Raport

Spis treści

Informacje

Autor: Franciszek Pakos

Temat: 10 - Magazyn Firmy spedycyjnej

Środowisko

System operacyjny: Windows 11 // WSL2 // Ubuntu 24.04.3

Zarządzanie kompilacją: Makefile

Kompilator: GCC

Instrukcja uruchomienia Skompiluj projekt:

Bash make all Uruchom główny proces symulacji:

Bash ./main

Bash ./dispatcher Parametry uruchomieniowe Główny proces main automatycznie powołuje do życia procesy potomne, przekazując im odpowiednie argumenty funkcją execl:

Worker: Otrzymuje ID (1, 2 lub 3), co definiuje typ generowanych paczek (A, B lub C).

Truck: Uruchamiany bez argumentów, działa w pętli obsługi doku.

Fast Worker: Uruchamiany jako oddzielny proces dedykowany do paczek ekspresowych.

Losowość w systemie Zgodnie z wymaganiami, system wprowadza elementy losowe:

Waga paczek: Losowana w przedziale 0.1–25.0 kg (Workerzy) oraz 1.0–23.0 kg (Fast Worker).

Czas powrotu ciężarówki: Czas TI oraz losowe opóźnienia w pętli.

Architektura Systemu System opiera się na modelu wieloprocesowym. Komunikacja między procesami odbywa się poprzez mechanizmy IPC Systemu V.

Mechanizmy IPC i Synchronizacja Pamięć Współdzielona (Shared Memory):

Przechowuje strukturę Magazyn, zawierającą bufor cykliczny taśmy, liczniki wagi/objętości, PID-y procesów oraz flagi sterujące (koniec_pracy, p4_priorytet).

Semafory (Semaphores): Wykorzystano zestaw 4 semaforów do ścisłej synchronizacji:

SEM_MUTEX (0): Mutex binarny chroniący sekcję krytyczną (dostęp do pamięci współdzielonej).

SEM_WOLNE (1): Zlicza wolne miejsca na taśmie (blokuje producentów, gdy taśma pełna).

SEM_ZAJETE (2): Zlicza paczki gotowe do pobrania (blokuje konsumenta/ciężarówkę, gdy taśma pusta).

SEM_DOK (3): Semiprywatny semafor zapewniający, że w doku znajduje się tylko jedna ciężarówka naraz.

Opis Procesów Main (Zarządca) Inicjalizuje zasoby (tworzy IPC), uruchamia procesy potomne (fork + exec) i nadzoruje ich pracę (wait). Posiada handler sygnału SIGINT (Ctrl+C), który gwarantuje bezpieczne usunięcie semaforów i pamięci w przypadku przerwania programu.

Dyspozytor (Panel sterowania) Interfejs użytkownika pozwalający na wysyłanie sygnałów do procesów:

1 -> Wysyła SIGUSR1 do Trucka (wymuszony odjazd).

2 -> Wysyła SIGUSR2 do Fast Workera (zlecenie ekspresowe).

3 -> Ustawia flagę końca pracy w pamięci dzielonej.

Truck (Konsument) Proces reprezentujący ciężarówkę.

Pobiera paczki z taśmy zgodnie z zasadą FIFO.

Implementuje mechanizm ustępowania pierwszeństwa: Przed wejściem do sekcji krytycznej sprawdza flagę p4_priorytet. Jeśli jest ustawiona, zwalnia Mutex i usypia, pozwalając pracownikowi P4 załadować paczkę ekspresową poza kolejnością.

Odjeżdża po zapełnieniu (waga/objętość) lub na żądanie dyspozytora.

Pracownicy P1-P3 (Producenci) Symulują standardowych pracowników.

Każdy generuje specyficzny typ paczki (A, B, C).

Sprawdzają limity udźwigu taśmy przed położeniem towaru.

Działają w oparciu o semafory SEM_WOLNE i SEM_MUTEX.

Pracownik P4 (Priorytet) Obsługuje przesyłki ekspresowe.

Działa w trybie uśpienia, oczekując na sygnał SIGUSR2.

Po otrzymaniu sygnału ustawia flagę p4_priorytet w pamięci współdzielonej.

Omija kolejkę taśmy transportowej i ładuje paczkę bezpośrednio na ciężarówkę, wykorzystując mechanizm priorytetu zaimplementowany w procesie Truck.

Poniżej znajdują się bezpośrednie odnośniki do fragmentów kodu realizujących kluczowe wymagania systemowe:

1. **Tworzenie procesów (`fork`, `exec`, `wait`)**
    - Zarządzanie procesami potomnymi w `main.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/main.c#L101-L122]
2. **Pamięć Współdzielona (System V Shared Memory)**
    - Alokacja pamięci (`shmget`) w `main.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/main.c#L64-L68]
    - Dołączanie pamięci (`shmat`) w `worker.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/worker.c#L15-L25]
    - Usuwanie pamięci (`shmctl`) w `main.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/main.c#L131-L135]
3. **Semafory (System V Semaphores)**
    - Implementacja operacji P (`semop`) w `sem.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/sem.c#L4-L17]
    - Implementacja operacji V (`semop`) w `sem.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/sem.c#L20-L29]
    - Inicjalizacja wartości (`semctl`) w `main.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/main.c#L89-L93]
    - Synchronizacja dostępu (sekcja krytyczna) w `truck.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/truck.c#L31-L44]
4. **Sygnały (`signal`, `kill`)**
    - Wysyłanie sygnałów sterujących w `dispatcher.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/dispatcher.c#L37-L50]
    - Obsługa sygnału wymuszonego odjazdu w `truck.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/truck.c#L50-L55]
    - Bezpieczne zamykanie systemu (SIGINT) w `main.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/98b7d3149dfc86611062cb8e87470d75f314e054/main.c#L10-L35]
5. **Obsługa plików **
    - Logowanie zdarzeń do pliku w `sem.c`: [https://github.com/Frenky777/SO-2025-2026-PROJECT/blob/e6e692b96bfae4a54e92e745399d35a983a5c5ca/sem.c#L31-L60]

Scenariusze Testowe

Poniżej przedstawiam zestawienie przeprowadzonych testów weryfikujących poprawność działania systemu.

1. Przepełnienie bufora taśmy (Synchronizacja)

Test modelu Producent-Konsument.

- **Jak wykonać:** Uruchomienie symulacji ze standardowymi parametrami (ustawionymi w pliku nagłówkowym).
- **Cel:** Sprawdzenie, czy semafory poprawnie blokują producentów, gdy bufor jest pełny.
- **Wynik:** Mechanizm synchronizacji działa poprawnie. Taśma nie ulega przepełnieniu, a procesy czekają na zwolnienie miejsca (zgodnie z założeniami semaforów).

2. Priorytet paczki ekspresowej (P4)

Weryfikacja obsługi priorytetów.

- **Jak wykonać:** Wysłanie sygnału do procesu P4 (zgodnie z instrukcją: `sygnał 2`).
- **Cel:** Sprawdzenie, czy proces P4 potrafi wymusić pierwszeństwo dostępu do taśmy.
- **Wynik:** P4 skutecznie "wpycha się" w kolejkę. Po otrzymaniu sygnału proces ten uzyskuje dostęp do zasobów szybciej niż standardowi pracownicy.

3. Przekroczenie udźwigu taśmy

Test logiki biznesowej i ograniczeń fizycznych.

- **Jak wykonać:** Zmniejszenie parametru ładowności/udźwigu taśmy w konfiguracji.

- **Cel:** Upewnienie się, że suma wag paczek nie przekroczy dopuszczalnego limitu.
- **Wynik:** Logi potwierdzają działanie zabezpieczeń. Pracownicy wstrzymują się z załadunkiem, jeśli waga nowej paczki miałaby przekroczyć `MAX_WAGA_TASMY`.

4. Wymuszony odjazd ciężarówki

Obsługa przerwań i sygnałów niestandardowych.

- **Jak wykonać:** Wysłanie sygnału do Ciężarówki (zgodnie z instrukcją: `sygnał 1` / `SIGUSR1`).
- **Cel:** Sprawdzenie, czy ciężarówka odjedzie na żądanie, nawet jeśli nie jest pełna.
- **Wynik:** Ciężarówka reaguje natychmiastowo. Po otrzymaniu sygnału przerywa oczekiwanie (stan `sem_wait`) i odjeżdża z obecnym ładunkiem.

5. Bezpieczne zamykanie i czyszczenie zasobów

Test stabilności i zarządzania pamięcią (Graceful Shutdown).

- **Jak wykonać:** Wysłanie sygnału zakończenia (`sygnał 3` lub `CTRL+C`).
- **Cel:** Weryfikacja, czy system nie pozostawia wycieków pamięci ani "wiszących" semaforów.
- **Wynik:** Program kończy działanie w sposób kontrolowany. Pamięć współdzielona i semafory są poprawnie usuwane, a w systemie nie pozostają procesy zombie.

Problemy i rozwiązania

Projekt okazał się świetnym poligonem doświadczalnym, zwłaszcza w kwestii synchronizacji procesów. Oto lista błędów, które spędzały mi sen z powiek, i sposoby, w jakie je pokonałem:

1. **Zmora wiszących semaforów** Każda awaria programu zostawiała w systemie "sieroty" – zablokowane zasoby, przez które ponowne uruchomienie aplikacji sypało błędami.

   > **Fix:** Dodałem solidną obsługę `SIGINT`. Teraz program sprząta po sobie (usuwa semafory i pamięć) nawet po wciśnięciu `Ctrl+C`.

2. **Nadaktywny Fast Worker** Proces P4 był tak szybki, że nie dopuszczał innych do głosu (Mutexa). Ciężarówka stała w korku, bo Worker ciągle zajmował "skrzyżowanie".

   > **Fix:** Zaimplementowałem mechanizm "ustępowania pierwszeństwa". Jeśli Worker nie może pracować, musi jawnie oddać zasoby, dając szansę Ciężarówce.

3. **Chaos na taśmie ** Początkowo zdarzało się, że nowe paczki nadpisywały te nieodebrane, albo Ciężarówka ładowała powietrze.

   > **Fix:** Rozdzieliłem zadania semaforów. Jeden pilnuje tylko tego, czy jest miejsce (liczy sloty), a drugi chroni moment samego zapisu, żeby nikt sobie nie wchodził w drogę.

4. **Wyścigi w terminalu** Przy 5 procesach logi mieszały się tak, że nie dało się ustalić chronologii zdarzeń.

   > **Fix:** Każdy proces (np. Truck, Worker) ma swój kolor w terminalu, co pozwala jednym rzutem oka ocenić sytuację.

5. **Sygnały** Ciężarówka ignorowała polecenie odjazdu, jeśli akurat spała, czekając na semaforze.

   > **Fix:** Obsłużyłem przerwanie funkcji systemowej (`EINTR`). Teraz, gdy sygnał budzi proces, ten sprawdza flagę priorytetową zamiast wracać do spania.