Raport projekt UL Adam Furtak

Ogólne założenia

Projekt w założeniu składa się z 4 rodzajów procesów, które komunikują się między sobą

1. hive
2. bee
3. queen
4. logger_server

Podczas gdy istnieje jeden proces hive i jeden proces queen, w jednym momencie istnieje wiele niezależnych procesów bee.

Proces hive podczas swojego startu tworzy procesy potomne queen i bee na podstawie pliku konfiguracyjnego, podanego jako parametr wywołania.

Pomiędzy procesem hive, queen oraz procesami bee, jest utworzony jest protokół komunikacji oparty o kolejki komunikatów, służący do koordynacji pracy ula w kontekście wchodzących i wychodzących pszczół przez poszczególne wejścia oraz składania jaj przez królową.

Ogólny szkic protokołu komunikacji wygląda następująco:

- każda z bram ula ma swój semafor, pilnujący dostępu do niego
- w momencie gdy pszczoła próbuje przejść przez bramę musi uzyskać do niej dostęp poprzez uzyskanie dostępu do semafora (pełniącego w tym przypadku rolę mutexa)
- Dodatkowo przed wejściem (oraz po wyjściu) zachodzi interakcja z innym semaforem, który pilnuje by w środku ula nie było zbyt wiele pszczół
- W momencie przejścia przez bramę pszczoła wysyła poprzez kolejkę komunikatów informację do ula, który odpowiada sygnałem ACK

Podobnie królowa, gdy chce złożyć jaja, najpierw upewnia się, że dla nowej pszczoły będzie miejsce w ulu. Nastepnie wysyła wiadomość kolejką komunikatów do ula by utworzyć nowy proces pszczoły

Bee trzyma informację o swoim stanie: czy znajduje się na zewnątrz ula, wewnątrz ula i tym podobne.

Nad wszystkim znajduje się biblioteka do logowania, która wysyła logi do serwera logowania, który następnie wypisuje je na standardowe wyjście lub przekierowuje do pliku. Komunikacja pomiędzy procesami a serwerem logowania odbywa się poprzez współdzieloną pamięć i jest koordynowana za pomocą semaforów. Dla wygody użytkownika (w tym przypadku programisty) dodano bibliotekę z wygodnym interfacem do wysyłania logów na serwer.

Obsługa błędów

Obsługa błędów jest delegowana do macra, który sprawdza rezultat wywołania danej funkcji i w wypadku wystąpienia błędu wywołuje funkcję czyszczącą oraz kończy proces z błędem. Proces hive natomiast słucha również, czy któryś z procesów dzieci nie zakończyło swojego działania z błędem. W takim wypadku odpala procedurę czyszczenia oraz zakończenia procesu.

Co udało się zrobić?

1. Zaimplementowano system logowania, pod warunkiem, że serwer logowania zostaje uruchomiony w odseparowaniu od pozostałych procesów. Pozostałe procesy wymagają uruchomionego procesu serwera logowania, w przeciwnym przypadku po zapełnieniu bufora, będą czekać na możliwość zapisania logów.
2. Zaimplementowano synchronizację dostępu do wspólnych zasobów pomiędzy procesami - wejścia do ula oraz miejsca w ulu.
3. Dodano mechanizm tworzenia nowych robotnic przez królową w porozumieniu z procesem ula pilnującym dostępu do miejsca w ulu
4. Zaimplementowano protokół komunikacji pomiędzy trzema rodzajami procesów, oparty na kolejce komunikatów odwzorowujący model działania klient i serwer z dodatkowym wysłaniem potwierdzenia do klientów.
5. Zaimplementowano współdzielone biblioteki ułatwiające implementację komunikacji pomiędzy procesami
6. Dodano walidację danych wejściowych w postaci pliku konfiguracyjnego odczytywanego przez proces hive
7. Zaimplementowano bezpieczną obsługę (async safe) sygnałów
8. Zaimplementowano

Z czym są problemy

1. Procesy zawieszają się w oczekiwaniu na wysłanie loga do serwera w przypadku gdy nie istnieje proces serwera logowania
2. Mechanizm wymiany informacji pomiędzy hive i bee oraz queen wchodzi w nieprawidłowy stan w przypadku gdy zakomentować wszystkie użycia sleep(x), z powodu na “zapychanie” się kolejki zdarzeń

Nie udało się zaimplementować

1. Dodania/usunięcia dodatkowych ramek do ula za pomocą sygnałów

Przeprowadzone testy

1. Zwykłe uruchomienie symulacji i sprawdzenie czy wszystkie mechanizmy komunikacji międzyprocesowej zostały odpowiednio wyczyszczone
2. Uruchomienie symulacji i wysłanie sygnału SIGINT

3. W trakcie działania symulacji zatrzymanie jej (ctrl + Z) i sprawdzenie czy nie powstają procesy zombie
4. Testowanie nieprawidłowych plików konfiguracyjnych
    a. przykładowy “zepsuty” plik configuracyjny znajduje się w repozytorium
5. Zakomentowanie “sleepów” w procesach queen i bee
    a. Zazwyczaj skutkuje wejściem symulacji w nieprawidłowy stan zakleszczenia
6. Uruchomienie symulacji bez serwera logowania
    a. symulacja będzie czekać na wczytanie logów przez serwer, po zapełnieniu dostępnego bufora
7. Sprawdzenie czy logi są wypisywane w odpowiedniej kolejności

Ważniejsze fragmenty kodu

1. wątek zbierający dzieci zombie w hive: https://github.com/furtka/ProjektSO/blob/main/src/hive.c#L105
2. wątek obsługujący bramę w ulu: https://github.com/furtka/ProjektSO/blob/main/src/hive.c#L120
3. parsowanie i walidacja pliku konfiguracyjnego: https://github.com/furtka/ProjektSO/blob/main/src/hive.c#L285
4. uruchomienie nowego procesu pszczoły: https://github.com/furtka/ProjektSO/blob/main/src/hive.c#L394C6-L394C24
5. Procedura czyszczenia w ulu: https://github.com/furtka/ProjektSO/blob/main/src/hive.c#L477
6. cykl życia królowej: https://github.com/furtka/ProjektSO/blob/main/src/queen.c#L45
7. Funkcja obsługująca cykl życia pszczoły: https://github.com/furtka/ProjektSO/blob/main/src/bee.c#L115
8. Wewnętrzny interfejs loggera: https://github.com/furtka/ProjektSO/blob/main/src/logger/logger_internal.h
9. Implementacja funkcji klienckiej loggera: https://github.com/furtka/ProjektSO/blob/main/src/logger/logger.c#L26
10. makefile: https://github.com/furtka/ProjektSO/blob/main/Makefile