Systemy Operacyjne
Zajęcia projektowe
Samolot z bagażami – raport

Autor: Norbert Szyszka

1. Uwagi ogólnie.

Wedle moich założeń, kluczowe było podzielenie kodu na kilka plików, każdy z nich odpowiadający za coś innego:

- dispatcher.h i dispatcher.c odpowiedzialne są za obsługę procesu dyspozytora,
- captain.h i captain.c odpowiedzialne są za obsługę procesów kapitanów samolotów,
- passenger.h i passenger.c odpowiedzialne są za obsługę procesów pasażerów,
- common.h i common.c to pliki, w których tworzę pamięć współdzieloną przez procesy,
- utilities.h i utilities.c to pliki, w których znajduje się kilka funkcji pomocniczych,
- main.c – tutaj inicjalizuję pamięć współdzieloną, tworzę odpowiednie procesy, inicjalizuję semafory i tworzę kolejkę wiadomości. Tutaj też pamięć dzielona jest odłączana i usuwana wraz z kolejką wiadomości, a semafory odpowiednio niszczone.

Największe problemy sprawiło mi zaimplementowanie funkcjonalności kończenia się programu w taki sposób, by samoloty mogły odlecieć z pasażerami już zboardowanymi. Nie udało mi się tego osiągnąć, jednak z pozostałych wymagań spełniona jest ich niemal całość, m.in.:

- implementacja maksymalnie 3 stanowisk, na którym może być maksymalnie dwóch pasażerów przy założeniu, że są oni tej samej płci,
- mechanizm frustracji pasażerów,
- wysyłanie próśb o wcześniejszy odlot jeżeli samolot jest pełny i zatwierdzanie ich,
- pasażerowie VIP,
- automatyczne zakończenie symulacji po wyczerpaniu się pasażerów na lotnisku.

2. Opis procedur.

common.c

1. `check_gender_and_set(int* currentGender, int slot, int gender)`:
    - Sprawdza płeć na danym stanowisku i ustawia ją, jeżeli natrafi na puste stanowisko.
    - Zwraca wartość 1, jeśli stanowisko jest puste lub jeżeli płeć nowego pasażera pasuje do płci pasażera na stanowisku.
    - Zwraca wartość 0, jeżeli płeć pasażera nie pasuje do płci pasażera na stanowisku kontroli.
2. `reset_gender(int* currentGender, int slot)`:
    - Resetuje płeć na wskazanym stanowisku aby wskazać, że stanowisko jest puste i nikogo nie obsługuje.

captain.c

1. `captain_process(int shmID, sem_t* semaphores, int gateID)`:
    - Reprezentuje proces kapitana.
    - Dołącza do pamięci współdzielonej i otwiera kolejkę komunikatów.
    - Sprawdza, czy samolot stojący przy bramce jest pełen i jeżeli jest, wysyła prośbę o wcześniejszy odlot do dyspozytora lotów.
    - Czeka na zatwierdzenie wcześniejszego odlotu przez dyspozytora, a następnie symuluje odlot i powrót samolotu na lotnisko.
    - Czeka na załadunek pasażerów, jeżeli samolot nie jest pełny.

passenger.c

1. `passenger_process(int shmID, sem_t* semaphores, int passengerID)`:
    - Reprezentuje proces pasażera.
    - Dołącza do pamięci współdzielonej i losuje wagę i płeć dla pasażera.
    - Symuluje kontrolę bagażu, a następnie dodaje pasażera do kolejki na kontrolę bezpieczeństwa.
    - Czeka w kolejce, przechodzi kontrolę bezpieczeństwa i czeka na dostęp do schodów.
    - Znajduje pierwszy samolot z wolnym miejscem i wchodzi na pokład, jeżeli ma dostęp do schodów.
    - Aktualizuje pamięć współdzieloną i zwalnia wykorzystywane przez siebie zasoby.
2. `verify_baggage(int weight, int isVip)`:
    - Weryfikuje, czy waga bagażu mieści się w wygenerowanym ustalonym limicie.
    - Zwraca wartość 1, jeżeli bagaż mieści się w limicie.
    - Zwraca wartość 0, jeżeli bagaż nie mieści się w limicie.

dispatcher.c

1. `dispatcher_process(int shmID, sem_t* semaphores)`:
    - Reprezentuje proces dyspozytora zarządzającego lotniskiem.
    - Dołącza do pamięci współdzielonej i tworzy kolejkę komunikatów.
    - Przez cały czas sprawdza stan samolotów i pasażerów.
    - Obsługuje prośby o wcześniejszy odlot wysyłane przez kapitanów i zatwierdza je.
    - Jeżeli żadni pasażerowie nie czekają na odlot, kończy symulację.

utilities.c

1. `generate_random_weight()`:
   - Generuje losową wagę bagażu pasażera o zadanym przedziale.
2. `generate_random_gender()`:
   - Generuje losową płeć dla pasażera (0 dla mężczyzny, 1 dla kobiety).

main.c

1. `main()`:
   - Funkcja główna, inicjalizuje zasoby, tworzy procesy dla wszystkich "członków" symulacji.
   - Czeka na zakończenie symulacji, a następnie czyści zasoby.
2. `signal_handler(int sig)`:
   - Obsługuje sygnał SIGINT, wykorzystywany do zakończenia symulacji.
   - Ustawia flagę zakończenia symulacji w pamięci współdzielonej i wysyła sygnał zakończenia do wszystkich procesów potomnych.
   - Czeka na zakończenie procesów potomnych i czyści zasoby.
3. `initialize_resources()`:
   - Inicjalizuje pamięć współdzieloną, semafory oraz kolejkę komunikatów.
   - Ustawia określone wartości początkowe w pamięci współdzielonej.
4. `cleanup_resources(int shmID, SharedData* sharedDataArg)`:
   - Czyści pamięć współdzieloną, semafory i kolejkę komunikatów.
   - Odłącza i usuwa pamięć współdzieloną.
   - Niszczy semafory i usuwa kolejkę komunikatów.

3. Wykorzystane testy.

1. Mało pasażerów, dużo samolotów – pasażerowie przechodzą odprawę, oczekiwanie w kolejce niemal nie zachodzi, kapitan nie wysyła próśb o wcześniejszy odlot – symulacja kończy się normalnie po odlocie pierwszego samolotu (co jest równoznaczne w tym przypadku z przewiezieniem wszystkich pasażerów), wszystko działa nominalnie.
2. Dużo pasażerów, mało samolotów – program trwa znacząco dłużej względem poprzedniego przypadku, oczekiwanie w kolejce trwa długo, kapitanowie wysyłają dużo próśb o wcześniejsze odloty, a symulacja kończy się po przewiezieniu wszystkich pasażerów.
3. Dużo pasażerów i dużo samolotów – Symulacja w „najnormalniejszym” ze swoich stanów – pasażerowie przechodzą odprawę i czekają na samoloty, które w stosunkowo „średnim” tempie są obsługiwane przez kapitanów i dyspozytora. Symulacja ponownie kończy się po przewiezieniu wszystkich pasażerów.
4. Użycie komendy `pkill airplane` - Symulacja kończy się natychmiast, pamięć zostaje zwolniona i semafory zostają zniszczone (nie dochodzi do błędu semaforów). Żaden proces nie zostaje procesem zombie, wszystkie procesy kończą się nominalnie.

4. Linki do najistotniejszych elementów kodu.

1. Wywołania `fork()`:
   https://github.com/norbert-szyszka15/so-projekt/blob/7fe535def0ce700e976b57fd29ba8ca538897c02/src/main.c#L33-L68
2. Otwarcie kolejki komunikatów i jej obsługa w wątku kapitania:
   https://github.com/norbert-szyszka15/so-projekt/blob/7fe535def0ce700e976b57fd29ba8ca538897c02/src/captain.c#L17-L46
3. Tworzenie kolejki wiadomości przez proces dyspozytora:
   https://github.com/norbert-szyszka15/so-projekt/blob/7fe535def0ce700e976b57fd29ba8ca538897c02/src/dispatcher.c#L19-L24
4. Obsługa i usuwanie kolejki wiadomości przez proces dyspozytora:
   https://github.com/norbert-szyszka15/so-projekt/blob/7fe535def0ce700e976b57fd29ba8ca538897c02/src/dispatcher.c#L48-L63
5. Funkcja do inicjalizacji pamięci współdzielonej:
   https://github.com/norbert-szyszka15/so-projekt/blob/7fe535def0ce700e976b57fd29ba8ca538897c02/src/main.c#L103-L153
6. Funkcja do czyszczenia zasobów po zakończeniu symulacji:
   https://github.com/norbert-szyszka15/so-projekt/blob/7fe535def0ce700e976b57fd29ba8ca538897c02/src/main.c#L155-L183