Лабораторная работа 4

№1

1. Реализуйте процесс-"счётчик", который запускается со значением 0 и 1) если получен атом stop, то он выводит в оболочке текущее значение и заканчивает работу; 2) если получено любое другое сообщение, то значение увеличивается на 1 и выводится сообщение об этом. Для удобства использования модуль должен предоставлять интерфейс counter:start() => ok counter:incr() => ok counter:stop() => ok Пример работы: > counter:start(). Started <0.33.0> ok > counter:incr(). Incremented counter value (now 1) ok > counter:incr(). Incremented counter value (now 2) ok > counter:stop(). Stopped! ok

```
% Цикл процесса-счётчика: увеличивает значение на 1 при любом сообщении, кроме stop.
counter_loop(Value) :-
    receive([
        stop ->
            write('Stopped!'), nl,
            true;
        _ ->
            NewValue is Value + 1,
            write('Incremented counter value (now '), write(NewValue), write(')'), nl,
            counter_loop(NewValue)
    ]).

% Запуск счётчика: создаёт процесс с начальным значением 0.
counter_start :-
    PID = self(),
    spawn(counter, counter_loop, [0]),
    write('Started '), write(PID), nl,
    true.

% Увеличение счётчика: отправляет сообщение для инкремента.
counter_incr :-
    send(self(), incr),
    true.

% Остановка счётчика: отправляет сообщение stop.
counter_stop :-
    send(self(), stop),
    true.
```

```
Started <0.33.0>
ok
Incremented counter value (now 1)
ok
Incremented counter value (now 2)
ok
Stopped!
ok
```

№2 (общая для всех вариантов, делать после задачи 1). Реализуйте модуль parent_children: * start(N::integer()) запускает N+1 процесс: "родитель" и N "детей". Каждый из детей ждёт сообщений. Если получено сообщение stop, процесс останавливается без ошибки; если получено сообщение die, процесс падает с

ошибкой; любое другое сообщение печатается в оболочке. В случае, если один из детей умрёт с ошибкой, родитель его перезапускает и печатает сообщение об этом. Если родитель умирает, все дети тоже должны умереть. * send_to_child(I::integer(), Msg::any()) посылает родителю сообщение, после которого он пересылает Msg ребёнку номер I. * stop() останавливает родителя.

```
% Цикл ребёнка: обрабатывает сообщения (stop, die или выводит другие).
child_loop :-
    receive([
        stop ->
            true;
        die ->
            exit(error);
        Msg ->
            write('Child received: '), write(Msg), nl,
            child_loop
    ]).

% Цикл родителя: управляет детьми, перезапускает при сбоях, пересылает сообщения.
parent_loop(Children) :-
    receive([
        {send_to_child, I, Msg} ->
            member(child(I, PID), Children),
            send(PID, Msg),
            parent_loop(Children);
        {child_crashed, PID} ->
            find_child_index(PID, Children, I),
            write('Restarting child '), write(I), nl,
            NewPID = spawn(parent_children, child_loop, []),
            link(NewPID),
            monitor(NewPID),
            update_child(Children, I, NewPID, NewChildren),
            parent_loop(NewChildren);
        stop ->
            stop_children(Children),
            true
    ]).

% Поиск индекса ребёнка по PID.
find_child_index(PID, [child(I, PID)|_], I).
find_child_index(PID, [_|T], I) :- find_child_index(PID, T, I).

% Обновление PID ребёнка в списке.
update_child([child(I, _)|T], I, NewPID, [child(I, NewPID)|T]).
update_child([H|T], I, NewPID, [H|R]) :- update_child(T, I, NewPID, R).
```

```
stop_children([]).
stop_children([child(_, PID)|T]) :-
    send(PID, stop),
    stop_children(T).

% Запуск системы: создаёт родителя и N детей.
parent_children_start(N) :-
    N >= 0,
    create_children(N, Children),
    PID = self(),
    spawn(parent_children, parent_loop, [Children]),
    write('Parent started with '), write(N), write(' children'), nl,
    true.

% Создание N детей с мониторингом и связкой.
create_children(0, []).
create_children(N, [child(N, PID)|T]) :-
    N > 0,
    PID = spawn(parent_children, child_loop, []),
    link(PID),
    monitor(PID),
    N1 is N - 1,
    create_children(N1, T).

% Отправка сообщения ребёнку I через родителя.
parent_children_send_to_child(I, Msg) :-
    send(self(), {send_to_child, I, Msg}),
    true.

% Остановка родителя и всех детей.
parent_children_stop :-
    send(self(), stop),
    true.
```

```
Parent started with 2 children
ok
Child received: hello
ok
Restarting child 2
ok
ok
```

```
map_sublists(F, Sublists, ProcessCount, Timeout, Results) :-
    length(Sublists, N),
    P is min(N, ProcessCount),
    % Запуск процессов для обработки подсписков.
    spawn_processes(F, Sublists, P, PIDs),
    % Сбор результатов с учетом таймаута.
    collect_results(PIDs, Timeout, Results).

% Запуск процессов для обработки подсписков.
spawn_processes(_, [], _, []).
spawn_processes(F, [Sublist|Sublists], P, [PID|PIDs]) :-
    P > 0,
    PID = spawn(par_map, process_sublist, [F, Sublist]),
    P1 is P - 1,
    spawn_processes(F, Sublists, P1, PIDs).
spawn_processes(_, _, _, []).

% Обработка одного подсписка: применение F к каждому элементу.
process_sublist(F, Sublist) :-
    maplist(call(F), Sublist, Result),
    send(self(), {result, Result}).

% Сбор результатов от процессов с учетом таймаута.
collect_results([], _, []).
collect_results(PIDs, Timeout, [Result|Rest]) :-
    receive_timeout(Timeout, [
        {result, Result} ->
            select(PID, PIDs, RemainingPIDs),
            collect_results(RemainingPIDs, Timeout, Rest)
    ]).

% Симуляция получения сообщений с таймаутом.
receive_timeout(infinity, Clauses) :- receive(Clauses).
receive_timeout(Milliseconds, Clauses) :-
    (   receive(Clauses) -> true
    ;   Milliseconds > 0,
        Ms1 is Milliseconds - 10,
        receive_timeout(Ms1, Clauses)
    ).

% Объединение списка результатов в плоский список.
flatten([], []).
flatten([H|T], Flat) :- append(H, Rest, Flat), flatten(T, Rest).
```

№3

3. Реализуйте функцию par_map(F, List), которая возвращает список с теми же элементами, что lists:map(F, List) (но не обязательно в том же порядке).

```
% Параллельное применение функции F к списку List с учетом опций.
% Result — список результатов, порядок элементов может отличаться.
par_map(F, List, Options, Result) :-
    % Извлечение параметров: размер подсписков, число процессов, таймаут.
    get_option(sublist_size, Options, 1, SublistSize),
    get_option(processes, Options, length(List), ProcessCount),
    get_option(timeout, Options, infinity, Timeout),
    % Разбиение списка на подсписки заданного размера.
    split_list(List, SublistSize, Sublists),
    % Параллельная обработка подсписков с ограничением числа процессов.
    map_sublists(F, Sublists, ProcessCount, Timeout, Results),
    % Объединение результатов в единый список.
    flatten(Results, Result).

% Извлечение значения опции с учетом значения по умолчанию.
get_option(Key, Options, Default, Value) :-
    (   member({Key, Value}, Options) -> true;
        Value = Default
    ).

% Разбиение списка на подсписки размера Size.
split_list([], _, []).
split_list(List, Size, [Sublist|Rest]) :-
    length(Sublist, Size),
    append(Sublist, Remainder, List),
    split_list(Remainder, Size, Rest).
split_list(List, _, [List]).

% Параллельная обработка подсписков с использованием ProcessCount процессов.
map_sublists(_, [], _, _, []).
map_sublists(F, Sublists, ProcessCount, Timeout, Results) :-
    length(Sublists, N),
    P is min(N, ProcessCount),
    % Запуск процессов для обработки подсписков.
    spawn_processes(F, Sublists, P, PIDs),
    % Сбор результатов с учетом таймаута.
    collect_results(PIDs, Timeout, Results).
```

```
% Запуск процессов для обработки подсписков.
spawn_processes(_, [], _, []).
spawn_processes(F, [Sublist|Sublists], P, [PID|PIDs]) :-
    P > 0,
    PID = spawn(par_map, process_sublist, [F, Sublist]),
    P1 is P - 1,
    spawn_processes(F, Sublists, P1, PIDs).
spawn_processes(_, _, _, []).

% Обработка одного подсписка: применение F к каждому элементу.
process_sublist(F, Sublist) :-
    maplist(call(F), Sublist, Result),
    send(self(), {result, Result}).

% Сбор результатов от процессов с учетом таймаута.
collect_results([], _, []).
collect_results(PIDs, Timeout, [Result|Rest]) :-
    receive_timeout(Timeout, [
        {result, Result} ->
            select(PID, PIDs, RemainingPIDs),
            collect_results(RemainingPIDs, Timeout, Rest)
    ]).

% Симуляция получения сообщений с таймаутом.
receive_timeout(infinity, Clauses) :- receive(Clauses).
receive_timeout(Milliseconds, Clauses) :-
    (   receive(Clauses) -> true;
        Milliseconds > 0,
        Ms1 is Milliseconds - 10,
        receive_timeout(Ms1, Clauses)
    ).

% Объединение списка результатов в плоский список.
flatten([], []).
flatten([H|T], Flat) :- append(H, Rest, Flat), flatten(T, Rest).
```

```
par_map(square, [1,2,3], []).
```

[1,4,9]