Winda

Autorzy: Kamil Sikora i Konrad Kalita

Cel programu

Celem jest symulacja działania pojedynczej windy. Użytkownik może zadawać rozkazy dla windy, zarówno odtwarzając sytuację przebywania wewnątrz windy jak i sytuację przywoływania windy na konkretne piętro z możliwością wyboru docelowego kierunku góra/dół. Użytkownik ma możliwość również wybrania ilości pięter, po których winda ma się poruszać, zadawania kolejnych operacji dla windy, jak i wyjścia z programu.

Opis struktury programu

```
main() ->
    Elevator_Pid = spawn(winda, run, [0]),
    Max_Floor_Pid = spawn(winda, max_floor, []),
    Error_Handler_Pid = spawn(winda, on_error, []),
    Listener_Pid = spawn(winda, listen, []),
    External_Pid = spawn(winda, external_manage, []),
    Internal_Pid = spawn(winda, internal_manage, []),
```

.

W procedurze głównej tworzymy 6 procesów komunikujących się ze sobą.
Elevator_Pid - służy do uruchamiania procesu dla windy. W nim znajduje się główna funkcja odpowiadająca za przetwarzanie kolejnych danych przez urządzenie windy. Opis niżej.

Max_Floor_Pid proces pobiera od użytkownika maksymalną ilość pięter i zapisuje ją do zmiennej ets. Następnie wywoływana jest funkcja odpowiedzialna za przechwycenie pięter zadanych przez użytkownika. W tej części pojawia się także obsługa błędów, gdy błąd wystąpi odpowiedni komunikat jest wysyłany do procesu Error_Handler_Pid.

```
max_floor() ->
    [{error_pid, Error_Handler_Pid}] = ets:lookup(pids, error_pid),

    receive
        {choose_max_floor} ->
            clear(),
            Floors_Number_Input = io:read("\nPodaj liczbe pieter \n Input: "),
            io:write(Floors_Number_Input),

             case Floors_Number_Input of
                 {ok,Max_Floor} ->
                    if
                        Max_Floor < 1 ->
                             Error_Handler_Pid ! {wrong_max_floor_input};
                        true->
                            ets:insert(max_floor_var, {Max_Floor, Max_Floor}),
                            handle_calls_input(0)
                    end;
                 {error, _} ->
                     Error_Handler_Pid ! {wrong_max_floor_input},
                     max_floor()
             end
    end.
```

Proces Error_Handler_Pid odpowiada za przechwytywanie błędów aplikacji i odpowiedniej reakcji na nie.

Proces Listener_Pid nasłuchuje pewne akcje użytkownika oraz wydarzenia w działaniu programu i podejmuje odpowiednią dla nich akcję lub przekazuje je dalej.

Procesy External_Pid i Internal_Pid przechwytują rozkazy dla odpowiednio wezwań z zewnątrz i wewnątrz windy. Dzięki rozróżnieniu komunikatów oraz zajściu odpowiednich warunków porównujących przekazany rozkaz z aktualną pozycją windy, aplikacja umieszcza rozkaz użytkownika w odpowiedniej zmiennej ets w celu dalszego wykorzystania przez program.

```
internal_manage()->
    receive
        {internal_call, Floor, Actual_Floor} ->
            if
                Floor > Actual_Floor ->
                    ets:insert(elevator_stops_up,{Floor,Floor});
                Floor < Actual_Floor ->
                    ets:insert(elevator_stops_down,{Floor,Floor})
            end,
            internal_manage()
    end.
```

```
external_manage()->
    receive
        {external_call, Floor, 1} ->
            ets:insert(elevator_stops_up,{Floor,Floor}),
            external_manage();
        {external_call, Floor, -1} ->
            ets:insert(elevator_stops_down,{Floor,Floor}),
            external_manage()
    end.
```

Funkcja handle_calls_input najpierw wysyła odpowiedni komunikat w celu rozpoczęcia działania windy, następnie przechwytuje rozkazy użytkownika i przakazuje je do odpowiedniej funkcji przetwarzającej je. Tu pojawia się także komuniat dla Procesu Listener_Pid, gdyby użytkownik zakończył działanie programu. Wywołuje ona funkcje handle_calls, przekazując Input użytkownika oraz aktualną pozycję windy.

Funkcja handle_calls wykonuje się rekurencyjnie w każdym wywołaniu przetwarzając jedna piętro i przesyłając komunikat do odpowiedniego procesu External_Pid/Internal_Pid

Funkcja on_error należy do procesu Error_Handler_Pid i podejmuje odpowiednie działanie po otrzymaniu konkretnego komunikatu.

Funkcja run oraz run_helper **TODO KONRAD KALITA**

Sekcja ELEVATOR STOPS UTILS obejmuje drobne funkcje usprawniające działanie programu oraz wykonujące działania z modułem ets oraz listami aby łatwo zarządzać windą.

Instrukcja obsługi

Kompilacja: c(winda).

Uruchomienie: winda:main().

Na starcie pojawia się informacja o programie oraz jesteśmy proszeni o podanie maksymalnej ilości pięter dla naszej windy, **podaną liczbę należy wpisać z końcową kropką.** Przykład:

```
Podaj liczbe pieter
 Input: 10.
```

Następnie podajemy rozkazy dla windy, instrukcja znajduje się także w programie.

```
Podaj wezwania dla windy w formacie listy krotek {pietro,kierunek},
      1 oznacza kierunek w gore, 0 wezwanie wewnatrz windy, -1 kierunek w dol,

      po wykonaniu dzialania windy bedziesz mogl wprowadzic kolejne wejscie

      Przyklad: [{3,1},{7,-1},{8,0}].

      Aby wyjsc z programu napisz exit.
      Input: [{5,1},{4,0},{2,1},{9,0},{6,-1}].
```

Należy pamiętać o końcowej kropce.
Program przetwarza rozkazy i wykonuje działanie, zatrzymując windę na odpowiednich piętrach w zoptymalizowanej kolejności. Dla poniższego przykładu chociaż piętro 2 zostało wprowadzone po 5 to winda reaguje najpierw na to wezwanie. Stop na 6 przetwarza na samym końcu, mimo mijania tego piętra w trakcie, gdyż został wybrany kierunek w dół. Output dla powyższego przykładu.

```
3>
 Floor: 1
3>
 Floor: 2
3>
 STOP
3>
 Floor: 3
3>
 Floor: 4
3>
 STOP
3>
 Floor: 5
3>
 STOP
3>
```

```
 Floor: 6
3>
 Floor: 7
3>
 Floor: 8
3>
 Floor: 9
3>
 STOP
3>
 Floor: 8
3>
 Floor: 7
3>
 Floor: 6
3>
 STOP
```

Następnie możemy zakończyć działanie programu wpisując **exit.** lub wprowadzić kolejne rozkazy, winda rozpocznie prace od piętra, gdzie ostatnio zakończyła.

Dodatkowe informacje, ograniczenia i możliwości rozbudowy

Mimo przetestowania programu nie wykluczamy wystąpienia błędu w wyświetlaniu komunikatów dla użytkownika na innych systemach bądź urządzeniach, związanych z modułem io.

Dalszy rozwój programu może się opierać na zwiększeniu ilości pracujących wind, w celu większej optymalizacji i szybszej obsługi wirtualnego użytkownika windy.