Anna Rotarska

Temat 12: Autobus podmiejski

1. Środowisko i narzędzia

System operacyjny: Windows 11 Pro

Środowisko uruchomieniowe: Torus Debian GNU/Linux 11

Język: C11

Kompilator: GCC

System budowania: Makefile

Edytor: Visual Studio Code

2. Budowa i uruchamianie

./sim_main <Nbus> <P> <R> <Tsec> <Mpass> <Ti_min> <Ti_max> [demo]

Nbus - liczba autobusów

P - liczba miejsc (osoby)

R - liczba miejsc na rowery

Tsec - czas bazowy

Mpass - ilu pasażerów ma zostać wygenerowanych

Ti_min, Ti_max - losowy czas powrotu autobusu

[demo] - dopisujemy, żeby włączyć tryb demo

Tryb demo (wolniej):

make demo

./sim_main 4 20 4 1 200 1 3 demo

Tryb test5000 (szybko, bez sleep):

make test5000

./sim_main 4 20 4 1 5000 1 3

3. Pokrycie wymagań

- Symulacja dworca i autobusów: pasażerowie przychodzą, kupują bilet w kasie, a następnie próbują wejść do autobusu.

- Różne typy pasażerów: VIP/normalni, dzieci (z opiekunem), pasażerowie z rowerem.

- Kasa biletowa: osobne kolejki VIP i NORMAL; kasa obsługuje priorytetowo VIP (gdy są obecni).

- Bilety: każdy pasażer wysyła żądanie do kasy i dostaje odpowiedź (OK/STOP) + numer biletu.

- Limity w autobusie:

  - P – maksymalna liczba osób (dziecko zajmuje 2 miejsca),

  - R – maksymalna liczba rowerów.

- Dwa wejścia do autobusu (pojemność 1 na wejście):

  - ENTR1 – wejście dla osób bez roweru,

  - ENTR2 – wejście dla osób z rowerem.

- Odjazd autobusu:

  - w trybie test5000 autobus odjeżdża automatycznie, gdy zapełni się do P,

  - dyspozytor może wymusić odjazd sygnałem (odjazd niepełny),

  - w DEMO autobus wraca po losowym czasie Ti z zakresu [Ti_min, Ti_max].

- Zamykanie wejść i bramki: w momencie odjazdu wejścia muszą być “puste”, a bramka (gate) blokuje nowe rezerwacje.

- STOP/Zakończenie:

  - dyspozytor sygnałem kończy symulację (STOP) i blokuje dalszą obsługę,

  - kasa dostaje komunikat STOP i przestaje sprzedawać,

  - logger kończy zapis i dopisuje podsumowanie.

- Logi i raport:

  - logi zdarzeń trafiają do procesu logger,

  - wynik zapisywany jest do raport.txt wraz z podsumowaniem na końcu.

4. Struktura kodu

- src/sim_main.c

Główny proces sterujący: tworzy IPC (kolejki, semafory, shm), uruchamia procesy (logger/kasa/dyspozytor/kierowca), generuje pasażerów, pilnuje zakończenia i sprzątania IPC.

- src/dyspozytor.c

Obsługa sygnałów sterujących:

- SIGUSR1 – wymuszenie odjazdu autobusu z niepełnym zapełnieniem (jeśli stoi na przystanku).

- SIGUSR2 – zakończenie symulacji (KONIEC/STOP), zablokowanie wejść i zatrzymanie procesów.

- src/kierowca.c

Logika autobusu: kontrola wejść (2 wejścia), odjazd, powrót autobusu, limity miejsc P i rowerów R, reakcja na sygnały STOP/KONIEC.

- src/kasa.c

Sprzedaż biletów: kolejki VIP i NORMAL, przydział numeru biletu, odpowiedź do pasażera przez msgrcv/msgsnd, aktualizacja statystyk.

- src/passenger_sim.c

Proces pasażera: losuje cechy (VIP/dziecko/rower), kupuje bilet, próbuje wejść przez właściwe wejście, rezerwuje miejsca/semafory, kończy przy STOP.

- src/logger.c

Zbiera logi z kolejki log_qid i zapisuje do raport.txt i na końcu dopisuje podsumowanie (kursy, liczba obsłużonych, VIP, dzieci, rowery, ostatni ticket).

- src/ipc.c + include/ipc.h

Generowanie kluczy IPC (ftok) dla kolejek/semaforów/shm.

- src/shm.c + include/shm.h

  Tworzenie/attach/detach pamięci dzielonej i inicjalizacja struktury station_state.

- src/sem_ipc.c + include/sem_ipc.h

  Tworzenie i obsługa semaforów (mutex, miejsca, rowery, wejścia, kolejki).

- src/kasa_ipc.c + include/kasa_ipc.h

  Struktury komunikatów i helpery do kolejek kasy (VIP/NORMAL).

- src/log_ipc.c + include/log_ipc.h

  Struktury komunikatów i helpery do logowania (kolejka loggera).

5. Sposób działania

IPC

Projekt używa trzech mechanizmów IPC:

* Pamięć dzielona SHM

W station_state trzymany jest wspólny stan symulacji: licznik kursów, flagi STOP/KONIEC, limity P i R, ile osób/rowerów jest aktualnie w autobusie, PID-y procesów, statystyki do raportu.

* Kolejki komunikatów (msg queues)

- Kasa ma dwie kolejki: VIP i NORMAL. Pasażer wysyła kasa_req, a kasa odsyła odpowiedź kasa_resp na mtype = pid pasażera (każdy odbiera swoją odpowiedź).

- Logger ma osobną kolejkę: kasa (i ewentualnie inne procesy) wysyła tekst logu log_msg, a logger zapisuje go do raport.txt.

* Semafory

Semafory kontrolują wejścia i limity (miejsca/rowery), kolejkę do kasy i bezpieczny dostęp do shm.

Najważniejsze z nich:

- SEM_MUTEX – mutex do sekcji krytycznych dla station_state.

- SEM_SEATS – ile jest wolnych miejsc w autobusie (limit P).

- SEM_BIKES – ile jest wolnych miejsc na rowery (limit R).

- SEM_ENTR1/SEM_ENTR2 – dwa wejścia do autobusu (każde przepuszcza 1 osobę naraz).

- SEM_QANY – bramka (gate): 0 = można rezerwować/wchodzić, 1 = zamknięta (odjazd).

- SEM_BOARDING – licznik trwających rezerwacji/wejść; kierowca czeka aż spadnie do zera przed odjazdem.

- SEM_KASA_ANY + SEM_QVIP/SEM_QNORM – sygnalizacja, że w kasie są zgłoszenia do obsłużenia.

PRZEPŁYW PASAŻERA

1) Losuje cechy: VIP/NORMAL, dziecko, rower.

2) Wysyła prośbę do kasy (kasa_req) na odpowiednią kolejkę (VIP albo NORMAL).

3) Kasa odsyła decyzję i numer biletu (kasa_resp).

4) Jeśli OK → pasażer próbuje wejść do autobusu:

- robi atomową rezerwację semaforów (miejsca + ewentualnie rower) oraz zwiększa SEM_BOARDING,

- przechodzi przez odpowiednie wejście (ENTR1 albo ENTR2),

- jeśli autobus nie odjeżdża i nie ma STOP → zwiększa w shm liczniki onboard i onboard_bikes,

- kończy rezerwację (SEM_BOARDING--).

5) Jeśli w trakcie jest STOP/KONIEC albo odjazd → rezerwacja jest wycofywana.

Dzięki temu nie da się przekroczyć limitów P i R, bo semafory ograniczają zasoby.

ZACHOWANIE KASY

1) Pasażerowie stoją w dwóch kolejkach (VIP i NORMAL).

2) Kasa w pierwszej kolejności próbuje obsłużyć VIP (gdy są zgłoszenia), inaczej NORMAL.

3) Dla każdego żądania przydziela kolejny numer biletu (next_ticket++) i aktualizuje statystyki.

4) Wysyła log do loggera co KASA_PRINT_EVERY (w trybie test5000 logi są rzadkie).

KIEROWCA I ODJAZDY AUTOBUSU

- Gdy autobus stoi na przystanku: wejścia są otwarte, gate (SEM_QANY) jest otwarty, semafory miejsc/rowerów mają wartości P i R.

- Odjazd:

1) kierowca zamyka gate (SEM_QANY = 1),

2) czeka, aż SEM_BOARDING == 0 (żeby nikt nie był w połowie wejścia),

3) zamyka wejścia (ENTR1=0, ENTR2=0),

4) zwiększa numer kursu i ustawia EV_ODJAZD,

5) w trybie demo autobus wraca po losowym czasie Ti (alarm), w test5000 powrót jest natychmiastowy (bez spania).

SYGNAŁY

- SIGUSR1 do dyspozytora: wymusza odjazd autobusu stojącego na przystanku (nawet jeśli nie jest pełny).

- SIGUSR2 do dyspozytora: KONIEC – ustawienie EV_KONIEC i stop=1, wysłanie STOP do kasy i loggera, zatrzymanie procesu kierowcy oraz pasażerów.

Po zakończeniu sim_main czeka na procesy potomne, a na końcu usuwa IPC (kolejki/semafory/shm). Logger dopisuje podsumowanie do raport.txt.

6. Logi i raport

Projekt zapisuje przebieg działania do pliku raport.txt przez osobny proces logger.

- Procesy (np. kasa) wysyłają tekstowe logi do kolejki loggera (log_qid).

- Logger zapisuje je linia po linii do raport.txt.

- Na końcu logger dopisuje podsumowanie:

SUMMARY:

- kursy – ile kursów wykonał autobus (licznik w shm),

- pasazerowie_obsluzeni – ilu pasażerów dostało odpowiedź z kasy (licznik żądań),

- vip, child, bike – statystyki typów pasażerów,

- ostatni_ticket – numer ostatniego biletu.

Częstotliwość logów zależy od trybu (w test5000 logi są ograniczone, żeby było szybko).

7. Testy

Test 1 - testuję czy przy zbyt dużej ilości pasażerów nadmiar ich wejdzie

Obsługa:
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/6bc10ee54e42dc9bf79349e7f5566c9a992bcbc8/passenger_sim.c#L97-L119

Limit miejsc jest wymuszany przez semafor SEM_SEATS - rezerwca odbywa się atomowo w reserve_begin_atomic()

Test 2 - testuję czy przy zbyt dużej ilości pasażerów z rowerami nadmiar ich wejdzie

Obsługa:
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/6bc10ee54e42dc9bf79349e7f5566c9a992bcbc8/passenger_sim.c#L97-L119 (gałąź else dla bike)

Limit rowerów jest wymuszany semaforem SEM_BIKES w reserve_begin_atomic(). Rezerwacja atomowa przed wejściem.

Test 3 - testuję czy na sygnał 1 od dyspozytora autobus odjedzie (nawet jeśli jest niepełny)

Dyspozytor każe odjechać:
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/dyspozytor.c#L118C9-L146C10

Kierowca reaguje i odjeżdża:
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/kierowca.c#L353C8-L373C10

Test 4 - testuję czy na sygnał 2 nikt nie wejdzie do autobusu

Dyspozytor:
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/kierowca.c#L353C8-L373C10

Kierowca:
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/kierowca.c#L353C8-L373C10

Pasażer nie wsiądzie bo sprawdza stop/event i robi rollback rezerwacji.

Test 5 - testuję czy autobus odjedzie co czas T

sim_main.c :
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/sim_main.c#L484C1-L496C7

8. Wymagane funkcje

1) Tworzenie i obsługa plików: fopen/fclose :
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/logger.c#L26C5-L71C15

2) Tworzenie procesów: fork, execv, waitpid :
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/sim_main.c#L86C1-L121C49

3) Kolejka komunikatów: msgget, msgsnd, msgctl, msgrcv :
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/kasa_ipc.c#L14C1-L101C2

https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/passenger_sim.c#L213C8-L213C94

4) Segmenty pamięci dzielonej: shmget, shmctl, shmat, shmdt :
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/shm.c#L16C1-L63C2

5) Obsługa sygnałów: sigaction, signal, kill :
https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/sim_main.c#L421C5-L562C14

6) Synchronizacja procesów: semget, semctl, semop
: https://github.com/Aaannniiiaa/so_projekt_rotarska/blob/2cc6e5a62d1ce8dc6e4056c359663b29e15ace3b/src/sem_ipc.c#L19C1-L170C2