

Assignment Project Exam Help
Message Passing

<https://powcoder.com>

Add WeChat powcoder

Message Passing
Assignment Project Exam Help

Exceptions
<https://powcoder.com>

Links and Monitors
Add WeChat powcoder

Assignment Project Exam Help

- Previously
 - Shared memory (low-level, non-structured)
 - Semaphores (low-level, non-structured)
 - Monitors (popular, structured, encapsulate synchronization)
- So what's the problem with monitors?
 - Highly centralized (un/blocking processes, maintaining queues of blocked processes, encapsulating data)
 - For modern, distributed architectures, need for less centralized solution

<https://powcoder.com>

Add WeChat powcoder

- Turn to interaction through communication rather than sharing

The Message Passing Model

- ▶ No shared memory
- ▶ A process sends a message
- ▶ Another process receives the message

- ▶ Operations:

```
receive(Var);  
send(PID,msg);
```

- ▶ `receive` blocks until a message is available in the mailbox
- ▶ `send(PID,msg)` is non-blocking; it sends message `msg` to process `PID`

- ▶ This model is the **asynchronous communication model** and is the one used in Erlang

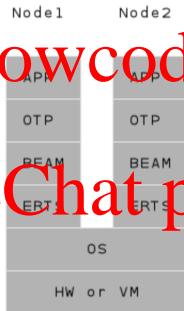
Nodes and Processes in Erlang¹

- ▶ A distributed Erlang system consists of a number of Erlang runtime systems communicating with each other (instances of the VM)

- ▶ Each such runtime system is called a **node**

<https://powcoder.com>

Add WeChat powcoder



¹Source: <https://blog.stenmans.org/theBeamBook/>

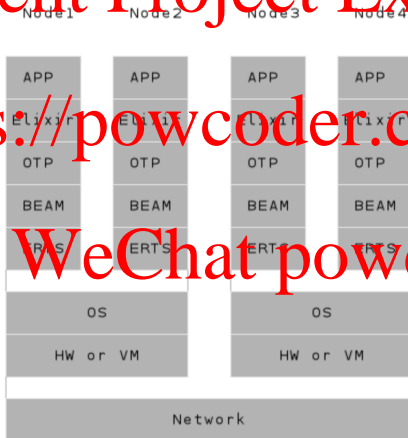
Distributed Nodes in Erlang

- ▶ A distributed Erlang system consists of a number of Erlang runtime systems communicating with each other (instances of the VM)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Nodes and Processes in Erlang

Assignment Project Exam Help

- ▶ Each such runtime system is called a `node`
 - ▶ node name is an atom `name@host`
 - ▶ `name` is the name given by the user
 - ▶ `host` is the full host name if long names are used, or the first part of the host name if short names are used
- ▶ The name of a node may be consulted using `node()`

```
1 1> node().  
2 node@prohost
```

Add WeChat powcoder

Processes and Communication in Erlang

- ▶ A **process** in a node has
 - ▶ a process id (pid)

```
1 > elc().  
2 <0.18.0>
```

- ▶ its own memory (a mailbox, a heap and a stack); and
- ▶ a process control block (PCB) with information about the process.

- ▶ Message passing between processes at different nodes, as well as links and monitors, are transparent when pids are used
- ▶ Registered names, however, are local to each node.

- ▶ Format of a PID:

- ▶ node id where process lives; 0 if node is local
- ▶ process index itself (index into process table)
- ▶ serial which increases every time MAXPROCS has been reached.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

A Simple Echo Server

- ▶ Process `echo` will receive a message and then send it back to the sender
- ▶ After that it will continue to wait for a new message
- ▶ It may be stopped by sending it the `stop` message

```
1 echo() ->  
2   receive  
3     {From, Msg} ->  
4       From ! {Msg},  
5       echo()  
6   stop -> true  
7 end.
```

<https://powcoder.com>

Add WeChat powcoder

- ▶ Processes are created using `spawn/1` and `spawn/3`

A Simple Echo Server (cont.)

```
1 -module(echo).
2 -export([start/0]).
3
4 echo() ->
5     receive
6         {From, Msg} ->
7             From ! {Msg},
8             echo();
9         stop -> true
10    end.
11
12 start() ->
13     Pid = spawn(fun echo/0), % Returns pid of a new process
14     % started by the application of echo/0 to []
15     Token = "Hello Server!", % Sending tokens to the server
16     Pid ! {self(), Token},
17     io:format("Sent ~s~n", [Token]),
18     receive
19         {Msg} ->
20             io:format("Received ~s~n", [Msg])
21     end,
22     Pid ! stop.           % Stop server
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

A Simple Echo Server

```
1 1> echo:start( ).
2 Sent Hello Server
3 Received Hello Server!
4 stop
```

If we export `echo/0` we can spawn from the interpreter:

```
1 59> X=spawn(fun echo echo/0).
2 <0.198.0>
3 60> X!{self(), "hello"}.
4 {<0.60.0>, "hello"}
5 61> X.
6 <0.198.0>
```

Note: the value of a send is the value of the message

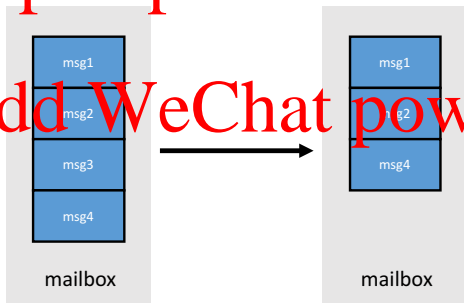
Reacting to Multiple Messages

- ▶ Erlang “listens” for messages from different senders
- ▶ In which order will they be processed?
- ▶ Can we force an order?
- ▶ A receive statement tries to find a match as early in the mailbox as it can

```
1 receive  
2   msg3 -> 42  
3 end
```

<https://powcoder.com>

Add WeChat powcoder



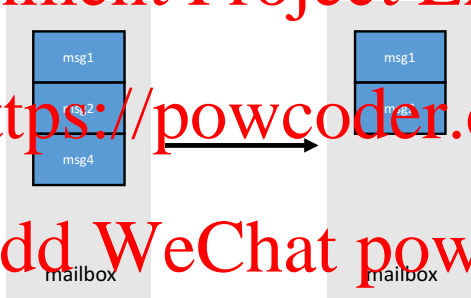
Reacting to Multiple Messages

```
1 receive
2   msg4 -> 42
3 end
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Reacting to Multiple Messages

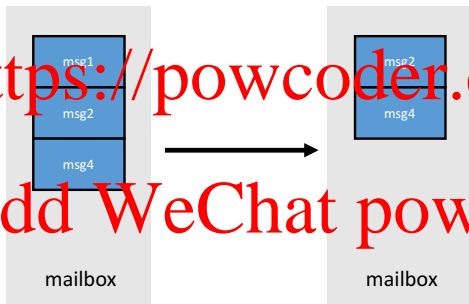
Waiting for multiple messages

```
1 receive
2   msg4 -> 42;
3   -> 41
4 end
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



- The oldest message is tried against every pattern of the receive until one of them matches

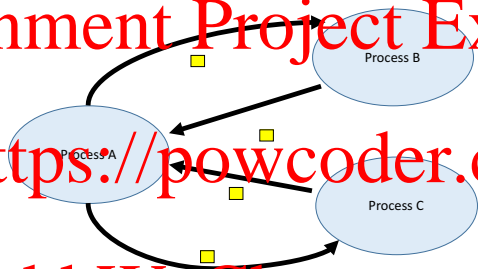
Sources of Multiple Messages

Multiple messages can come from different processes

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



- ▶ How do we know who sent a message?
- ▶ Distinguish the source by Pids

Sources of Multiple Messages

```
1 -module(echo2).  
2 -export([start/0]).  
3  
4 echo() ->  
5     receive  
6         {From, Msg} ->  
7             timer:sleep(rand:uniform(100)),  
8             From ! {Msg},  
9             echo();  
10        stop ->  
11            true  
12    end.  
13  
14 % continued on next slide.
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

- ▶ `timer:sleep(N)` sleeps a process for `N` milliseconds
- ▶ `rand:uniform(N)` produces a random integer between 1 and `N`

Sources of Multiple Messages

```
1 start() ->
2     PidB = spawn(fun echo/0),
3     PidC = spawn(fun echo/0),
4
5     % sending tokens
6     Token = 42,
7     PidB ! {self(), Token},
8     io:format("Sent ~w~n", [Token]),
9     Token2 = 41,
10    PidC ! {self(), Token2},
11    io:format("Sent ~w~n", [Token2]),
12
13    % receive message
14    receive
15    [Msg] ->
16        io:format("Received ~w~n", [Msg])
17    end,
18
19    % stop echo-servers
20    PidB ! stop,
21    PidC ! stop.
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Sources of Multiple Messages

- ▶ How do we know who sent a message?

- ▶ Distinguish the source by Pids

```
1 -module(echo2).  
2 -export([start/0]).  
3  
4 echo() ->  
5     receive  
6         {From, Msg} ->  
7             timer:sleep(rand:uniform(100)),  
8             From ! {self(), Msg},  
9             echo(),  
10        stop ->  
11            true  
12    end.  
13  
14 % continued on next slide...
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Sources of Multiple Messages

```
1 start() ->
2     PidB = spawn(fun echo/0),
3     PidC = spawn(fun echo/0),
4
5     % sending tokens
6     Token = 42,
7     PidB ! {self(), Token},
8     io:format("Sent ~w~n", [Token]),
9     Token2 = 41,
10    PidC ! {self(), Token2},
11    io:format("Sent ~t~n", [Token2]),
12
13    % receive messages
14    receive
15        {PidB, Msg} ->
16            io:format("Received from B: ~w~n", [Msg]);
17        {PidC, Msg} ->
18            io:format("Received from C: ~w~n", [Msg])
19    end,
20
21    % stop echo-servers
22    PidB ! stop,
23    PidC ! stop.
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder;

Sources of Multiple Messages

```
1 11> echo2:start().
2 Sent42
3 Sent41
4 Received from F: 42
5 stop
6 12> echo2:start().
7 Sent42
8 Sent41
9 Received from B: 42
10 stop
11 13> echo2:start().
12 Sent42
13 Sent41
14 Received from C: 41
15 stop
16 14> echo2:start().
17 Sent42
18 Sent41
19 Received from B: 42
20 stop
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Sources of Multiple Messages

Multiple messages can come from the same processes

Assignment Project Exam Help

- ▶ Send several messages of the same shape and continue computing
- ▶ When receiving the responses, how can the code match them to the appropriate request?
- ▶ BIF `make_ref` provides globally unique reference objects (references for short) different from every other object in the Erlang system, including remote nodes
- ▶ References can be used to uniquely identify messages

<https://powcoder.com>

Add WeChat powcoder

Sources of Multiple Messages

Assignment Project Exam Help

```
1 -module(echo3).  
2 -export([start/0]).  
3  
4 echo() ->  
5     receive  
6         {From, Ref, Msg} ->  
7             From ! {self(), Ref, Msg},  
8             echo();  
9         stop ->  
10            true  
11     end  
12  
13 % continues in next slide...
```

<https://powcoder.com>

Add WeChat powcoder

Sources of Multiple Messages

```
1 start() ->
2   PidB = spawn(fun echo/0),
3   % sending tokens
4   Token = 42,
5   Ref = make_ref(),
6   PidB ! {self(), Ref, Token},
7   io:format("Sent ~w~n", [Token]),
8   Token2 = 41,
9   Ref2 = make_ref(),
10  PidB ! {self(), Ref2, Token2},
11  io:format("Sent ~w~n", [Token2]),
12  % receive messages
13  receive
14    {PidB, Ref2, Msg} ->
15      io:format("Received 41? ~w~n", [Msg]);
16    {PidB, Ref, Msg} ->
17      io:format("Received 42? ~w~n", [Msg])
18
19  end,
20
21  % stop echo-servers
22  PidB ! stop.
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

- ▶ Clauses can have guards
- ▶ Guards must be composed of terminating functions (BIFs)

```
1 receive  
2 {Pid, Ref, N} when N>0 -> ...
```

Add WeChat powcoder

Timeouts

```
1 f(Pid) ->
2   receive
3     {Pid, Msg} -> Msg
4   after 3000 ->
5     timeout
6 end.
```

- ▶ The after part will be triggered if 3000 milliseconds have passed without receiving a message that matches the pattern.
- ▶ Other uses

```
1 sleep(T) ->
2   receive
3   after T ->
4     ok
5 end.
6
7 flush() ->
8   receive
9     _ -> flush()
10  after 0 ->
11    ok
12 end.
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

- ▶ Implement a semaphore
- ▶ Use the `when` clause

Template that you can start from:

```
1 -module(semaphore).  
2 -compile([export_all]).  
3  
4 make_semaphore(Permits) ->  
5 spawn(?MODULE, semaphore, [Permits]).  
6  
7 % complete
```

<https://powcoder.com>

Add WeChat powcoder

- ▶ `?MODULE`: macro that refers to the name of the current module

A Semaphore

```
1 -module(semaphore).
2 -compile(export_all).
3
4 make_semaphore(Permits) ->
5   spawn([MODULE, semaphore, [Permits]]).
6
7 semaphore(0) ->
8   receive
9     {From, Ref, release} ->
10      semaphore(1)
11   end;
12 semaphore(P) when P>0 ->
13   receive
14     {From, Ref, release} ->
15      From ! {self(), Ref, ok},
16      semaphore(P+1),
17     {From, Ref, acquire} ->
18      From ! {self(), Ref, ok},
19      semaphore(P-1)
20   end.
```

- semaphore could be specified as a FSM

Semaphore - Print "a" before "b"

```
1 start() ->
2     S = make_semaphore(0),
3     spawn(?MODULE,p1,[S]),
4     spawn(?MODULE,p2,[S]).
5
6 release(S) -> % could be included in semaphore module
7     R = make_ref(),
8     S!{self(),R,release},
9     receive
10         {S,R,ok} ->
11             done
12     end.
13
14 p1(S) ->
15     io:format("a"),
16     release(S).
17
18 p2(S) -> % acquire is inlined
19     R = make_ref(),
20     S!{self(),R,acquire},
21     receive
22         {S,R,ok} ->
23             io:format("b")
24     end.
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Message Passing
Assignment Project Exam Help

Exceptions
<https://powcoder.com>

Links and Monitors
Add WeChat powcoder

Three Kinds of Exceptions

- ▶ Errors

- ▶ Ends the execution in the current process and includes a stack trace of the last functions

- ▶ Errors are the means for a function to stop its execution when you can't expect the calling code to handle what just happened

- ▶ Throws

- ▶ Used for cases that the programmer can be expected to handle (`try...catch`).

- ▶ Exits

- ▶ Same as errors except used to signal abnormal termination between processes.

- ▶ More lightweight than errors in that stack trace not included

Note: `try...catch` actually can catch them all

Errors – Example

Assignment Project Exam Help

```
1 1> erlang:error(badarith).
2 ** exception error: bad argument in an arithmetic expression
3 2> erlang:error(custom_error).
4 ** exception error: custom_error
5 3> catch(1+a).
6 {'EXIT',{badarith,[[{erlang,error,[1,a]]}],
7   {erl_eval,do_apply,6,[{file,"erl_eval.erl"},{line,681}]}],
8   {erl_eval,expr,5,[{file,"erl_eval.erl"},{line,434}]}],
9   {shell,exprs,7,[{file,"shell.erl"},{line,686}]}],
10  {shell,eval_exprs,7,[{file,"shell.erl"},{line,642}]}],
11  {shell,eval_loop,.,[{file,"shell.erl"},{line,627}]}]}
```

<https://powcoder.com>
Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

- ▶ Pid1 can be linked to Pid2 by calling `link(Pid2)`

- ▶ Creates a two-way link

- ▶ Terminating processes emit exit signals to all linked processes, which can terminate as well or handle the exit in some way.
- ▶ This feature can be used to build hierarchical program structures where some processes are supervising other processes, for example, restarting them if they terminate abnormally.

Note: Some comments on monitors are present at the end of these set of slides

Example

```
1 -module(linkmon).  
2 -compile(export_all).  
3  
4 myproc() ->  
5   timer:sleep(2000),  
6   exit(reason).
```

In the shell:

```
1 > c(linkmon)  
2 {ok,linkmon}  
3 > self().  
4 <0.79.0>  
5 > spawn(fun linkmon:myproc/0).  
6 <0.75.0>  
7 > self().  
8 <0.79.0>  
9 > link(spawn(fun linkmon:myproc/0)).  
10 true  
11 ** exception error: reason  
12 > self().  
13 <0.83.0>
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

```
l = spawn(fun() -> chain(N-1) end),
nk Fil),
ceive
```

Add WeChat powcoder

Add WeChat powcoder

Another Example (cont.)

```
[shell] == [3] == [2] == [1] == [0]
[shell] == [3] == [2] == [1] == *dead*
[shell] == [3] == [2] == *dead*
[shell] == [3] == *dead*
[shell] == *dead*
*dead, error message shown*
[shell] <-- restarted
```

- ▶ After the process running `linkmon:chain(0)` dies, the error is propagated down the chain of links until the shell process itself dies because of it.
- ▶ The crash could have happened in any of the linked processes
 - ▶ because links are bidirectional, you only need one of them to die for the others to follow suit.

Assignment Project Exam Help

- ▶ Links cannot be stacked
 - ▶ Calling `link/1` multiple times for the same two processes, will still create only one link between them
 - ▶ A single call to `unlink/1` will be enough to tear it down.
- ▶ `link(spawn(Function))` or `link(spawn(M,F,A))` happens in more than one step. In some cases, it is possible for a process to die before the link has been set up and then provoke unexpected behavior.
- ▶ `spawn_link/1-3` spawns and links as an atomic operation

Add WeChat powcoder

Trapping Exit Signals

Assignment Project Exam Help

- ▶ In order to be reliable, an application needs to be able to both kill and restart a process quickly.
 - ▶ Links convenient for the killing part but restarting is missing.
- ▶ When a linked process terminates, it terminates with an `exit reason` that is sent through a special message known as an `exit signal`
 - ▶ Eg. `exit signal` with `exit reason "chain dies here"`
`exit("chain dies here")`

Add WeChat powcoder

Trapping Exit Signals

Assignment Project Exam Help

- ▶ The default behaviour when a process receives an `exit` signal with an `exit reason` other than normal, is to terminate and in turn emit exit signals with the same exit reason to its linked processes.
- ▶ `System processes`: normal processes, except they can convert exit signals to regular messages.
 - ▶ Done by calling `process_flag(trap_exit, true)` in a running process.
- ▶ Allows a process to react to exit signals

<https://powcoder.com>

Add WeChat powcoder

Chain Example Revisited

Chain example with a system process at the beginning

```
1 1> process.flag(:trap_exit, true).
2 true
3 2> spawn_link(fun() -> linkmon:chain(3) end).
4 <0.49.0>
5 3> receive X -> X end.
6 {'EXIT',{0.49.0}, "chain dies here"}
```

Description of behavior:

```
[shell] == [3] == [2] == [1] == [0]
[shell] == [3] == [2] == [1] == *dead*
[shell] == [3] == [2] == *dead*
[shell] == [3] == *dead*
[shell] <-- {'EXIT',Pid,"chain dies here"} -- *dead*
[shell] <-- still alive!
```


Kill Reason

- ▶ Acts as a special signal that can't be trapped.
- ▶ Ensures any process you terminate with it will be dead.
 - ▶ A last resort, when everything else has failed.
- ▶ As the kill reason can never be trapped, it needs to be changed to killed when other processes receive the message.
 - ▶ Otherwise, every other process linked to it would in turn die for the same kill reason and would in turn kill its neighbors, and so on.
 - ▶ This explains why `exit(kill)` looks like killed when received from another linked process.

```
1 > spawn_link(fun() -> exit(kill) end).  
2 ** exception exit: killed
```

MSC for Critic Example

Judge

Critic

`{PID_J, {Band, Album}}`



`{PID_C, Answer}`



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Restarting Processes

```
1 start_critic() ->
2     spawn(?MODULE, critic, []).
3
4 judge(Pid, Band, Album) ->
5     Pid ! {self(), {Band, Album}},
6     receive
7         {Pid, Criticism} -> Criticism
8         after 2000 ->
9             timeout
10    end
11
12 critic() ->
13    receive
14        {From, {"Rage Against the Turing Machine", "Unit Testify"}} ->
15            From ! {self(), "They are great!"};
16        {From, {"System of Downtime", "Memoize"}} ->
17            From ! {self(), "They're not Johnny Crash but they're good."}
18        {From, {"Johnny Crash", "The Token Ring of Fire"}} ->
19            From ! {self(), "Simply incredible."};
20        {From, {_Band, _Album}} ->
21            From ! {self(), "They are terrible!"}
22    end,
23    critic().
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Restarting Processes

Assignment Project Exam Help

```
1 1> c(linkmon)
2 [out: linkmon]
3 2> Critic = linkmon:start_critic().
4 <0.47.0>
5 3> linkmon:judge(Critic, "Genesis", "The Lambda Lies Down on Broad
6 "They are terrible!"
```

<https://powcoder.com>

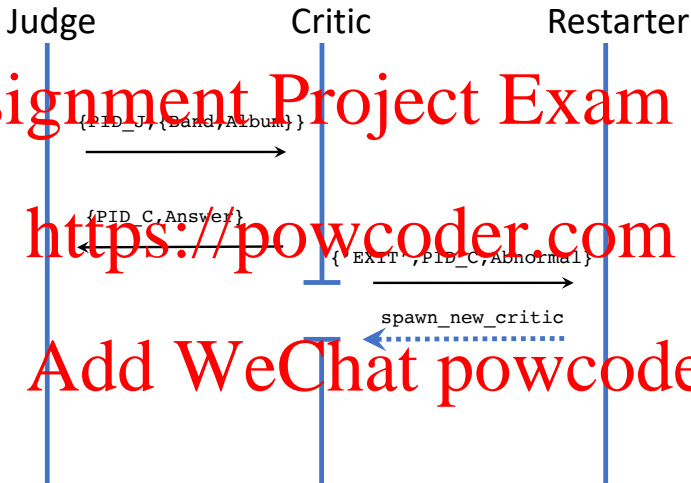
We now kill the Critic process

```
1 4> exit(Critic, solar_storm).
2 true
3 5> linkmon:judge(Critic, "Genesis", "A trick of the Tail Recursion
4 timeout
```

Add WeChat powcoder

We need a “supervisor” process to keep critics alive

MSC for Critic Example



Restarting Processes

```
1 start_critic2() ->
2 spawn(?MODULE, restarter, []).
3
4 restarter() ->
5   process_flag(trap_exit, true),
6   Pid = spawn_link(?MODULE, critic, []),
7   receive
8     {'EXIT', Pid, normal} -> % not a crash
9     ok;
10    {'EXIT', Pid, shutdown} -> % manual termination, not a crash
11    ok;
12    {'EXIT', Pid, _} ->
13  restarter()
14 end.
```

Problem: Pid of the critic is part of internal state, it is not known

```
1 1> c(linkmon).
2 {ok,linkmon}
3 2> linkmon:start_critic2().
4 <0.48.0>
5 3> linkmon:judge(?????, "Genesis", "The Lambda Lies Down on Broadw
```

Restarting Processes

- ▶ We can name a process, using an atom, rather than use its pid via `erlang:register/2`
- ▶ If a process dies, it will automatically lose its name or you can also use `unregister/1`
- ▶ You can get a list of all registered processes with `registered/0` or a more detailed one with the shell command `regs()`.

```
1 restarter() =>
2   process_flag(trap_exit, true),
3   Pid = spawn_link(?MODULE, critic, []),
4   register(critic, Pid),
5   receive
6     {'EXIT', Pid, normal} -> % not a crash
7       ok;
8     {'EXIT', Pid, shutdown} -> % manual termination, not a crash
9       ok;
10    {'EXIT', Pid, _} ->
11      restarter()
12  end.
```

What about the judge?

Restarting a Process

Assignment Project Exam Help

```
1 judge2(Band, Album) ->  
2   critic ! {self(), {Band, Album}},  
3   Pid = whereis(critic),  
4   receive  
5     {Pid, Criticism} -> {criticism  
6   after 2000 ->  
7     timeout  
8   end.
```

<https://powcoder.com>

Add WeChat powcoder

Restarting a Process

Assignment Project Exam Help

```
1 linkmon: start_critics().
2 <0.58.0>
3 2> whereis(critic).
4 <0.59.0>
5 3> linkmon:judge2("Genesis", "A trick of the Tail Recursion").
6 "They are terrible!"
7 4> exit(whereis(critic),solar_storm).
8 true
9 5> linkmon:judge2("Genesis", "A trick of the Tail Recursion").
10 "They are terrible!"
11 6> whereis(critic).
12 <0.63.0>
```

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

- ▶ `critic` is stored in a `shared` registry
- ▶ There are processes that read it such as `judge2`
- ▶ And processes that write to it such as `restarter`
- ▶ Race conditions are therefore possible

Add WeChat powcoder

Race Conditions due to Shared State – Example 1

1. critic ! Message

2. critic receives

3. critic replies

4. critic dies

5. whereis fails

6. critic is restarted

7. code crashes

```
1 judge2(Band, Album) ->  
2   critic ! {self(), {Band, Album}},  
3   %% critic dies at this point  
4   %% register still not updated  
5   Pid = whereis(critic) %% fail: (returns undefined)  
6   receive  
7     {Pid, Criticism} -> Criticism %% undefined!=Pid  
8   after 2000 ->  
9     timeout  
10  end.
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Race Conditions due to Shared State – Example 2

1. critic ! Message

2. critic receives

3. critic replies

4. critic dies

5. critic is restarted

6. whereis picks up wrong pid

7. message never matches

```
1 judge2(Band, Album) ->
2   critic {self(), [Band, Album]},
3   %% critic dies at this point
4   %% register updated with new Pid
5   Pid = whereis(critic), %% successful (but different Pid)
6   receive
7     {Pid, {criticism}} => Criticism, %% no match
8   after 2000 ->
9     timeout
10  end.
```

- Both may be solved by replacing the use of `whereis` (and `Pid` matching) to that of reference matching

Adding References to Messages

```
1 judge2(Band, Album) ->
2   Ref = make_ref(),
3   critic ! {self(), Ref, {Band, Album}},
4   receive
5     {Ref, Criticism} -> Criticism
6   after 2000 ->
7     timeout
8   end.
9
10 critic2() ->
11   receive
12     {From, Ref, {"Rage Against the Turing Machine", "Unit Testify"}} ->
13     From ! {Ref, "They are great!"};
14     {From, Ref, {"System of a Downtime", "Memoize"}} ->
15     From ! {Ref, "They're not Johnny Crash but they're good."};
16     {From, Ref, {"Johnny Crash", "The Token Ring of Fire"}} ->
17     From ! {Ref, "Simply incredible."};
18     {From, Ref, {_Band, _Album}} ->
19     From ! {Ref, "They are terrible!"}
20   end,
21   critic2().
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

Appendix: More on Exceptions

<https://powcoder.com>

Appendix: Monitors

Add WeChat powcoder

Revisiting Exceptions – How Processes Trap Them

- ▶ `spawn_link(fun() ->ok end)`

- ▶ Untrapped Result: Nothing

- ▶ Trapped Result: `{'EXIT', <0.61.0>, normal}`

- ▶ The process exited normally, without a problem.

- ▶ `spawn_link(fun() ->exit(reason) end)`

- ▶ Untrapped Result: `**exit: on exit: reason`

- ▶ Trapped Result: `{'EXIT', <0.55.0>, reason}`

- ▶ The process has terminated for a custom reason.

- ▶ `spawn_link(fun() ->exit(normal) end)`

- ▶ Untrapped Result: Nothing

- ▶ Trapped Result: `{'EXIT', <0.58.0>, normal}`

- ▶ Emulates process terminating normally.

Revisiting Exceptions

- ▶ `spawn_link(fun() ->1/0 end)`

- ▶ Untrapped Result:

- Error in process <0.44.0> with `exit` value: `{badarith, [{erlang, '/',`

- ▶ Trapped Result:

- `{'EXIT', <0.52.0>, {badarith, [{erlang, '/',`

- ▶ `spawn_link(fun() ->erlang:error(reason) end)`

- ▶ Untrapped Result:

- Error in process <0.47.0> with `exit` value: `{reason, [{erlang, apply,`

- ▶ Trapped Result:

- `{'EXIT', <0.74.0>, {reason, [{erlang, apply, 2}]}}`

- ▶ Similar to 1/0.

- ▶ `spawn_link(fun() ->throw(rocks) end)`

- ▶ Untrapped Result:

- Error in process <0.51.0> with `exit` value: `{{nocatch, rocks}, [{erlang,`

- ▶ Trapped Result:

- `{'EXIT', <0.79.0>, {{nocatch, rocks}, [{erlang, apply, 2}]}}`

- ▶ Because the throw is never caught by a try ... catch, it bubbles up into an error, which in turn bubbles up into an EXIT.

- Without trapping exit, the process fails.

Revisiting Exceptions – the `exit/2` case

Allows a process to kill another one from a distance, safely

▶ `exit(self(), normal)`

▶ Untrapped Result: `** exception exit: normal`

▶ Trapped Result: `{'EXIT', <0.31.0>, normal}`

▶ When not trapping exits, `exit(self(), normal)` acts the same as `exit(normal)`.

▶ `exit(spawn_link(fun() -> timer:sleep(50000) end), normal)`

▶ Untrapped Result: `nothing`

▶ Trapped Result: `nothing`

▶ `exit(spawn_link(fun() -> timer:sleep(5000) end), reason)`

▶ Untrapped Result: `** exception exit: reason`

▶ Trapped Result: `{'EXIT', <0.52.0>, reason}`

Assignment Project Exam Help

▶ `exit(spawn_link(fun() -> timer:sleep(50000) end), kill)`

▶ Untrapped Result: ** exception `exit`: killed

▶ Trapped Result: `{'EXIT', <0.58.0>, killed}`

▶ `exit(self(), kill)`

▶ Untrapped Result: ** exception `exit`: killed

▶ Trapped Result: ** exception `exit`: killed

▶ `spawn_link(fun() -> exit(kill) end)`

▶ Untrapped Result: ** exception `exit`: killed

▶ Trapped Result: `{'EXIT', <0.67.0>, kill}`

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

- ▶ Special type of link with two differences
 - ▶ they are unidirectional,
 - ▶ can monitor via a registered name, and
 - ▶ they can be stacked.
- ▶ <https://powcoder.com>
- ▶ Allows a process to, unobtrusively, monitor another one
- ▶ Useful for when you have multiple libraries that you call and they all need to know whether a process is alive or not
 - ▶ You can stack links and remove them individually

Add WeChat powcoder

Example

`erlang:monitor/2` sets up a monitor, where the first argument is the atom `process` and the second one is the pid

```
1> erlang:monitor(process, spawn(fun() -> timer:sleep(1000) end)).
2 #Ref<0.0.0.77>
3 2> flush().
4 Shell got {'DOWN', #Ref<0.0.0.77>, process, <0.63.0>, normal}
5 ok
```

<https://powcoder.com>

- ▶ When monitored process goes down, send message to monitor: `{'DOWN', MonitorReference, process, Pid, Reason}`.
- ▶ The reference allows you to demonitor the process.
 - ▶ Monitors are stackable, so it's possible to take more than one down.
 - ▶ References allow you to track each of them in a unique manner.

Add WeChat powcoder

Example

Atomic function to spawn process while monitoring it:

```
1 3> {Pid, Ref} = spawn_monitor(fun() -> receive _ -> exit(boom) end)
2 {<0.73.0>, #Ref<0.0.0.100>}
3 4> erlang:demonitor(Ref).
4 true
5 5> Pid ! die
6 die
7 6> flush().
8 ok
```

- ▶ We demonitored the other process before it crashed hence no trace of it dying.