

Raport - Symulator Tramwaju Wodnego

Przedmiot: Systemy Operacyjne

Projekt: Tramwaj wodny Kraków Wawel - Tyniec

Autor: Szymon Rafałowski

Repozytorium: <https://github.com/szyraf/tramwaj-wodny>

1. Założenia projektowe

1.1. Opis problemu

Symulacja tramwaju wodnego kursującego między przystankami Tyniec i Wawel. Program modeluje:

- Pasażerów czekających w kolejkach na obu przystankach
- Statek o ograniczonej pojemności (N osób, M rowerów)
- Mostek (trap) o ograniczonej przepustowości (K slotów)
- Kapitana zarządzającego fazami rejsu
- Dyspozytora obsługującego sygnały sterujące

1.2. Parametry konfiguracyjne

Parametr	Opis
N	Pojemność statku (osoby)
M	Pojemność statku (rowery)
K	Pojemność mostka
T1	Maksymalny czas załadunku (ms)
T2	Czas podróży (ms)
R	Liczba rejsów dziennie

1.3. Architektura wieloprocесowa

Program składa się z niezależnych procesów:

1. **Main** - proces główny, tworzy IPC i spawnuje procesy potomne
2. **Captain** - zarządza fazami: załadunek → podróż → rozładunek
3. **Dispatcher** - obsługuje sygnały z klawiatury (1=wcześniejszy odpływ, 2=koniec dnia)
4. **Passenger** (N procesów) - każdy pasażer to osobny proces

2. Ogólny opis kodu

2.1. Struktura plików

```
src/
├─ main.cpp          - Inicjalizacja IPC, fork() procesów
├─ captain.cpp       - Logika kapitana (fazy rejsu)
├─ passenger.cpp     - Automat stanowy pasażera
├─ dispatcher.cpp    - Obsługa klawiatury
├─ ipc.cpp/h         - Wrappery System V IPC
├─ config.cpp/h      - Parsowanie konfiguracji + walidacja
├─ logger.cpp/h      - Logowanie z timestampem
└─ common.h          - Struktury danych, enumy
```

2.2. Mechanizmy IPC

Program wykorzystuje **trzy mechanizmy IPC System V**:

1. **Pamięć współdzielona** - przechowuje stan symulacji (SharedState)
2. **Semafory** - synchronizacja (mutex + semafor per pasażer)
3. **Kolejki komunikatów** - potwierdzenia akcji (ACK)

2.3. Fazy symulacji

[illegible]

2.4. Stany pasażera

STATE_QUEUE → STATE_BRIDGE → STATE_SHIP → STATE_BRIDGE → STATE_EXITED

3. Co udało się zrobić

- Pełna symulacja z wieloma procesami (fork + exec)
- Synchronizacja przez semafor System V
- Komunikacja przez pamięć współdzieloną + kolejki komunikatów
- Obsługa dwóch sygnałów (Signal1, Signal2)
- Walidacja danych wejściowych (limity procesów, parametry)
- Obsługa błędów z perror()
- Minimalne prawa dostępu (0600)
- Sprzątanie zasobów IPC po zakończeniu
- Logowanie z timestampem do pliku i terminala
- Obsługa pasażerów z rowerami (zajmują 2 sloty na mostku)

4. Kluczowe decyzje projektowe

4.1. Ochrona sekcji krytycznych

Dostęp do `sharedState` wymaga synchronizacji - zastosowano semafor `SEM_MUTEX` jako mutex, aby tylko jeden proces mógł modyfikować współdzielone dane w danym momencie.

4.2. Mechanizm opróżniania mostka

Przed odpłynięciem statku pasażerowie pozostający na mostku muszą wrócić do kolejki.

Wprowadzono fazę `PHASE_BRIDGE_CLEAR`, w której kapitan aktywnie budzi pasażerów semaforami.

4.3. Obsługa przerwan systemowych (EINTR)

Sygnały (np. `SIGCHLD`) mogą przerwać wywołania `semop()` i `msgrcv()`. Operacje IPC są powtarzane w petli `while` gdy `errno == EINTR`.

4.4. Sprzątanie zasobów IPC

Zasoby System V IPC pozostają w systemie po zakończeniu procesu. Program obsługuje `SIGINT` / `SIGTERM` i wywołuje `cleanup_ipc()` również na starcie (usuwa pozostałości po poprzednim uruchomieniu).

5. Elementy specjalne

5.1. Dynamiczna liczba semaforów

Każdy pasażer ma dedykowany semafor do budzenia. Liczba semaforów = 2 + liczba_pasażerów.

5.2. Blokada pliku logów (`flock`)

Wieloprosesowy zapis do pliku logu z synchronizacją przez `flock()` .

5.3. Sprawdzanie limitu procesów

Przed startem program sprawdza `RLIMIT_NPROC` , aby nie przekroczyć limitu systemowego.

5.4. Timestampy względne

Logi pokazują czas od startu symulacji (nie czas systemowy).

6. Testy

Test 1: Podstawowy (`basic.env`)

Cel: Prosta symulacja z kilkoma pasażerami

Konfiguracja: N=5, M=2, K=3, T1=5s, T2=1s, R=2, 6 pasażerów

Wynik: PASS - wszyscy pasażerowie przewiezieni, 2 rejsy wykonane

Test 2: Pełny statek (full_ship.env)

Cel: Sprawdzenie limitu pojemności statku

Konfiguracja: N=3, K=2, R=3, 6 osób w Tyńcu

Wynik: PASS - statek nigdy nie przekracza 3 osob, osoby na mostku wracaja do kolejki

Test 3: Signal1 - wcześniejszy odpływ (signal1.env)

Cel: Wczesne odpłynięcie po naciśnięciu '1'

Konfiguracja: N=10, T1=30s, dużo pasażerów

Wynik: PASS - statek odpływa natychmiast po Signal1

Test 4: Signal2 - koniec dnia (signal2.env)

Cel: Zakończenie dnia po naciśnięciu '2'

Konfiguracja: N=10, R=10, dużo pasażerów

Wynik: PASS - symulacja konczy sie przed wykonaniem wszystkich rejsow

Test 5: Rowery (bikes.env)

Cel: Sprawdzenie limitu rowerów (M)

Konfiguracja: N=5, M=2, K=4, 2 osoby + 4 z rowerami

Wynik: PASS - max 2 rowery na statku

Test 6: Obciążeniowy (stress.env)

Cel: Stabilność przy dużej liczbie pasażerów

Konfiguracja: N=50, M=20, K=30, R=20, 300 pasażerów

Wynik: PASS - brak zawieszenia, program konczy sie normalnie

7. Linki do kodu (GitHub)

7.1. Tworzenie i obsługa plików

Funkcja	Lokalizacja
fopen()	logger.cpp:48

Funkcja	Lokalizacja
fclose()	logger.cpp:54
read()	dispatcher.cpp:37
flock()	logger.cpp:50-53

7.2. Tworzenie procesów

Funkcja	Lokalizacja
fork()	main.cpp:119 , main.cpp:128 , main.cpp:138
execl()	main.cpp:122 , main.cpp:131 , main.cpp:143
exit() / _exit()	main.cpp:34 , main.cpp:124
wait()	main.cpp:156
waitpid()	main.cpp:24

7.3. Obsługa sygnałów

Funkcja	Lokalizacja
signal()	main.cpp:111-113
kill()	main.cpp:30
Handler SIGINT/SIGTERM	main.cpp:27-35
Handler SIGCHLD	main.cpp:22-25

7.4. Synchronizacja procesów (semafory)

Funkcja	Lokalizacja
semget()	ipc.cpp:45 , ipc.cpp:54
semctl()	ipc.cpp:90 , ipc.cpp:97 , ipc.cpp:106
semop()	ipc.cpp:64 , ipc.cpp:73

Funkcja	Lokalizacja
Użycie mutex	captain.cpp:61 , passenger.cpp:110

7.5. Segmenty pamięci współdzielonej

Funkcja	Lokalizacja
<code>shmget()</code>	ipc.cpp:6 , ipc.cpp:15
<code>shmat()</code>	ipc.cpp:24
<code>shmdt()</code>	ipc.cpp:33
<code>shmctl()</code>	ipc.cpp:39 , main.cpp:13

7.6. Kolejki komunikatów

Funkcja	Lokalizacja
<code>msgget()</code>	ipc.cpp:112 , ipc.cpp:121
<code>msgsnd()</code>	ipc.cpp:133
<code>msgrcv()</code>	ipc.cpp:143
<code>msgctl()</code>	ipc.cpp:154 , main.cpp:19

7.7. Obsługa błędów i walidacja

Element	Lokalizacja
<code>perror()</code>	ipc.cpp:8 , ipc.cpp:17 , main.cpp:120
Walidacja konfiguracji	config.cpp:64-102
Sprawdzanie <code>RLIMIT_NPROC</code>	config.cpp:65-82
Obsługa <code>EINTR</code>	ipc.cpp:64-68

7.8. Podział na moduły

Moduł	Odpowiedzialność
main.cpp	Inicjalizacja, tworzenie procesów
captain.cpp	Logika kapitana
passenger.cpp	Automat stanowy pasażera
dispatcher.cpp	Obsługa klawiatury
ipc.cpp	Wrappery IPC
config.cpp	Konfiguracja
logger.cpp	Logowanie

8. Podsumowanie użytych konstrukcji

Wymaganie	Status	Opis
Tworzenie procesów	TAK	<code>fork()</code> , <code>execl()</code> , <code>wait()</code> , <code>waitpid()</code>
Mechanizmy synchronizacji	TAK	Semafore System V (<code>semget</code> , <code>semop</code> , <code>semctl</code>)
Dwa mechanizmy IPC	TAK	Pamięć współdzielona + kolejki komunikatów
Obsługa sygnałów	TAK	<code>SIGINT</code> , <code>SIGTERM</code> , <code>SIGCHLD</code> + <code>Signal1/Signal2</code>
Walidacja danych	TAK	<code>validate_config()</code> w <code>config.cpp</code>
Podział na moduły	TAK	7 plików źródłowych
Obsługa błędów	TAK	<code>perror()</code> , <code>errno</code> , obsługa <code>EINTR</code>
Minimalne prawa dostępu	TAK	0600 dla wszystkich zasobów IPC
Sprzątanie zasobów	TAK	<code>cleanup_ipc()</code> + obsługa sygnałów

9. Nieużyte konstrukcje

- **Wątki (pthread)** - projekt oparty w całości na procesach
- **Łącza (pipe, mkfifo)** - zastąpione kolejkami komunikatów
- **Gniazda (socket)** - nie wymagane w projekcie