Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

PROJEKT SEMESTRALNY PWIR

ANNA GNOIŃSKA, JULIA IGNACYK

PROGRAM MODELUJĄCY FUNKCJONOWANIE DZIEKANATU

Kraków 2020

Spis treści

1. Wprowadzenie

1.1. Cel projektu

Celem projektu jest utworzenie programu symulującego działanie dziekanatu od strony obsługi studentów. Symulator został napisany w języku Erlang, który zapewnił współbieżne działanie instytucji. Poprzez uwzględnienie godzin pracy dziekanatu oraz dostępnych pracowników zostają przyjmowani studenci, którzy odwiedzają dziekanat w różnych celach.

2. Opis programu

2.1. Procesy współbieżne

Erlang został zaprojektowany z myślą o zastosowaniach w programach implementujących współbieżność. Symulator dziekanatu opracowany na potrzeby projektu opiera się na współbieżności procesów. Pierwszy proces związany jest z kolejką FIFO, na której działa cała logika programu. To jego tworzymy na początku, a następnie przechodzimy do metod *createStudents()* i *createWorker()*. W metodzie *createWorker()* wraz z tworzeniem nowego pracownika tworzymy nowy proces. Funkcja *createStudents()* tworzy jeden proces, w którym odbywa się inicjalizacja studentów i dodawanie ich do kolejki.

```
QueuePid = spawn(students_queue,
                 initQueue,
                 [WorkersNumber]),
students:createStudents(QueuePid),
timer:sleep(100),
office_worker:createWorker(QueuePid,
                           Start,
                           End,
                           WorkersNumber).
```

Wynik śledzenia procesów i ich PID'ów w programie:

```
5> main:deansOffice().
QueuePid <0.165.0>
Dziekanat otwarty. Zapraszamy!
StudentPid <0.166.0>
WorkerPid <0.167.0>
WorkerPid <0.168.0>
WorkerPid <0.169.0>
WorkerPid <0.170.0>
WorkerPid <0.171.0>
ok
```

2.2. Wykorzystane struktury

W naszej symulacji została wykorzystana kolejka FIFO. Odpowiada ona rzeczywistej kolejce studentów oczekujących na swoją kolej w dziekanacie. Zaraz po utworzeniu studenta dodajemy go do kolejki - *pushToFifo()*, gdy załatwi on swoją sprawę w dziekanacie poprzez funkcje *popFromFifo()* zostaje usunięty z kolejki. W trakcie działania symulacji często korzystamy z metody *notEmpty()*.

```
-module(fifo).

-export([newFifo/0,
         popFromFifo/1,
         pushToFifo/2,
         notEmpty/1]).

newFifo() -> {[], []}.

pushToFifo({In, Out}, X) -> {[X | In], Out}.

popFromFifo({[], []}) -> erlang:error(empty);
popFromFifo({In, []}) ->
    popFromFifo({[], lists:reverse(In)});
popFromFifo({In, [H | T]}) -> {H, {In, T}}.

notEmpty({[], []}) -> false;
notEmpty({_, _}) -> true.
```

2.3. Opis struktury zadaniowej programu

1. **main.erl** - główny plik z procedurą textbfdeansOffice(), która odpowiada za rozpoczęcie symulacji wywołując poszczególne metody. W tym pliku można zmieniać czas pracy dziekanatu oraz liczbę pracowników obsługujących studentów.

2. **fifo.erl** - plik zawierający implementację kolejki FIFO. Posiada funkcje:

   – *newFifo()* - tworzy nową kolejkę

   – *pushToFifo()* - dodaje element (w naszym przypadku studenta) do kolejki

   – *popFromFifo()* - pobiera pierwszy element z kolejki

   – *notEmpty()* - sprawdza, czy kolejka jest pusta (czy wszyscy studenci zostali obsłużeni)

3. **office_worker.erl** - plik z funkcją odpowiadającą za tworzenie pracowników, a także zawierający implementację funkcji przydzielającej zadania pracownikowi w czasie jego pracy.

   – *initWorker()* - przechwytuje pracownika tworzonego w *createWorker()*

   – *workerName()* - funkcja nadająca pracownikowi jego ”imię”

   – *randomBreakReason()* - funkcja generująca powód przerwy będącej wynikiem tego, że kolejka jest pusta

   – *workerTasks()* - funkcja odpowiadająca za zarządzanie czasem pracy oraz obsługą zadań pracownika. Jeśli jego czas pracy się skończył - pracownik idzie do domu, w innym przypadku pracownik obsługuje studenta lub ma przerwę

   – *createWorker()* - odpowiada za utworzenie procesu dla każdego pracownika oraz zaincjalizowanie go

4. **request_processing.erl** - plik zawierający implementację funkcji, która jest odpowiedzialna za przetwarzanie żądania studenta.
   - *requestProcessing()* - funkcja w zależności od przypadku, z jakim student się zjawił przetwarza żądanie i obsługuje określoną sprawę
5. **students.erl** - plik z funkcją odpowiedzialną za tworzenie studentów i dodawanie ich do kolejki. Jeśli dziekanat zostanie zakmnięty, oczekujący w kolejce nie zostaną obsłużeni.
   - *createStudents()* - funkcja, która tworzy proces odpowiedzialny za tworzenie studentów
   - *initStudents()* - przechwytuje proces tworzony w textitcreateStudents(), a następnie rozpoczyna nasłuchiwanie
   - *randomNames()* - funkcja nadająca studentowi jego imię
   - *randomRequest()* - funkcja generująca losową sprawę z jaką student kieruje się do dziekanatu
   - *randomStudent()* - tworzy studenta z losowym imieniem oraz losową sprawą
   - *listen()* - odpowiedzialna za wywoływanie tworzenia nowych studentów, po dostaniu odpowiedzi tworzy nowego studenta lub w przypadku zamknięcia dziekanatu kończy swoją pracę
6. **students_queque.erl** - plik odpowiedzialny za obsługę kolejki.
   - *initQueue()* - funkcja tworząca kolejkę i rozpoczynająca jej obsługę
   - *clearQueue()* - metoda, która po zakończeniu pracy dziekanatu ”odsyła” studentów, którzy nie skończyli, a także kończy pracę dziekanatu
   - *listen()* - funkcja kierująca logiką programu, to tutaj do kolejki są dodawani nowi studenci oraz sprawy są rozpatrywane. W razie zakończenia czasu pracy pracownika dziekanat jest zamykany a kolejka czyszczona

3. Obsługa symulatora

3.1. Instrukcja

Program złożony jest z sześciu plików. W celu uruchomienia symulacji należy skompilować wszystkie pliki. Można posłużyć się procedurą *cover:compile_directory()*, która przeprowadza kompilacje wszystkich plików znajdujących się w folderze. Program uruchamiamy poprzez wywołanie procedury *main:deansOffice()*. Aby zmienić godziny pracy dziekanatu oraz ilość pracowników, należy ręcznie zmodyfikować parametry ustawione we wcześniej wspomnianej funkcji. Domyślnie tworzymy pięciu pracowników wykonujących zadania w godzinach 8-16.

3.2. Prezentacja działania

1. Początek pracy programu.

```
2> main:deansOffice().
Dziekanat otwarty. Zapraszamy!
Kolejka jest pusta -> Pracownik_5 rozmawia z kolega (5 min).
Kolejka jest pusta -> Pracownik_4 idzie na kawe (7 min).
Kolejka jest pusta -> Pracownik_3 rozmawia z kolega (6 min).
Kolejka jest pusta -> Pracownik_2 sprawdza poczte (6 min).
Kolejka jest pusta -> Pracownik_1 rozmawia z kolega (1 min).
<0.124.0>
3> Kolejka jest pusta -> Pracownik_1 idzie na kawe (5 min).
3> Marek -> sprawa: grupa poscigowa
3> Student Marek wchodzi do dziekanatu.
3> Marek podszedl do Pracownik_5. Sprawa: grupa poscigowa -> Pracownik udzieli informacji. Zajmie mu to 19 min.
3> Bartek -> sprawa: zaswiadczenie
3> Student Bartek wchodzi do dziekanatu.
3> Bartek podszedl do Pracownik_3. Sprawa: zaswiadczenie -> Pracownik wystawi zaswiadczenie. Zajmie mu to 1 min.
```

2. Tworzenie studentów i ich obsługa.

```
3> Kasia -> sprawa: skarga
3> Kolejka jest pusta -> Pracownik_2 idzie na przerwe (4 min).
3> Stefan -> sprawa: urlop dziekanski
3> Aga -> sprawa: urlop dziekanski
3> Karolina -> sprawa: urlop dziekanski
3> Student Kasia wchodzi do dziekanatu.
3> Student Stefan wchodzi do dziekanatu.
3> Kasia podszedl do Pracownik_1. Sprawa: skarga -> Student zlozy skarge. Zajmie mu to 1 min.
3> Stefan podszedl do Pracownik_2. Sprawa: urlop dziekanski -> Pracownik wprowadzi zmiany odnosnie toku studiow. Zajmie mu to 14 min.
3> Student Aga wchodzi do dziekanatu.
3> Aga podszedl do Pracownik_3. Sprawa: urlop dziekanski -> Pracownik wprowadzi zmiany odnosnie toku studiow. Zajmie mu to 3 min.
3> Student Karolina wchodzi do dziekanatu.
3> Karolina podszedl do Pracownik_1. Sprawa: urlop dziekanski -> Pracownik wprowadzi zmiany odnosnie toku studiow. Zajmie mu to 13 min.
3> Kolejka jest pusta -> Pracownik_4 idzie na kawe (5 min).
3> Ala -> sprawa: zaswiadczenie
3> Student Ala wchodzi do dziekanatu.
3> Ala podszedl do Pracownik_5. Sprawa: zaswiadczenie -> Pracownik wystawi zaswiadczenie. Zajmie mu to 9 min.
```

3. Koniec pracy programu.

```
4> Pracownik_2 skonczyl prace.
4> Basia -> sprawa: legitymacja
4> Marek -> sprawa: legitymacja
4> Pracownik_5 skonczyl prace.
4> Kamila -> sprawa: grupa poscigowa
4> Pracownik_3 skonczyl prace.
4> Jasiek -> sprawa: urlop dziekanski
4> Pracownik_1 skonczyl prace.
4> Karol -> sprawa: grupa poscigowa
4> Karol -> sprawa: urlop dziekanski
4> Pracownik_4 skonczyl prace.
4> Student Basia nie doczekal sie.
4> Student Marek nie doczekal sie.
4> Student Kamila nie doczekal sie.
4> Student Jasiek nie doczekal sie.
4> Student Karol nie doczekal sie.
4> Student Karol nie doczekal sie.
4> Dziekanat zamkniety.
Zapraszamy jutro!
```

4. Uwagi do działania programu

4.1. Ograniczenia symulatora

Ograniczeniem symulatora jest jedynie moment, w którym pamięć zostanie zapełniona poprzez zbyt dużą ilość studentów dodanych do kolejki FIFO. Jest to mało prawdopodobne. Za ograniczenia można również uznać konieczność ręcznego zmieniania czasu działania programu i ilości pracowników. Dodatkowo należałoby usprawnić losowanie ”nazwy” pracownika, ponieważ w przypadku zwiększenia ich liczby program kończy działanie. Spowodowane jest to tym, że lista, z której losujemy ”nazwę” posiada tylko pięć elementów. Jeśli chcemy zwiększyć liczbę pracowników, musimy dodać do niej nowe ”nazwy”. Jednak nie było to kluczowe założenie w trakcie realizacji projektu.

4.2. Możliwe rozszerzenia programu

Możliwych rozszerzeń programu jest mnóstwo.

- można popracować nad implementacją interfejsu graficznego wraz z panelem do obsługi dziekanatu
- warto by było opracować dodatkową funkcję generującą czas potrzebny na realizacje sprawy w zależności od jej "trudności"
- dodatkowo można zastanowić się nad dodaniem funkcjonalności definiującej godziny pracy dla poszczególnych pracowników