Лабораторная работа 4

1. Загрузить среду программирования.

2. Выполнить задачи по варианту. Номер варианта равен номеру рабочего места.

3. Представить результат преподавателю.

Варианты:

1	1. Реализуйте функцию ring(N, M), которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй -- третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID будут отличаться): > ring:ring(3, 2). Current process is <0.31.0> Created <0.33.0> Created <0.34.0> Created <0.35.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.35.0> received 1 from <0.34.0> <0.33.0> received 2 from <0.35.0> <0.33.0> finished <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.34.0> <0.35.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists.

	3. Реализуйте функцию par_filter(F, List, Options), которая возвращает список с теми же элементами, что lists:filter(F, List) (но не обязательно в том же порядке).
2	1. Реализуйте функцию star(N, M), которая создаёт N+1 процессов (1 "центральный" и N "крайних") и посылает сообщение центральному процессу, который посылает сообщение всем остальным процессам и дожидается от них ответа, после чего это повторяется (всего M раз). После того, как все сообщения получены, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID и порядок строк будут отличаться): > star:star(3, 2). Current process is <0.31.0> Created <0.33.0> (center) Created <0.34.0> Created <0.35.0> Created <0.36.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.34.0> <0.35.0> received 1 from <0.33.0> <0.36.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.35.0> <0.33.0> received 1 from <0.36.0> <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.35.0> <0.35.0> finished <0.36.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.36.0> <0.33.0> received 2 from <0.34.0> <0.36.0> finished <0.33.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists.

	3. Реализуйте функцию par_foreach(F, List, Options), которая работает так же, как lists:foreach(F, List) (но параллельно). Заметьте, что закончить работу можно только тогда, когда функция F применена ко всем значениям в List!
3	1. Реализуйте процесс-"эхо", который ожидает сообщения и 1) если получен атом stop, то он заканчивает работу; 2) если получено {print, Term}, то выводит Term в оболочке. Для удобства использования модуль должен предоставлять интерфейс echo:start() => ok echo:print(Term) => ok echo:stop() => ok Пример работы: > echo:start(). Started <0.33.0> ok > echo:print(1). 1 ok > echo:print(stop). stop ok > echo:stop(). Stopped! Ok 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_partition(F, List, Options), которая возвращает пару списков с теми же элементами, что lists:partition(F, List) (но не обязательно в том же порядке).
4	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 Задача 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 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).
5	1. Реализуйте функцию ring(N, M), которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй -- третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID будут отличаться): > ring:ring(3, 2). Current process is <0.31.0>

	Created <0.33.0> Created <0.34.0> Created <0.35.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.35.0> received 1 from <0.34.0> <0.33.0> received 2 from <0.35.0> <0.33.0> finished <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.34.0> <0.35.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_filter(F, List, Options), которая возвращает список с теми же элементами, что lists:filter(F, List) (но не обязательно в том же порядке).
6	1. Реализуйте функцию star(N, M), которая создаёт N+1 процессов (1 "центральный" и N "крайних") и посылает сообщение центральному процессу, который посылает сообщение всем остальным процессам и дожидается от них ответа, после чего это повторяется (всего M раз). После того, как все сообщения получены, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID и порядок строк будут отличаться): > star:star(3, 2). Current process is <0.31.0> Created <0.33.0> (center) Created <0.34.0> Created <0.35.0> Created <0.36.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.34.0> <0.35.0> received 1 from <0.33.0> <0.36.0> received 1 from <0.33.0>

	<0.33.0> received 1 from <0.35.0> <0.33.0> received 1 from <0.36.0> <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.35.0> <0.35.0> finished <0.36.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.36.0> <0.33.0> received 2 from <0.34.0> <0.36.0> finished <0.33.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_foreach(F, List, Options), которая работает так же, как lists:foreach(F, List) (но параллельно). Заметьте, что закончить работу можно только тогда, когда функция F применена ко всем значениям в List!
7	1. Реализуйте процесс-"эхо", который ожидает сообщения и 1) если получен атом stop, то он заканчивает работу; 2) если получено {print, Term}, то выводит Term в оболочке. Для удобства использования модуль должен предоставлять интерфейс echo:start() => ok echo:print(Term) => ok echo:stop() => ok Пример работы: > echo:start(). Started <0.33.0> ok > echo:print(1). 1 ok > echo:print(stop). stop

	ok > echo:stop(). Stopped! Ok 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_partition(F, List, Options), которая возвращает пару списков с теми же элементами, что lists:partition(F, List) (но не обязательно в том же порядке).
8	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 Задача 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 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).
9	1. Реализуйте функцию ring(N, M), которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй -- третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID будут отличаться): > ring:ring(3, 2). Current process is <0.31.0> Created <0.33.0> Created <0.34.0> Created <0.35.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.35.0> received 1 from <0.34.0> <0.33.0> received 2 from <0.35.0> <0.33.0> finished <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.34.0> <0.35.0> finished 2. 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.

	Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_filter(F, List, Options), которая возвращает список с теми же элементами, что lists:filter(F, List) (но не обязательно в том же порядке).
10	1. Реализуйте функцию star(N, M), которая создаёт N+1 процессов (1 "центральный" и N "крайних") и посылает сообщение центральному процессу, который посылает сообщение всем остальным процессам и дожидается от них ответа, после чего это повторяется (всего M раз). После того, как все сообщения получены, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID и порядок строк будут отличаться): > star:star(3, 2). Current process is <0.31.0> Created <0.33.0> (center) Created <0.34.0> Created <0.35.0> Created <0.36.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.34.0> <0.35.0> received 1 from <0.33.0> <0.36.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.35.0> <0.33.0> received 1 from <0.36.0> <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.35.0> <0.35.0> finished <0.36.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.36.0> <0.33.0> received 2 from <0.34.0> <0.36.0> finished <0.33.0> finished 2. 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.

	Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_foreach(F, List, Options), которая работает так же, как lists:foreach(F, List) (но параллельно). Заметьте, что закончить работу можно только тогда, когда функция F применена ко всем значениям в List!
11	1. Реализуйте процесс-"эхо", который ожидает сообщения и 1) если получен атом stop, то он заканчивает работу; 2) если получено {print, Term}, то выводит Term в оболочке. Для удобства использования модуль должен предоставлять интерфейс echo:start() => ok echo:print(Term) => ok echo:stop() => ok Пример работы: > echo:start(). Started <0.33.0> ok > echo:print(1). 1 ok > echo:print(stop). stop ok > echo:stop(). Stopped! Ok 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes

	каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_partition(F, List, Options), которая возвращает пару списков с теми же элементами, что lists:partition(F, List) (но не обязательно в том же порядке).
12	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 Задача 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 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).

13	1. Реализуйте функцию ring(N, M), которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй -- третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID будут отличаться): > ring:ring(3, 2). Current process is <0.31.0> Created <0.33.0> Created <0.34.0> Created <0.35.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.35.0> received 1 from <0.34.0> <0.33.0> received 2 from <0.35.0> <0.33.0> finished <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.34.0> <0.35.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_filter(F, List, Options), которая возвращает список с теми же элементами, что lists:filter(F, List) (но не обязательно в том же порядке).
14	1. Реализуйте функцию star(N, M), которая создаёт N+1 процессов (1 "центральный" и N "крайних") и посылает сообщение центральному процессу, который посылает сообщение всем остальным процессам и дожидается от них ответа, после чего это повторяется (всего M раз). После того, как все сообщения получены, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID и порядок строк будут отличаться): > star:star(3, 2). Current process is <0.31.0>

	Created <0.33.0> (center) Created <0.34.0> Created <0.35.0> Created <0.36.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.34.0> <0.35.0> received 1 from <0.33.0> <0.36.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.35.0> <0.33.0> received 1 from <0.36.0> <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.35.0> <0.35.0> finished <0.36.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.36.0> <0.33.0> received 2 from <0.34.0> <0.36.0> finished <0.33.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_foreach(F, List, Options), которая работает так же, как lists:foreach(F, List) (но параллельно). Заметьте, что закончить работу можно только тогда, когда функция F применена ко всем значениям в List!
15	1. Реализуйте процесс-"эхо", который ожидает сообщения и 1) если получен атом stop, то он заканчивает работу; 2) если получено {print, Term}, то выводит Term в оболочке. Для удобства использования модуль должен предоставлять интерфейс echo:start() => ok echo:print(Term) => ok echo:stop() => ok Пример работы:

	> echo:start(). Started <0.33.0> ok > echo:print(1). 1 ok > echo:print(stop). stop ok > echo:stop(). Stopped! Ok 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_partition(F, List, Options), которая возвращает пару списков с теми же элементами, что lists:partition(F, List) (но не обязательно в том же порядке).
16	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 Задача 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 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).
17	1. Реализуйте функцию ring(N, M), которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй -- третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID будут отличаться): > ring:ring(3, 2). Current process is <0.31.0> Created <0.33.0> Created <0.34.0> Created <0.35.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.35.0> received 1 from <0.34.0> <0.33.0> received 2 from <0.35.0> <0.33.0> finished <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.34.0> <0.35.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_filter(F, List, Options), которая возвращает список с теми же элементами, что lists:filter(F, List) (но не обязательно в том же порядке).
18	1. Реализуйте функцию star(N, M), которая создаёт N+1 процессов (1 "центральный" и N "крайних") и посылает сообщение центральному процессу, который посылает сообщение всем остальным процессам и дожидается от них ответа, после чего это повторяется (всего M раз). После того, как все сообщения получены, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID и порядок строк будут отличаться): > star:star(3, 2). Current process is <0.31.0> Created <0.33.0> (center) Created <0.34.0> Created <0.35.0> Created <0.36.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.34.0> <0.35.0> received 1 from <0.33.0> <0.36.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.35.0> <0.33.0> received 1 from <0.36.0> <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.35.0> <0.35.0> finished <0.36.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.36.0> <0.33.0> received 2 from <0.34.0> <0.36.0> finished <0.33.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_foreach(F, List, Options), которая работает так же, как lists:foreach(F, List) (но параллельно). Заметьте, что закончить работу можно только тогда, когда функция F применена ко всем значениям в List!
19	1. Реализуйте процесс-"эхо", который ожидает сообщения и 1) если получен атом stop, то он заканчивает работу; 2) если получено {print, Term}, то выводит Term в оболочке. Для удобства использования модуль должен предоставлять интерфейс echo:start() => ok echo:print(Term) => ok echo:stop() => ok Пример работы: > echo:start(). Started <0.33.0> ok > echo:print(1). 1 ok > echo:print(stop). stop ok > echo:stop(). Stopped! Ok 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_partition(F, List, Options), которая возвращает пару списков с теми же элементами, что lists:partition(F, List) (но не обязательно в том же порядке).
20	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 Задача 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые

	обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 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).
21	1. Реализуйте функцию ring(N, M), которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй -- третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID будут отличаться): > ring:ring(3, 2). Current process is <0.31.0> Created <0.33.0> Created <0.34.0> Created <0.35.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.35.0> received 1 from <0.34.0> <0.33.0> received 2 from <0.35.0> <0.33.0> finished <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.34.0> <0.35.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_filter(F, List, Options), которая возвращает список с теми же элементами, что lists:filter(F, List) (но не обязательно в том же порядке).
22	1. Реализуйте функцию star(N, M), которая создаёт N+1 процессов (1 "центральный" и N "крайних") и посылает сообщение центральному процессу, который посылает сообщение всем остальным процессам и дожидается от них

	ответа, после чего это повторяется (всего M раз). После того, как все сообщения получены, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID и порядок строк будут отличаться): > star:star(3, 2). Current process is <0.31.0> Created <0.33.0> (center) Created <0.34.0> Created <0.35.0> Created <0.36.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.34.0> <0.35.0> received 1 from <0.33.0> <0.36.0> received 1 from <0.33.0> <0.33.0> received 1 from <0.35.0> <0.33.0> received 1 from <0.36.0> <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.35.0> <0.35.0> finished <0.36.0> received 2 from <0.33.0> <0.33.0> received 2 from <0.36.0> <0.33.0> received 2 from <0.34.0> <0.36.0> finished <0.33.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_foreach(F, List, Options), которая работает так же, как lists:foreach(F, List) (но параллельно). Заметьте, что закончить работу можно только тогда, когда функция F применена ко всем значениям в List!
23	1. Реализуйте процесс-"эхо", который ожидает сообщения и 1) если получен атом stop, то он заканчивает работу; 2) если получено {print, Term}, то выводит Term в оболочке.

	Для удобства использования модуль должен предоставлять интерфейс echo:start() => ok echo:print(Term) => ok echo:stop() => ok Пример работы: > echo:start(). Started <0.33.0> ok > echo:print(1). 1 ok > echo:print(stop). stop ok > echo:stop(). Stopped! Ok 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_partition(F, List, Options), которая возвращает пару списков с теми же элементами, что lists:partition(F, List) (но не обязательно в том же порядке).
24	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 Задача 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 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).
25	1. Реализуйте функцию ring(N, M), которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй -- третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы должны закончить работу. Все события выводятся в оболочке с помощью io:format. Пример работы (конкретные PID будут отличаться): > ring:ring(3, 2). Current process is <0.31.0> Created <0.33.0> Created <0.34.0> Created <0.35.0> <0.33.0> received 0 from <0.31.0> <0.34.0> received 1 from <0.33.0> <0.35.0> received 1 from <0.34.0> <0.33.0> received 2 from <0.35.0>

	<0.33.0> finished <0.34.0> received 2 from <0.33.0> <0.34.0> finished <0.35.0> received 2 from <0.34.0> <0.35.0> finished 2. 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. Параметр Options в задаче 3 всех вариантов -- список, который может содержать следующие элементы: {sublist_size, integer()} (размер частей, на которые разбивается список), {processes, integer()} (число процессов, которые обрабатывают части списка), {timeout, Milliseconds::integer()|infinity} (максимальное время, за которое функция должна закончить работу). При отсутствии sublist_size в каждой части списка 1 элемент. При отсутствии processes каждая часть обрабатывается отдельным процессом. Для работы с такими списками можно использовать модуль proplists. 3. Реализуйте функцию par_filter(F, List, Options), которая возвращает список с теми же элементами, что lists:filter(F, List) (но не обязательно в том же порядке).

Дополнительные задания:

4-5. Познакомиться с OTP, и оформить решение задач 1 и 2 как OTP-приложение. Срок -- до конца семестра!

6 (кроме варианта 3). То же, что задание 3, но элементы должны возвращаться в том же порядке.

6 (вариант 3). Реализуйте функцию par_sort(List, Options), которая сортирует список параллельно с помощью сортировки

Критерии оценивания

	Задание сдано в срок	**Задание сдано позже**
Задача 1 выполнена верно		
Задача 2 выполнена верно		
Задача 3 выполнена верно		
Верно выполнено дополнительное задание 4		
Верно выполнено дополнительное задание 5		
Верно выполнено дополнительное задание 6		
Итого	7	3,5