasa 2: 0 osob
Kasy samoobslugowe: 0 osob

--- STATUS KAS ---
Klienci w sklepie: 0
Kasy samoobslugowe czynne: 3/6
Kasa stacjonarna 1: ZAMKNIETA
Kasa stacjonarna 2: ZAMKNIETA
-----

=== PANEL KIEROWNIKA ===
1. Otworz kase stacjonarna 2
2. Zamknij kase stacjonarna 1
3. Zamknij kase stacjonarna 2
4. EWAKUACJA (SIGTERM)
5. Pokaz status kolejek
0. Wyjście
Wybor: |
```

Zrzut 3: Kierownik pokazuje 0 klientów w sklepie oraz 3 czynne kasy samoobsługowe

## Test 2 – Odporność na przepełnienie kolejek (VIP)

- **Polecenie:** `./dyskont.out 10000 1000 1`
- **Opis:** Test weryfikujący, czy system radzi sobie z lawinowym napływem komunikatów do kas samoobsługowych, gdzie kolejki mogą się przepełnić. Test sprawdza, czy mechanizm `WyslijKomunikatVIP` (zwiększanie limitu kolejki) pozwala zawsze odesłać odpowiedź.
- **Oczekiwany wynik:**
  1. Kasy samoobsługowe nie ulegają deadlockowi na operacji `msgsnd` mimo pełnej kolejki (blokujący `msgsnd` przechodzi, ponieważ komunikaty zwrotne VIP nie podlegają limitowaniu przez semafore).
  2. Wszystkie 10 000 klientów otrzymuje swoje paragony.
  3. Kierownik raportuje pełne kolejki (domyślnie 408 komunikatów [16384B / 40B - 1]), ale system nadal przetwarza klientów bez utraty danych.

```

--- STATUS KOLEJEK ---
Wspólna kolejka do kas stacjonarnych: 0 osob
Kolejka Kasa 1: 3 osob
Kolejka Kasa 2: 0 osob
Kasy samoobsługowe: 402 osob

--- STATUS KAS ---
Klienci w sklepie: 1000
Kasy samoobsługowe czynne: 6/6
Kasa stacjonarna 1: ZAJETA
Kasa stacjonarna 2: ZAMKNIETA
-----

=== PANEL KIEROWNIKA ===
1. Otworz kase stacjonarna 2
2. Zamknij kase stacjonarna 1
3. Zamknij kase stacjonarna 2
4. EWAKUACJA (SIGTERM)
5. Pokaz status kolejek
0. Wyjście
Wybor: |

```

Zrzut 1: Prawie pełne obłożenie kolejek (~403 komunikatów) a system nadal działa

```

karpziel.kamil.155184@torus:~/SO_Projekty/Dyskont/Logs$ grep -E "\(status: [0-1])" dyskонт_2026-01-26_21-44.log | wc -l
10000

```

Zrzut 2: Wszystkich 10 000 klientów zostało obsłużonych

### Test 3 – Ręczne zarządzanie kasami w szczycie (Sygnały)

- **Polecenie:** `./dyskont.out 10000 1000 0` + użycie `kierownik` (opcje 1-3)
- **Opis:** Weryfikacja stabilności systemu, gdy podczas obsługi 10 000 klientów. Kierownik dynamicznie otwiera i zamyka kasy stacjonarne. Sprawdza poprawność przekierowywania klientów przez wątek zarządcy kas w momencie nagłej zmiany stanu kasy.
- **Oczekiwany wynik:**
  1. Zamknięcie kasy (status `KASA_ZAMYKANA`) powoduje dokończenie obsługi wszystkich klientów z jej prywatnej kolejki, a następnie całkowite wyłączenie (`KASA_ZAMKNIETA`).
  2. Wątek Zarządcy natychmiast przestaje kierować nowych klientów do zamykanej kasy, przekierowując ich do drugiej kasy lub pozostawiając w kolejce wspólnej (gdzie oczekują np. na automatyczne otwarcie Kasy 1).
  3. Otwarcie kasy powoduje natychmiastowe przejście części ruchu i rozładowanie kolejki wspólnej tak, by klient wybrał najmniejszą kolejkę.
  4. System działa stabilnie – nie zawiesza się (deadlock) i nie kończy niespodziewanie błędem segmentacji (segfault) mimo intensywnych zmian konfiguracji kas.

```
--- STATUS KOLEJEK ---  
Wspolna kolejka do kas stacjonarnych: 0 osob  
Kolejka Kasa 1: 11 osob  
Kolejka Kasa 2: 0 osob  
Kasy samoobslugowe: 209 osob  
  
--- STATUS KAS ---  
Klienci w sklepie: 1000  
Kasy samoobslugowe czynne: 6/6  
Kasa stacjonarna 1: ZAJETA  
Kasa stacjonarna 2: ZAMKNIETA  
-----  
  
=== PANEL KIEROWNIKA ===  
1. Otworz kase stacjonarna 2  
2. Zamknij kase stacjonarna 1  
3. Zamknij kase stacjonarna 2  
4. EWAKUACJA (SIGTERM)  
5. Pokaz status kolejek  
0. Wyjście  
Wybor: |
```

Zrzut 1: Stan początkowy - stabilna praca, Kasa 1 obsługuje klientów

```
--- STATUS KOLEJEK ---  
Wspolna kolejka do kas stacjonarnych: 5 osob  
Kolejka Kasa 1: 28 osob  
Kolejka Kasa 2: 0 osob  
Kasy samoobslugowe: 402 osob  
  
--- STATUS KAS ---  
Klienci w sklepie: 1000  
Kasy samoobslugowe czynne: 6/6  
Kasa stacjonarna 1: ZAMYKANA  
Kasa stacjonarna 2: ZAMKNIETA  
-----  
  
=== PANEL KIEROWNIKA ===  
1. Otworz kase stacjonarna 2  
2. Zamknij kase stacjonarna 1  
3. Zamknij kase stacjonarna 2  
4. EWAKUACJA (SIGTERM)  
5. Pokaz status kolejek  
0. Wyjście  
Wybor:
```

Zrzut 2: Zamknięcie kas - Kasa 1 zamykana, klienci gromadzą się we wspólnej kolejce

```
--- STATUS KOLEJEK ---
Wspolna kolejka do kas stacjonarnych: 0 osob
Kolejka Kasa 1: 17 osob
Kolejka Kasa 2: 13 osob
Kasy samoobslugowe: 402 osob

--- STATUS KAS ---
Klienci w sklepie: 1000
Kasy samoobslugowe czynne: 6/6
Kasa stacjonarna 1: ZAMYKANA
Kasa stacjonarna 2: ZAJETA
-----

=== PANEL KIEROWNIKA ===
1. Otworz kase stacjonarna 2
2. Zamknij kase stacjonarna 1
3. Zamknij kase stacjonarna 2
4. EWAKUACJA (SIGTERM)
5. Pokaz status kolejek
0. Wyjscie
Wybor: |
```

Zrzut 3: Interwencja - otwarcie Kasy 2 powoduje rozładowanie zatoru z kolejki samoobsługowej

#### Test 4 – Ewakuacja przy maksymalnym obciążeniu

- **Polecenie:** `./dyskont.out 10000 1000 1` -> Wywołanie Ewakuacji (przez `kierownik`, `kill -SIGTERM` lub `CTRL + C`)
- **Opis:** Sprawdzenie czy system potrafi bezpiecznie i całkowicie posprzątać zasoby w momencie największego obciążenia (tysiące aktywnych procesów klientów i procesów obsługujących kasy).
- **Oczekiwany wynik:**
  1. Główny proces natychmiast wysyła `SIGTERM` do grupy procesów.
  2. Wszystkie procesy (3 managery + tysiące klientów) kończą się w ciągu kilku sekund.
  3. Polecenie `ipcs` wykazuje brak wiszących kolejek, semaforów i pamięci współdzielonej.
  4. Brak procesów zombie (wszystkie poprawnie odebrane przez `wait`).

```
[22:07:27] [INFO] Klient [PID: 1315090] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315091] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315092] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315093] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315094] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315095] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315096] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315097] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315098] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315099] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315100] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315101] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315102] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315103] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315104] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315105] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315106] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315107] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315108] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315109] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315110] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315111] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315112] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315113] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Klient [PID: 1315114] Ewakuacja (SIGTERM)
[22:07:27] [INFO] Kasy stacjonarne: Zakonczone wszystkie procesy potomne
[22:07:27] [INFO] Kasy stacjonarne: Zakonczone
[22:07:27] [INFO] Kasa samoobslugowa [PID: 1315173]: Ewakuacja (SIGTERM)
[22:07:27] [INFO] Kasa samoobslugowa [PID: 1305115]: Ewakuacja (SIGTERM)
[22:07:27] [INFO] Kasa samoobslugowa [PID: 1315172]: Ewakuacja (SIGTERM)
[22:07:27] [INFO] Kasa samoobslugowa [PID: 1305054]: Ewakuacja (SIGTERM)
[22:07:27] [INFO] Kasa samoobslugowa [PID: 1305053]: Ewakuacja (SIGTERM)
[22:07:27] [INFO] Kasy samoobslugowe: Zakonczone wszystkie procesy potomne.
[22:07:27] [INFO] Kasy samoobslugowe: Zakonczone.
[22:07:27] [INFO] Pracownik obslugi: Konczy dzialanie (SIGTERM)
[22:07:27] [INFO] Zwalnienie zasobow IPC
[22:07:27] [INFO] Usuniecie kolejki komunikatow
=== Koniec symulacji ===
karpiel.kamil.155184@torus:~/SO_Projekty/Dyskont$
```

Zrzut 1: Uruchomienie procedury ewakuacji (SIGTERM) przy pełnym obciążeniu sklepu

```
[22:07:27] [INFO] Kasy samoobslugowe: Zakonczone wszystkie procesy potomne.
[22:07:27] [INFO] Kasy samoobslugowe: Zakonczone.
[22:07:27] [INFO] Pracownik obslugi: Konczy dzialanie (SIGTERM)
[22:07:27] [INFO] Zwalnienie zasobow IPC
[22:07:27] [INFO] Usuniecie kolejki komunikatow
=== Koniec symulacji ===
karpiel.kamil.155184@torus:~/SO_Projekty/Dyskont$ ipcs | grep karpiel
karpiel.kamil.155184@torus:~/SO_Projekty/Dyskont$
```

Zrzut 2: Potwierdzenie czystego środowiska (brak wiszących zasobów IPC) po zakończeniu programu

## 8. Linki do kodu (Wymagane funkcje systemowe)

Poniżej znajdują się odniesienia do kluczowych mechanizmów systemowych wykorzystanych w projekcie:

### a. Tworzenie i obsługa plików (`open`, `close`, `write`)

- **open** (otwarcie pliku logów): [logi.c:37-41](#)
- **read** (odczyt z pliku/urządzenia): [semafory.c:17-18](#)
- **write** (zapis do logu): [logi.c:103](#)
- **close** (zamknięcie pliku): [logi.c:109](#)

#### **b. Tworzenie procesów (**fork**, **exec**, **exit**, **wait**)**

- **fork** (tworzenie procesu generatora klientów): [main.c:336-340](#)
- **exec** (**execl** - nadpisanie obrazu procesu): [main.c:348](#)
- **exit** (**\_exit** - zakończenie procesu potomnego): [main.c:352](#)
- **wait** (oczekiwanie na dzieci): [main.c:360](#)

#### **c. Tworzenie i obsługa wątków (**pthread\_create**, **pthread\_join**, **pthread\_exit**)**

- **pthread\_create** (utworzenie wątku loggera): [logi.c:130](#)
- **pthread\_join** (oczekiwanie na wątek sprzątający): [main.c:162](#)
- **pthread\_exit** (zakończenie wątku): [logi.c:110](#)
- **pthread\_sigmask** (blokowanie sygnałów w wątku): [logi.c:21](#)

#### **d. Obsługa sygnałów (**kill**, **signal**, **sigaction**)**

- **signal** (rejestracja handlerów sygnałów): [main.c:215-219](#)
- **sigaction** (rejestracja handlera w kierowniku): [kierownik.c:99-103](#)
- **kill** (wysłanie sygnału): [kierownik.c:129](#)

#### **e. Synchronizacja procesów (Semafor Systemu V)**

- **semget** (utworzenie/pobranie zestawu semaforów): [semafory.c:57](#)
- **semop** (operacja na semaforze - wait/signal): [semafory.c:41](#)
- **semctl** (ustawienie wartości/usunięcie): [semafory.c:110](#)

#### **f. Segmenty pamięci dzielonej (**shmget**, **shmat**, **shmdt**, **shmctl**)**

- **ftok** (generowanie unikalnego klucza IPC): [pamiec\\_wspoldzielona.c:172](#)
- **shmget** (alokacja segmentu pamięci): [pamiec\\_wspoldzielona.c:31](#)
- **shmat** (dołączenie segmentu do przestrzeni adresowej): [pamiec\\_wspoldzielona.c:38](#)
- **shmdt** (odłączenie segmentu): [pamiec\\_wspoldzielona.c:75](#)
- **shmctl** (usunięcie segmentu): [pamiec\\_wspoldzielona.c:85](#)

#### **g. Kolejki komunikatów (**msgget**, **msgsnd**, **msgrcv**, **msgctl**)**

- **msgget** (utworzenie kolejki): [kolejki.c:25](#)
- **msgsnd** (wysłanie komunikatu): [kolejki.c:54](#)
- **msgrcv** (odebranie komunikatu): [kolejki.c:69](#)
- **msgctl** (statystyki kolejki/usuwanie): [kolejki.c:85](#)