

# Projekt Systemy Operacyjne - Tramwaj Wodny

**Autor** Paweł Drabik

**Numer albumu** 155171

**Repozytorium Github**

## 1. Opis projektu

W sezonie letnim po Wiśle na trasie Kraków Wawel – Tyniec kursuje tramwaj wodny o pojemności N pasażerów. Dodatkowo tramwajem można przewieźć M rowerów ( $M < N$ ). Statek z lądem jest połączony mostkiem o pojemności K ( $K < N$ ). Na statek próbują dostać się pasażerowie, z tym, że na statek nie może ich wejść więcej niż N, a wchodząc na statek na mostku nie może być ich równocześnie więcej niż K. Jeżeli na statek wchodzi osoba z rowerem na mostku zajmuje 2 miejsca. Statek co określoną ilość czasu T1 (np.: jedną godzinę) wypływa w rejs. W momencie odpływania kapitan statku musi dopilnować, aby na mostku nie było żadnego wchodzącego pasażera (pasażerowie, którzy nie weszli na statek muszą zejść z mostka począwszy od ostatniego w kolejce). Jednocześnie musi dopilnować by liczba pasażerów na statku nie przekroczyła N, a liczba rowerów M. Dodatkowo statek może odpływać przed czasem T1 w momencie otrzymania polecenia (sygnał1) od dyspozytora. Rejs trwa określoną ilość czasu równą T2. Po dotarciu do Tyńca pasażerowie opuszczają statek. Po opuszczeniu statku przez ostatniego pasażera, kolejni (inni) pasażerowie próbują dostać się na pokład w rejs powrotny (mostek jest na tyle wąski, że w danym momencie ruch może odbywać się tylko w jedną stronę). Statek może wykonać maksymalnie R rejsów w danym dniu lub przerwać ich wykonywanie po otrzymaniu polecenia (sygnał2) od dyspozytora (jeżeli to polecenie nastąpi podczas załadunku, statek nie wypływa w rejs, a pasażerowie opuszczają statek. Jeżeli polecenie dotrze do kapitana w trakcie rejsu statek kończy bieżący rejs normalnie – dopływa do przystanku w Tyńcu lub w Krakowie.

## 2 Założenia projektowe

### 2.1 Kompilacja

```
make clean  
  
make  
  
. /main
```

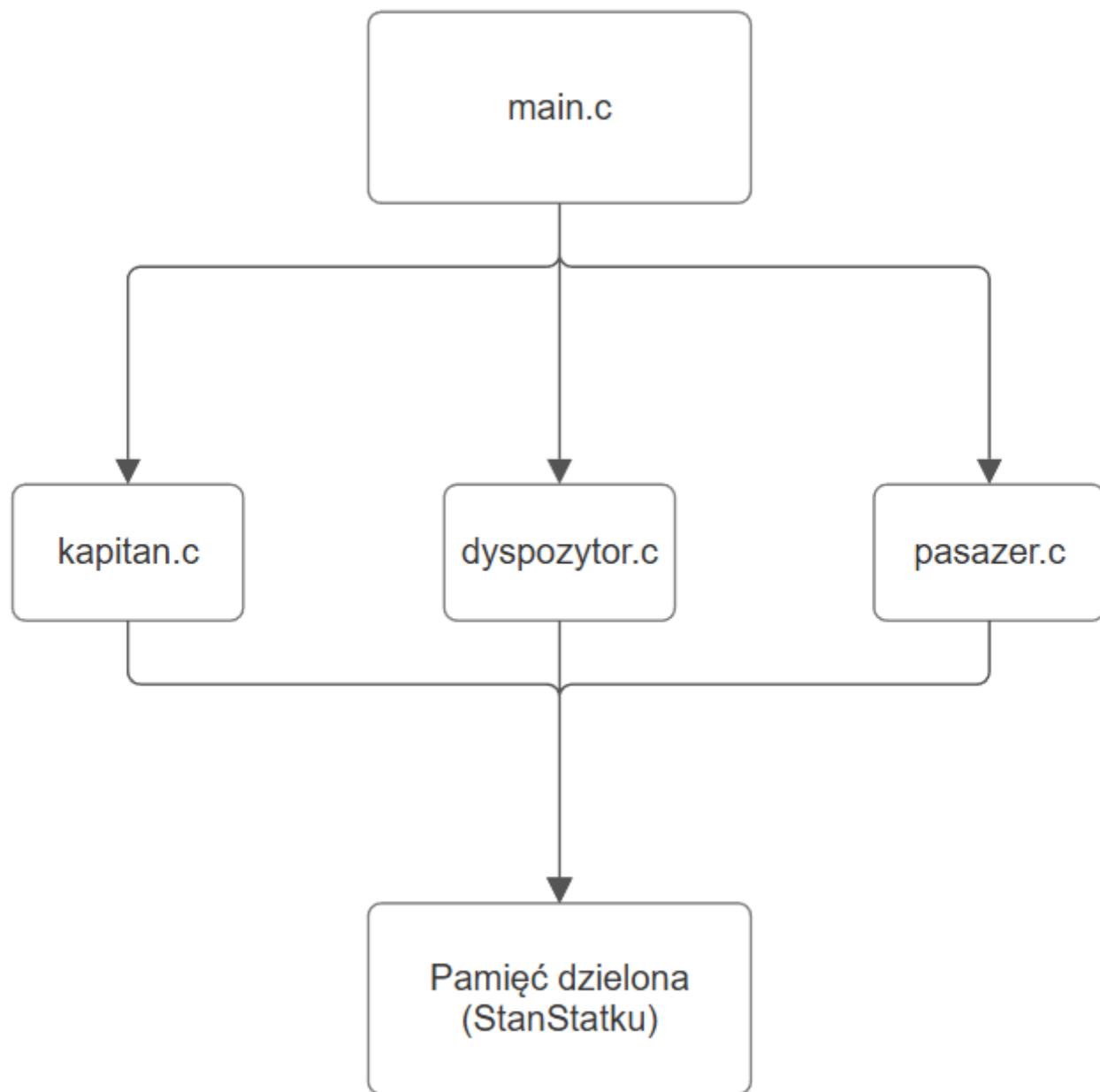
## 2.2 Założenia zadania

- Pasażerowie przybywają na przystanki Wawel lub Tyniec losowo (50% szans)
- Pasażer może mieć rower (30% szans)
- Na tramwaj może wejść maksymalnie N pasażerów
- Na mostku może znajdować się jednocześnie K pasażerów
- Pasażer z rowerem zajmuje 2 miejsca na mostku
- Kapitan musi dopilnować, aby na mostku nie było żadnego pasażera w momencie odpływu
- Pasażerowie obecni na mostku podczas odpływu muszą go opuścić zaczynając od ostatniego w kolejce
- Tramwaj odpływa co T1 czasu, lub po otrzymaniu sygnału od dyspozytora
- Rejs trwa T2 czasu
- Tramwaj może wykonać maksymalnie R rejsów w ciągu dnia lub przerwać ich wykonywanie po otrzymaniu sygnału od dyspozytora
- Tramwaj kursuje na trasie Kraków Wawel - Tyniec
- Mostek jest jednokierunkowy

## 2.3 Architektura systemu

System oparty jest na architekturze wieloprocesowej z następującymi procesami:

- Main - Proces główny - tworzy zasoby IPC oraz uruchamia pozostałe procesy
- Kapitan - Zarządza statkiem, załadunkiem oraz rozładunkiem pasażerów
- Dyspozytor - Monitoruje symulację oraz wysyła sygnały do kapitana
- Pasażer - Próbuje wejść na statek



## 2.4 Parametry systemu

Następujące parametry są wprowadzane przez użytkownika:

- **N** - Pojemność tramwaju (maksymalna liczba pasażerów)
- **M** - Liczba miejsc na rowery ( $M < N$ )
- **K** - Pojemność mostka ( $K < N$ )
- **T<sub>1</sub>** - Czas załadunku w sekundach
- **T<sub>2</sub>** - Czas rejsu w sekundach
- **R** - Maksymalna liczba rejsów

## 2.5 Walidacja danych

Wszystkie wprowadzane parametry są walidowane:

- Wartości muszą być większe od 0
- M musi być mniejsze od N
- K musi być mniejsze od N
- Nieprawidłowe dane powodują wyświetlenie komunikatu błędu i ponowne zapytanie

## 2.6 Pliki projektu

Plik	Opis
common.h	Definicje struktur, stałych, funkcji pomocniczych
logger.h	Nagłówek modułu logowania
logger.c	Implementacja systemu logowania
main.c	Proces główny - inicjalizacja i zarządzanie
kapitan.c	Logika kapitana statku
dyspozytor.c	Logika dyspozytora (wysyłanie sygnałów)
pasazer.c	Logika pojedynczego pasażera

## 3. Opis implementacji

### 3.1 Mechanizmy IPC

#### 3.1.1 Pamięć dzielona

Pamięć dzielona służy do przechowywania wspólnych stanów systemu

```
typedef struct {
    //Stany statku
    int pasazerowie_na_statku;
    int rowery_na_statku;
    int aktualny_przystanek;
    int liczba_rejsow;
    int koniec_symulacji;

    //Stan kapitana
    pid_t pid_kapitan;
    int status_kapitana;
    int zaladunek_owarty;
    int rejs_id;

    //Kolejka na mostku
    PasazerInfo kolejka_mostek[MAX_PASAZEROV_MOSTEK];
    int liczba_na_mostku;
```

```

int wypychanie_aktywne;

int miejsca_zajete_mostek;

//Lista wypchniętych w bieżącym cyklu
pid_t wypchnieci[MAX_PASAZEROV_MOSTEK];
int liczba_wypchniętych;

//Pasazerowie czekający na wejście na statek
int pasazerow_czekajacych_na_wejscie;

//Pasazerowie na statku do rozładunku
int pasazerow_do_rozladunku;

//Statystyki
int total_pasazerow_wawel;
int total_pasazerow_tyniec;
int pasazerow_odrzuconych;
} StanStatku;

```

### 3.1.2 Semaforы

System wykorzystuje 12 semaforów do synchronizacji:

Semafor	Init	Funkcja
SEM_MUTEX	1	Wzajemne wykluczanie przy dostępie do pamięci dzielonej
SEM_MOSTEK	K	Licznik miejsc na mostku
SEM_STATEK_LUDZIE	N	Licznik miejsc na statku
SEM_STATEK_ROWERY	M	Licznik miejsc na rowery
SEM_LOGGER	1	Synchronizacja dostępu do pliku log
SEM_ZALADUNEK_WAWEL	0	Sygnalizacja otwarcia załadunku na Wawelu
SEM_ZALADUNEK_TYNIEC	0	Sygnalizacja otwarcia załadunku w Tyńcu
SEM_WEJSCIE	0	Kapitan wpuszcza pasażerów na statek
SEM_ROZLADUNEK	0	Sygnalizacja rozpoczęcia rozładunku
SEM_ROZLADUNEK_KONIEC	0	Ostatni pasażer sygnalizuje zejście
SEM_DYSPOZYTOR_READY	0	Dyspozytor czeka na gotowość kapitana
SEM_DYSPOZYTOR_EVENT	0	Zdarzenia dla dyspozytora

### 3.1.3 Sygnały

System obsługuje sygnały:

Sygnal	Nadawca	Odbiorca	Działanie
SIGUSR1	Dyspozytor	Kapitan	Wcześniejszy odjazd przed czasem T1
SIGUSR2	Dyspozytor	Kapitan	Zakończenie pracy przed wykonaniem R rejsów
SIGUSR1	Kapitan	Pasażer	Wypychanie pasażera z mostka
SIGTERM	Main	Wszystkie	Zakończenie symulacji
SIGINT	System	Main	Przerwanie przez użytkownika

## 3.2 Algorytmy

### 3.2.1 Algorytm załadunku pasażerów

1. Kapitan otwiera załadunek (ustawia flage)
2. Sygnalizuje otwarcie załadunku semaforem N razy
3. Przez T1 czasu:
  - a. Sprawdza czy otrzymał sygnał do wcześniejszego odjazdu SIGUSR1
  - b. Wpuszcza pasażerów z mostka na statek
  - c. Sygnalizuje dyspozytorowi zdarzenie
4. Zamknie załadunek (czyści semafory)
5. Wypycha pasażerów pozostałych na mostku od ostatniego
6. Rozpoczyna rejs

### 3.2.2 Algorytm wchodzenia pasażera na statek

1. Pasażer czeka na semafor załadunku odpowiedni dla jego przystanku
2. Rezerwuje miejsce na statku
3. Jeżeli ma rower to rezerwuje miejsce na rower
4. Rezerwuje miejsce na mostku (1 lub 2 dla rowerzysty)
5. Dodaje się do kolejki na mostku
6. Czeka na semafor wejścia
7. Sprawdza czy nie został wypchnięty
8. Wchodzi na statek i zwalnia miejsce na mostku
9. Czeka na rozładunek
10. Schodzi ze statku

### 3.2.3 Algorytm wypychania pasażerów z mostka

1. Kapitan pobiera liczbę pasażerów na mostku
2. Od ostatniego:
  - a. Dodaje PID do listy wypchniętych
  - b. Wysyła SIGUSR1 do pasażera
  - c. Zwalnia miejsce na mostku
  - d. Zwalnia miejsce zarezerwowane na statku
  - e. Budzi pasażera
3. Zeruje kolejkę na mostku

---

## 4. Opis implementacji

### 4.1 Proces główny main.c

1. Wczytuje parametry od użytkownika z walidacją
2. Tworzy pamięć dzieloną i semafory
3. Uruchamia proces kapitana (fork() i exec())
4. Uruchamia proces dyspozytora (fork() i exec())
5. Generuje procesy pasażerów (10000 procesów)
6. Czeka na zakończenie kapitana i dyspozytora
7. Generuje raport końcowy
8. Czyści zasoby

### 4.2 Proces kapitana

Główna pętla kapitana:

1. Otwórz załadunek
2. Przez T1 czasu wpuszczaj pasażerów
3. Zamknij załadunek
4. Wypchnij pasażerów z mostka
5. Rozpocznij rejs i płyn przez T2 czasu
6. Dotrzyj do celu
7. Rozładuj pasażerów
8. Powtórz

### 4.3 Proces dyspozytora

1. Monitoruj przebieg symulacji
2. Wyślij sygnał do wcześniejszego odjazdu po 2 rejsach
3. Wyślij sygnał do końca pracy po 75% rejsów

### 4.4 Proces pasażera

1. Przybądź na losowy przystanek z rowerem lub bez
2. Czekaj na otwarcie załadunku

3. Rezerwuj miejsce na statku
  4. Rezerwuj miejsce na mostku
  5. Wejdź na mostek
  6. Czekaj na pozwolenie wejścia na statek
  7. Wejdź na statek
  8. Zwolnij miejsce na mostku
  9. Czekaj na rozładunek
  10. Zejdź ze statku
  11. Zwolnij miejsce na statku
  12. Koniec
- 

## 5. Sekcje krytyczne kodu

### 5.1 Dostęp do pamięci dzielonej

Każdy dostęp do pamięci dzielonej jest chroniony semaforem **SEM\_MUTEX**:

```
// common.h - Implementacja operacji semaforowych
static inline int sem_wait(int semid, int sem_num) {
    return sem_op(semid, sem_num, -1, 0); //czekaj
}

static inline int sem_signal(int semid, int sem_num) {
    return sem_op(semid, sem_num, 1, 0); //sygnalizuj
}

// Przykład użycia w pasazer.c - wejście na statek
sem_wait(semid, SEM_MUTEX); //POCZĄTEK SEKCJI KRYTYCZNEJ
usun_z_kolejki(wspolne, moj_pid);
wspolne->pasazerow_czekajacych_na_wejscie--;
wspolne->pasazerowie_na_statku++;
if (ma_rower) wspolne->rowery_na_statku++;
int na_statku = wspolne->pasazerowie_na_statku;
int rowerow = wspolne->rowery_na_statku;
sem_signal(semid, SEM_MUTEX); //KONIEC SEKCJI KRYTYCZNEJ
```

**Wyjaśnienie:** semafor **SEM\_MUTEX** jest inicjalizowany wartością 1. Operacja **sem\_wait** zmniejsza wartość semafora - jeśli wynik jest ujemny to proces jest blokowany. Operacja **sem\_signal** zwiększa wartość semafora i ewentualnie budzi oczekujący proces.

## 5.2 Zarządzanie kolejką na mostku

Dodawanie i odejmowanie pasażerów z kolejki na mostku jest kluczowe, aby zachować spójność:

```
// pasazer.c - Dodanie pasażera do kolejki na mostku
static int dodaj_do_kolejki(StanStatku *wspolne, pid_t pid, int ma_rower) {
    if (wspolne->liczba_na_mostku >= MAX_PASAZEROW_MOSTEK) {
        return -1;
    }

    int idx = wspolne->liczba_na_mostku;
    wspolne->kolejka_mostek[idx].pid = pid;
    wspolne->kolejka_mostek[idx].ma_rower = ma_rower;
    wspolne->kolejka_mostek[idx].rozmiar = ma_rower ? 2 : 1;
    wspolne->liczba_na_mostku++;
    wspolne->miejscza_zajete_mostek += (ma_rower ? 2 : 1);

    return idx;
}
```

**Wyjaśnienie:** funkcja ta jest zawsze wywoływana wewnątrz sekcji krytycznej chronionej przez SEM\_MUTEX, pozwala to na zapewnienie, że tylko jeden proces może modyfikować strukturę kolejki.

## 5.3 Wypychanie pasażerów z mostka

Algorytm wypychania pasażerów z mostka wymaga atomowej modyfikacji wielu zmiennych:

```
// kapitan.c - Wypychanie pasażerów
void wypchnij_z_mostka(int semid, StanStatku *wspolne) {
    int liczba = wspolne->liczba_na_mostku;

    if (liczba == 0) return;

    wspolne->wypychanie_aktywne = 1; // Flaga blokująca nowych pasażerów
    wspolne->liczba_wypchniętych = 0;

    // Wypychanie od ostatniego (LIFO)
    for (int i = liczba - 1; i >= 0; i--) {
        PasazerInfo *p = &wspolne->kolejka_mostek[i];

        // Dodaj do listy wypchniętych
        wspolne->wypchnieci[wspolne->liczba_wypchniętych++] = p->pid;

        // Wyślij sygnał do pasażera
        kill(p->pid, SIGUSR1);

        // Zwolnij zasoby
        sem_signal_n(semid, SEM_MOSTEK, p->rozmiar);
        wspolne->miejscza_zajete_mostek -= p->rozmiar;
        sem_signal(semid, SEM_STATEK_LUDZIE);
    }
}
```

```

    if (p->ma_rower) sem_signal(semid, SEM_STATEK_ROWERY);

    // Obudź pasażera
    sem_signal(semid, SEM_WEJSCIE);
}

wspolne->liczba_na_mostku = 0;
wspolne->miejsca_zajete_mostek = 0;
wspolne->wyptychanie_aktywne = 0;
}

```

### Wyjaśnienie:

- Flaga `wyptychanie_aktywne` zapobiega wchodzeniu nowych pasażerów podczas wypytychania
- Lista `wypchnieci[]` pozwala pasażerom sprawdzić czy zostali wypchnięci

## 5.4 Synchronizacja rozładunku

Ostatni pasażer schodzący ze statku powiadamia kapitana:

```

// pasazer.c - Schodzenie ze statku
sem_wait(semid, SEM_MUTEX);
wspolne->pasazerowie_na_statku--;
if (ma_rower) wspolne->rowery_na_statku--;
wspolne->pasazerow_do_rozladunku--;

int ostatni = (wspolne->pasazerow_do_rozladunku == 0); // Sprawdzenie atomowe
int nowy_przystanek = wspolne->aktualny_przystanek;
sem_signal(semid, SEM_MUTEX);

// ... zwalnianie zasobów ...

// Jeśli ostatni - powiadom kapitana
if (ostatni) {
    sem_signal(semid, SEM_ROZLADUNEK_KONIEC);
}

```

**Wyjaśnienie:** Sprawdzenie czy pasażer jest ostatni musi być atomowe i odbywać się wewnątrz sekcji krytycznej. W innym przypadku dwóch pasażerów mogłoby uznać się za ostatnich.

## 5.5 Obsługa sygnałów

Handlery sygnałów używają `volatile sig_atomic_t` dla bezpiecznej komunikacji:

```

// kapitan.c
static volatile sig_atomic_t wczesniejszy_odjazd = 0;
static volatile sig_atomic_t koniec_pracy = 0;

void sigusr1_handler(int sig) {

```

```
(void) sig;
wczesniejszy_odjazd = 1; // Atomowa operacja zapisu
}

void sigusr2_handler(int sig) {
    (void) sig;
    koniec_pracy = 1;
}

// W głównej pętli:
for (int sekunda = 0; sekunda < T1; sekunda++) {
    if (wczesniejszy_odjazd) { // Atomowy odczyt
        // Przerwij załadunek
        break;
    }
    // ...
}
```

## Wyjaśnienie:

- `volatile` zapobiega optymalizacji przez kompilator
- `sig_atomic_t` gwarantuje atomowy zapis/odczyt w kontekście obsługi sygnałów
- Handler sygnału może być wywołany asynchronicznie, więc musi być bezpieczny

---

## 6. Testy

### 6.1 Test nr. 1

Opuszczanie mostka od ostatniego w kolejce

Założenia: Dowolne parametry

Oczekiwanie: Pasażerowie będący na mostku po skończonym załadunku opuszczają go od ostatniego

Wynik: Sukces

```
[PASAŻER 710467] Wszedlem na mostek (pozycja: 4, mostek: 5/5)
[PASAŻER 710447] Wszedlem na STATEK (pasazerow: 6, rowerow: 1)
[PASAŻER 710493] Brak miejsca na rower - czekam na nastepny rejs
[PASAŻER 710468] Wszedlem na mostek (pozycja: 4, mostek: 5/5)
[PASAŻER 710463] Wszedlem na STATEK (pasazerow: 7, rowerow: 1)
[PASAŻER 710494] Brak miejsca na rower - czekam na nastepny rejs
[PASAŻER 710461] Wszedlem na STATEK (pasazerow: 8, rowerow: 2)
[PASAŻER 710502] Przybylem na przystanek Tyniec
[KAPITAN] Załadunek zakonczony: 8 pasazerow, 2 rowerow, 2 na mostku
[KAPITAN] Wypycham 2 pasazerow z mostka (od ostatniego)
[KAPITAN] Wypychasz pasazera PID:710468
[KAPITAN] Wypychasz pasazera PID:710467
```

## 6.2 Test nr. 2

Test sygnału do wcześniejszego odpłynięcia

Założenia:  $R > 4$

Oczekiwanie: Kapitan po otrzymaniu sygnału przerwie załadunek, wypchnie pasażerów z mostka od ostatniego i rozpoczęcie rejsu

Wynik: Sukces

**[DYSPOZYTOR] – SIGUSR1 – Wczesniejszy odjazd (po 2 rejsach)**

```
[PASAŻER 3263796]Przybyłem na przystanek Tyniec [ROWER]
[KAPITAN]SIGUSR1 – wcześniejszy odjazd po 3/5 s
[PASAŻER 3263799]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263798]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263800]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263801]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263807]Przybyłem na przystanek Krakow Wawel
[KAPITAN]Załadunek zakonczony: 2 pasazerow, 0 rowerow
[PASAŻER 3263802]Przybyłem na przystanek Tyniec
[PASAŻER 3263803]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263804]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263809]Przybyłem na przystanek Tyniec
[PASAŻER 3263806]Przybyłem na przystanek Krakow Wawel
[KAPITAN]Wypycham 3 pasazerow z mostka (od ostatniego)
[PASAŻER 3263808]Przybyłem na przystanek Tyniec
[PASAŻER 3263805]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263810]Przybyłem na przystanek Tyniec
[PASAŻER 3263811]Przybyłem na przystanek Tyniec [ROWER]
[PASAŻER 3263813]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263812]Przybyłem na przystanek Tyniec
[KAPITAN]Wypychasz pasazera PID:3263771
[PASAŻER 3263818]Przybyłem na przystanek Krakow Wawel
[PASAŻER 3263816]Przybyłem na przystanek Tyniec
[PASAŻER 3263814]Przybyłem na przystanek Tyniec [ROWER]
[PASAŻER 3263817]Przybyłem na przystanek Krakow Wawel [ROWER]
[KAPITAN]Wypychasz pasazera PID:3263759
[KAPITAN]Wypychasz pasazera PID:3263758
[PASAŻER 3263771]Zostalem wypchnienty – czekam na nastepny rejs
[PASAŻER 3263759]Zostalem wypchnienty – czekam na nastepny rejs
[KAPITAN]REJS #3 START: Krakow Wawel -> Tyniec (2 pasazerow, 0 rowerow)
```

## 6.3 Test nr. 3

Test sygnału do zakończenia pracy podczas rejsu

Założenia:  $R > 4$

Oczekiwanie: Dyspozytor wyśle sygnał do końca pracy podczas rejsu, a kapitan dokończy rejs i po rozładunku zakończy pracę

Wynik: Sukces

[DYSPOZYTOR]– SIGUSR2 – Koniec pracy (po 1 rejsach)

[PASAŻER 655182]Przybyłem na przystanek Tyniec

[KAPITAN]SIGUSR2 w trakcie rejsu – dokoncze rejs

[DYSPOZYTOR]Koniec pracy

[KAPITAN]SIGUSR2 w trakcie rejsu – dokoncze rejs

[KAPITAN]REJS #1 KONIEC – dotarliśmy do Tyniec

[KAPITAN]Rozpoczynam rozładunek

[KAPITAN]Rozładunek zakończony

[KAPITAN]Konczę prace po rejsie #1

[KAPITAN]Budzę oczekujących pasażerów...

[PASAŻER 655184]Przybyłem na przystanek Tyniec

Symulacja zakończona – przerywam generowanie pasażerów

#### 6.4 Test nr. 4

Test sygnału do zakończenia pracy podczas załadunku

Założenia: R > 4 oraz odkomentować w dyspozytor.c `&& status == STATUS_ZALADUNEK`

Oczekiwanie: Kapitan po otrzymaniu sygnału przerwie załadunek, a pasażerowie opuszczą statek, a następnie kapitan oraz dyspozytor zakończą pracę

Wynik: Sukces

[DYSPOZYTOR] - SIGUSR2 - Koniec pracy (po 5 rejsach)

[PASAVER 454679] Za późno - zaladunek zamkniety  
[PASAVER 454667] Wszedlem na mostek (pozycja: 3, mostek: 3/5)  
[PASAVER 454670] Wszedlem na mostek (pozycja: 4, mostek: 4/5)  
[PASAVER 454664] Wszedlem na STATEK (pasazerow: 6, rowerow: 1)  
[PASAVER 454649] Wszedlem na mostek (pozycja: 4, mostek: 4/5)  
[DYSPOZYTOR] Koniec pracy  
[PASAVER 454704] Przybylem na przystanek Tyniec [ROWER]  
[PASAVER 454705] Przybylem na przystanek Krakow Wawel  
[KAPITAN] Zaladunek zakonczony: 6 pasazerow, 1 rowerow, 4 na mostku  
[KAPITAN] Wypycham 4 pasazerow z mostka (od ostatniego)  
[KAPITAN] Wypychasz pasazera PID:454649  
[KAPITAN] Wypychasz pasazera PID:454670  
[PASAVER 454706] Przybylem na przystanek Krakow Wawel [ROWER]  
[KAPITAN] Wypychasz pasazera PID:454667  
[KAPITAN] Wypychasz pasazera PID:454655  
[PASAVER 454708] Przybylem na przystanek Tyniec  
[KAPITAN] Koniec rejsow na dzisiaj prosze opuscic statek  
[PASAVER 454684] Za późno - zaladunek zamkniety  
[PASAVER 454656] Schodze ze STATKU  
[PASAVER 454659] Schodze ze STATKU  
Symulacja zakonczona - przerywam generowanie pasazerow  
Wygenerowano 75 pasazerow

[PASAVER 454663] Schodze ze STATKU  
[PASAVER 454688] Za późno - zaladunek zamkniety  
[PASAVER 454692] Za późno - zaladunek zamkniety  
[PASAVER 454694] Za późno - zaladunek zamkniety  
[PASAVER 454674] Schodze ze STATKU  
[PASAVER 454664] Schodze ze STATKU  
[PASAVER 454696] Za późno - zaladunek zamkniety  
[PASAVER 454697] Za późno - zaladunek zamkniety  
[PASAVER 454645] Schodze ze STATKU  
[PASAVER 454649] Zostalem wypchnienty - czekam na nastepny rejs  
[PASAVER 454670] Zostalem wypchnienty - czekam na nastepny rejs  
[PASAVER 454667] Zostalem wypchnienty - czekam na nastepny rejs  
[PASAVER 454631] Za późno - zaladunek zamkniety  
[PASAVER 454644] Za późno - zaladunek zamkniety  
[PASAVER 454655] Zostalem wypchnienty - czekam na nastepny rejs

## 6.5 Test nr. 5

Test miejsc rowerów

Założenia: Miejsca na rowery (M) ustawione na 1

Oczekiwanie: Pasażer z rowerem próbuję wejść na statek na którym aktualnie znajduję się rower i wysyła komunikat Brak miejsca na rower - czekam na nastepny rejs

Wynik: Sukces

```
[KAPITAN] === REJS #5 – ZALADUNEK na Krakow Wawel ===  
[PASAER 3596385] Wszedlem na mostek (pozycja: 1, mostek: 2/5) [ROWER – 2 miejsca]  
[PASAER 3596446] Przybylem na przystanek Tyniec  
[PASAER 3596448] Przybylem na przystanek Tyniec  
[PASAER 3596387] Wszedlem na mostek (pozycja: 2, mostek: 3/5)  
[PASAER 3596449] Przybylem na przystanek Krakow Wawel  
[PASAER 3596447] Przybylem na przystanek Krakow Wawel [ROWER]  
[PASAER 3596385] Wszedlem na STATEK (pasazerow: 1, rowerow: 1)  
[PASAER 3596359] Brak miejsca na rower – czekam na nastepny rejs
```

## 6.6 Test nr. 6

Wszyscy pasażerowie z Wawelu

Założenia: Parametry dowolne, zakomentować w pasazer.c `int moj_przystanek = (rand() % 2) ? PRZYSTANEK_WAWEI : PRZYSTANEK_TYNIEC;` oraz odkomentować `int moj_przystanek = PRZYSTANEK_WAWEI;`

Oczekiwanie: W raporcie końcowym liczba pasażerów przewiezionych z Tyńca wyniesie 0

Wynik: Sukces

```
RAPORT KONCOWY  
Wygenerowano 10000 pasazerow  
Liczba rejsów: 7500/10000  
Pasażerowie przewiezieni:  
- z Wawelu: 10000  
- z Tynca: 0  
- lacznie: 10000  
Pasażerowie odrzuceni: 0
```

**Wszystkie testy przebiegły pomyślnie**

---

## 7. Co udało się zrealizować

### 7.1 Zrealizowane funkcjonalności

- Pełna implementacja symulacji tramwaju wodnego zgodna ze specyfikacją
- Wieloprocesowa architektura z użyciem fork() i exec()
- Synchronizacja procesów z użyciem semaforów
- Pamięć dzielona do wymiany danych między procesami
- Obsługa sygnałów SIGUSR1, SIGUSR2, SIGTERM, SIGINT
- Poprawne wypychanie pasażerów z mostka od ostatniego w kolejce
- Zarządzanie miejscami na rowery
- System logowania z synchronizacją dostępu
- Walidacja parametrów wejściowych
- Raport końcowy z statystykami
- Czyszczenie zasobów IPC przy zakończeniu

## 7.2 Elementy wyróżniające

- **Kolorowe wyjścia w terminalu** - procesy kapitana, dyspozytora oraz pasażera są wyraźnie rozróżnialne
  - **System logowania** - zdarzenia są zapisywane do pliku .log z timestampami
- 

## 8. Napotkane problemy

### 8.1 Procesy zombie

Duża liczba procesów pasażerów (10000) powodowała gromadzenie się zombie.

**Rozwiążanie:** Implementacja aktywnego zbierania zombie przez `waitpid(WNOHANG)` co 100 pasażerów oraz handler `SIGCHLD`.

### 8.2 Problem przy sprawdzaniu ostatniego pasażera

Dwóch pasażerów mogło uznać się za ostatniego schodzącego ze statku.

**Rozwiążanie:** Dodanie sprawdzenia `pasazerow_do_rozladunku == 0` do wnętrza sekcji krytycznej.

---

## 9. Linki do fragmentów kodu

### 9.1 Tworzenie i obsługa plików (`open()`, `close()`, `read()`, `write()`, `unlink()`)

Funkcja	Plik	Link
<code>fopen()</code> - otwarcie pliku log	logger.c	<a href="#">Link do kodu</a>
<code>fclose()</code> - zamknięcie pliku log	logger.c	<a href="#">Link do kodu</a>
<code>fprintf()</code> - zapis do pliku	logger.c	<a href="#">Link do kodu</a>
<code>unlink()</code> - usunięcie pliku	main.c	<a href="#">Link do kodu</a>
<code>fopen/fscanf/fclose</code> - plik konfiguracji	common.h	<a href="#">Link do kodu</a>

### 9.2 Tworzenie i obsługa procesów (`fork()`, `exec()`, `exit()`, `wait()`)

Funkcja	Plik	Link
<code>fork()</code> - tworzenie kapitana	main.c	<a href="#">Link do kodu</a>
<code>fork()</code> - tworzenie dyspozytora	main.c	<a href="#">Link do kodu</a>
<code>fork()</code> - tworzenie pasażerów	main.c	<a href="#">Link do kodu</a>
<code>exec()</code> - uruchomienie kapitana	main.c	<a href="#">Link do kodu</a>
<code>waitpid()</code> - czekanie na procesy	main.c	<a href="#">Link do kodu</a>
<code>exit()</code> - zakończenie procesu	pasazer.c	<a href="#">Link do kodu</a>

### 9.3 Obsługa sygnałów (kill(), signal(), sigaction())

Funkcja	Plik	Link
sigaction() - rejestracja handlerów	kapitan.c	<a href="#">Link do kodu</a>
kill() - wysyłanie SIGUSR1	dyspozytor.c	<a href="#">Link do kodu</a>
kill() - wysyłanie SIGUSR2	dyspozytor.c	<a href="#">Link do kodu</a>
kill() - wypychanie pasażera	kapitan.c	<a href="#">Link do kodu</a>
signal() - ignorowanie SIGTERM	main.c	<a href="#">Link do kodu</a>

### 9.4 Synchronizacja procesów (ftok(), semget(), semctl(), semop())

Funkcja	Plik	Link
ftok() - generowanie klucza	main.c	<a href="#">Link do kodu</a>
semget() - tworzenie semaforów	main.c	<a href="#">Link do kodu</a>
semctl(SETVAL) - inicjalizacja	main.c	<a href="#">Link do kodu</a>
semctl(IPC_RMID) - usuwanie	main.c	<a href="#">Link do kodu</a>
semop() - operacje P/V	common.h	<a href="#">Link do kodu</a>

### 9.5 Segenty pamięci dzielonej (ftok(), shmget(), shmat(), shmdt(), shmctl())

Funkcja	Plik	Link
shmget() - tworzenie segmentu	main.c	<a href="#">Link do kodu</a>
shmat() - przyłączenie segmentu	main.c	<a href="#">Link do kodu</a>
shmdt() - odłączenie segmentu	main.c	<a href="#">Link do kodu</a>
shmctl(IPC_RMID) - usuwanie	main.c	<a href="#">Link do kodu</a>

## 10. Struktura plików

```

tramwaj_wodny/
├── common.h          # Definicje struktur, stałych, funkcji pomocniczych
├── logger.h          # Nagłówek modułu logowania
├── logger.c          # Implementacja systemu logowania
├── main.c            # Proces główny
├── kapitan.c         # Proces kapitana
├── dyspozytor.c      # Proces dyspozytora
├── pasazer.c         # Proces pasażera
├── Makefile           # Plik komplikacji
├── README.md          # Dokumentacja projektu
├── img/
│   └── schemat.png    # Screenshoty z testów

```

```
└── test_mostek.png  
└── test_sigusr1.png  
└── test_sigusr2.png  
└── test_sigusr2_rejs.png  
└── test_rower.png  
└── tramwaj_wodny.log # Plik logu (generowany podczas działania)
```

---

## 11. Podsumowanie

Projekt tematu nr. 11 "Tramwaj wodny" zrealizowany zgodnie ze specyfikacją oraz wymaganiami projektowymi. Zaimplementowano wszystkie wymagane funkcjonalności tj.:

- Wieloprocesowa symulacja z użyciem fork() i exec()
- Synchronizacja za pomocą semaforów
- Komunikacja poprzez pamięć dzieloną
- Obsługa sygnałów
- Prawidłowe czyszczenie zasobów IPC