# Pracownia z Kursu języka Erlang
## Lista 5

W poniższych zadaniach dobrym pomysłem może być użycie modułu `error_logger` zamiast funkcji `io:format`. Zachęcam też do poznawania modułu `debugger`, jeśli będzie taka potrzeba.

1. **(2 pkt)** Rozszerz ideę szablonu procesu z wykładu. Moduł przechowujący implementację szablonu powinien eksportować następujące funkcje:

```
start(Module, Args) -> Result
start(ServerName, Module, Args) -> Result
start_link(Module, Args) -> Result
start_link(ServerName, Module, Args) -> Result
call(Server, Request) -> Reply
call(Server, Request, Timeout) -> Reply
cast(Server, Request) -> ok
reply(Client, Reply) -> ok
```

   - `start/2` i `start/3` startują serwer zaimplementowany w przekazanym module; przed wejściem do pętli głównej wykonują funkcję `Module:init/1`.
   - `start_link/2` i `start_link/3` działają analogicznie przy czym od razu tworzą łącze i powodują, że wywołujący proces staje się systemowym.

   Powyższe funkcje zwracają `{ok,Pid}`, a w przypadku błędu krotkę `{error,Error}` z jego opisem.

   - `call/2` i `call/3` służą do synchronicznej komunikacji z serwerem; po wysłaniu komunikatu `Request` wywołujący czeka ustaloną ilość czasu (domyślnie 10 sekund) na odpowiedź; jeśli w podanym czasie nie przyjdzie odpowiedź należy rzucić wyjątkiem.
   - `cast/2` asynchronicznie wysyła komunikat od razu wraca z atomem `ok`.
   - `reply/2` jest funkcją pomocniczą i służy do odesłania klientowi odpowiedzi.

   Szablon powinien odwoływać się do modułu, który implementuje właściwą część procesu. Moduł ten musi spełniać następujący interfejs:

```
init(Args) ->
  {ok,NewState} |
  {ok,NewState,Timeout} |
  {stop,Reason}
handle_call(Request, Client, State) ->
  {reply,Reply,NewState} |
  {reply,Reply,NewState,Timeout} |
  {stop,Reason,Reply,NewState}
handle_cast(Request, State) ->
  {noreply,NewState} |
  {noreply,NewState,Timeout} |
  {stop,Reason,NewState}
handle_info(Info, State) ->
  {noreply,NewState} |
  {noreply,NewState,Timeout} |
  {stop,Reason,NewState}
terminate(Reason, State) -> ok
```

   Parametr `Timeout` pojawiający się w wywołaniach oznacza czas oczekiwania serwera na następny komunikat. Po przekroczeniu tego czasu atom `timeout` zostaje wysłany jako pierwszy argument do `handle_info`. Parametr `State` jest stanem serwera przechowywanym w pętli głównej. Zwrócenie, z którejkolwiek funkcji krotki zaczynającej się od atomu `stop` powoduje zatrzymanie serwera. Parametr `Reason` odpowiada parametrowi funkcji `exit/1`, jeśli jest atomem `normal` oznacza prawidłowe zakończenie procesu.

   - `init/1` na podstawie argumentów ustala początkowy stan serwera.

- `handle_call/3` obsługa komunikatu wymagającego odpowiedzi.
- `handle_cast/2` obsługa komunikatu wysłanego asynchronicznie.
- `handle_info/2` obsługa błędów, zdarzeń wyjątkowych, nieznanych komunikatów; tu przychodzą wszystkie komunikaty, które nie zostały obsłużone przez `handle_call` i `handle_cast`, czyli między innymi sygnały błędów.
- `terminate/2` wywoływane w wyniku zakończenia działania serwera; powoduje zwolnienie wszystkich zasobów z nim związanych.

**Uwaga:** Przygotuj jakiś prosty proces spełniający powyższe wymagania, który będzie pokazywał, że Twój szablon procesu działa prawidłowo. Pamiętaj o tym, żeby odpowiednio opakować wywołania `call`, `cast`, tak by użytkownik procesu nie musiał ręcznie konstruować komunikatów itd.

*Komentarz: Celem zadania jest tak na prawdę częściowa implementacja zachowania* `gen_server`*. Chcemy pokazać, że warto oddzielać zachowanie od implementacji.*

2. **(1 pkt)** Stwórz uproszczoną wersję zarządcy procesów. W tym celu zaimplementuj dwa proste procesy.

   Pierwszy (robotnik) będzie kończył swoje działanie na różne sposoby:

   - normalnie,
   - wyjście w wyniku jakiegoś błędu (niezłapany wyjątek),
   - bezwarunkowe zakończenie (poprzez wysłanie sygnału `kill`).

   Drugi (nadzorca) można będzie wystartować na dwa sposoby – tak, by tworzył domyślnie łącza lub monitory. Proces nadzorcy utworzy przy starcie pewną ilość robotników, a kończąc prześle do nich sygnał `kill`. Jeśli którykolwiek z robotników zakończy swoje działanie, nadzorca go zrestartuje.

   **Uwaga:** Przygotuj prezentację rozwiązania. Należy pokazać: jakie komunikaty wchodzą do nadzorcy w wyniku śmierci procesu robotnika, oraz jakie akcje są podejmowane.

   Do implementacji można użyć zachowania `gen_server` lub szablonu procesu z poprzedniego zadania.

Lista i materiały znajdują się pod adresem
http://cahirwpz.cs.uni.wroc.pl/main-pl/erlang-language-summer-2010/ *Krystian Bacławski*