Control Systems Optimization

Igor Wojnicki

AGH – Univeristy of Science and Technology

2010

Outline

Credits

- http://erlang.org/course/concurrent_programming.html

Definitions

- Process – A concurrent activity. A complete virtual machine. The system may have many concurrent processes executing at the same time.
- Message – A method of communication between processes.
- Timeout – Mechanism for waiting for a given time period.
- Registered Process – Process which has been registered under a name.
- Client/Server Model – Standard model used in building concurrent systems.

Creating a New Process

`spawn(Mod, Func, Args)`

- Concurrent execution.
- A process created by running a given function.
- Returns a PID of type Pid (A NEW TYPE).
- PID known to the parent process.
- `self()` (`erlang:self()`) current PID.
- `processes()` a list of currently running processes.
- info about process from the shell: `i()`
- `Pid=pid(0,30,1)`

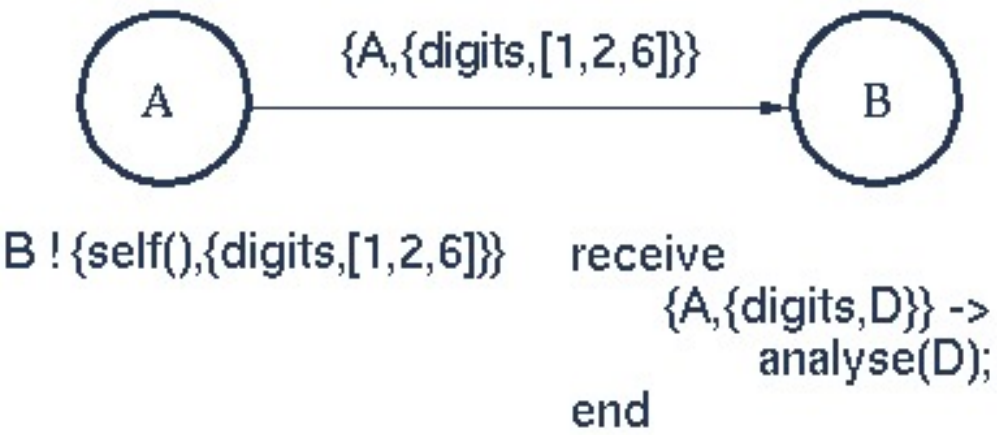
Simple Message Passing


```
B ! {self(),foo}
```

```
receive
    {From,Msg} ->
        Actions
end
```

- `From` and `Msg` become bound when the message is received.
- Sending a message never fails

Messages can carry data.



- Messages can carry data and be selectively unpacked.
- The variables `A` and `D` become bound when receiving the message.
- If `A` is bound before receiving a message then only data from this process is accepted.

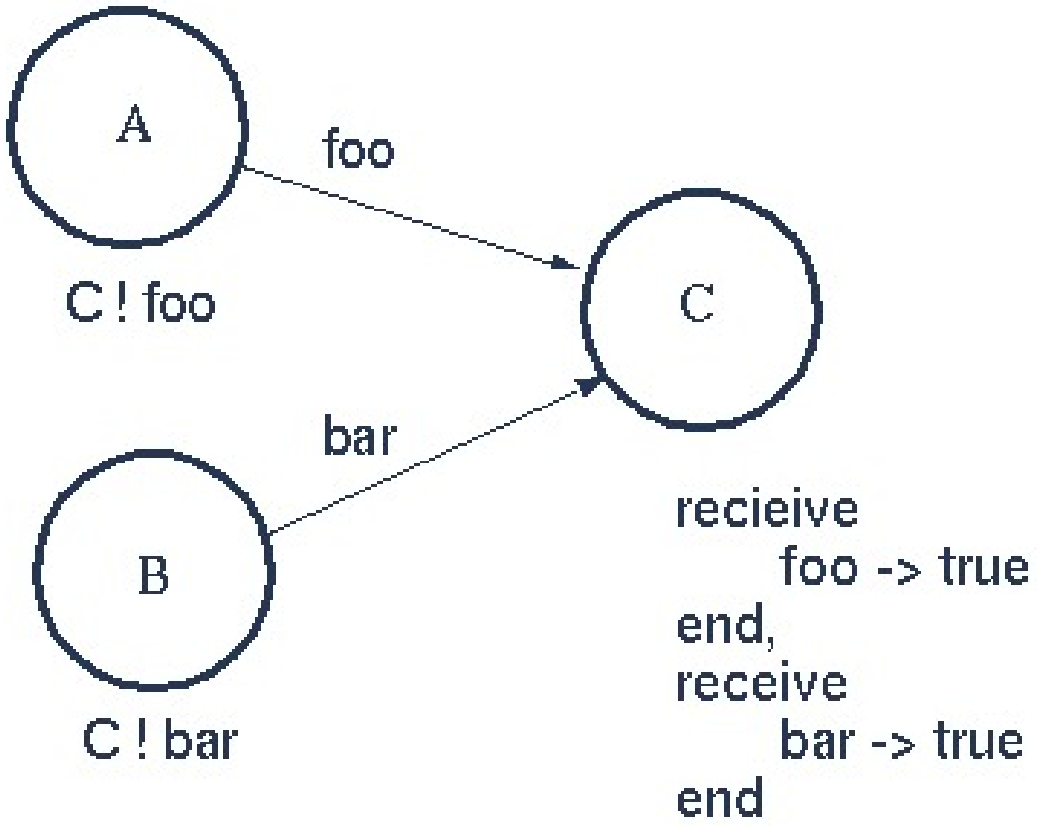
An Echo process I

```
-module(echo).
-export([go/0, loop/0]).

go() ->
        Pid2 = spawn(echo, loop, []),
        Pid2 ! {self(), hello},
        receive
                {Pid2, Msg} ->
                        io:format("P1 ~w~n",[Msg])
        end,
        Pid2 ! stop.
```

An Echo process II

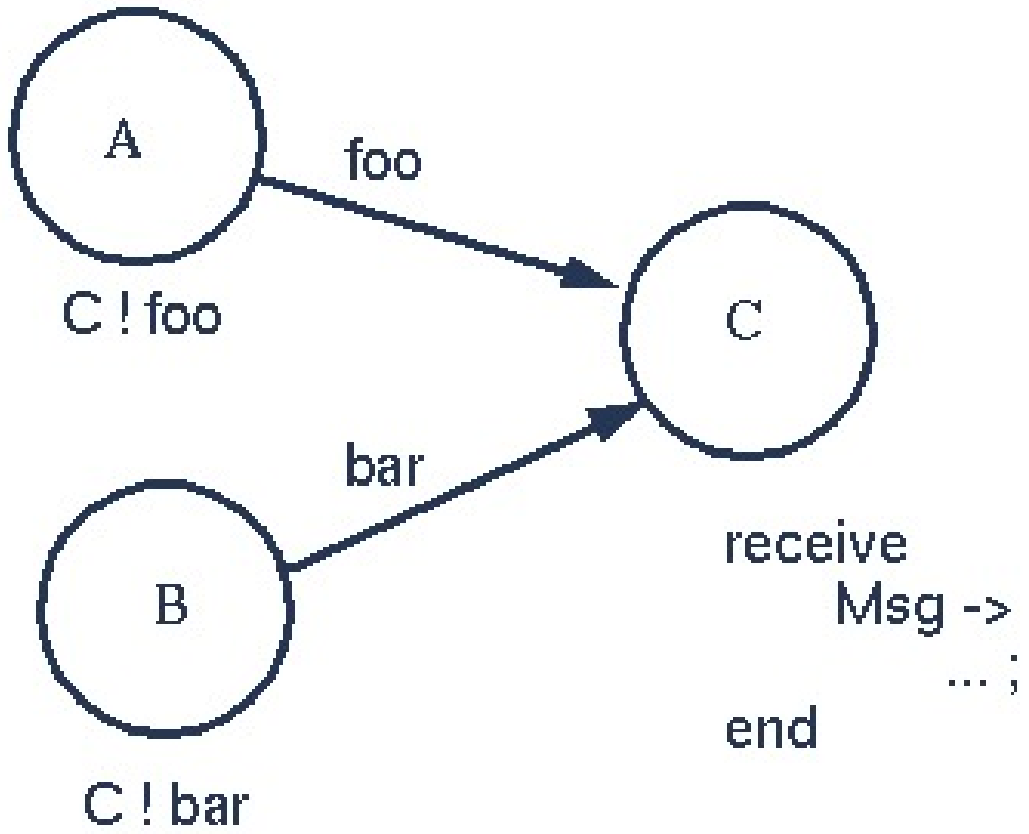
```
loop() ->
        receive
                {From, Msg} ->
                        From ! {self(), Msg},
                        loop();
                stop ->
                        true
        end.
```

Selective Message Reception



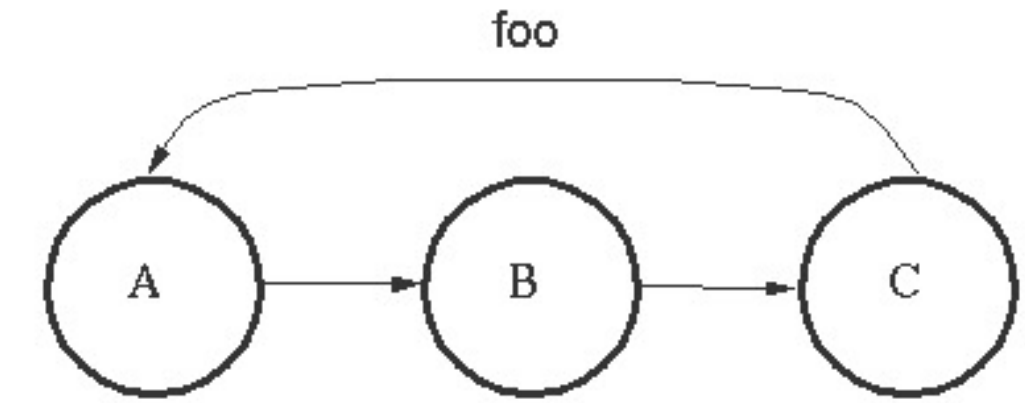
The message `foo` is received – then the message `bar` – irrespective of the order in which they were sent.

Selection of Any Message



The first message to arrive at the process `C` will be processed - the variable `Msg` in the process `C` will be bound to one of the atoms `foo` or `bar` depending on which arrives first.

Pids can be sent in messages



- A sends a message to `B` containing the Pid of `A`.
- `B` sends a transfer message to `C`.
- `C` replies directly to `A`.

Registered Processes

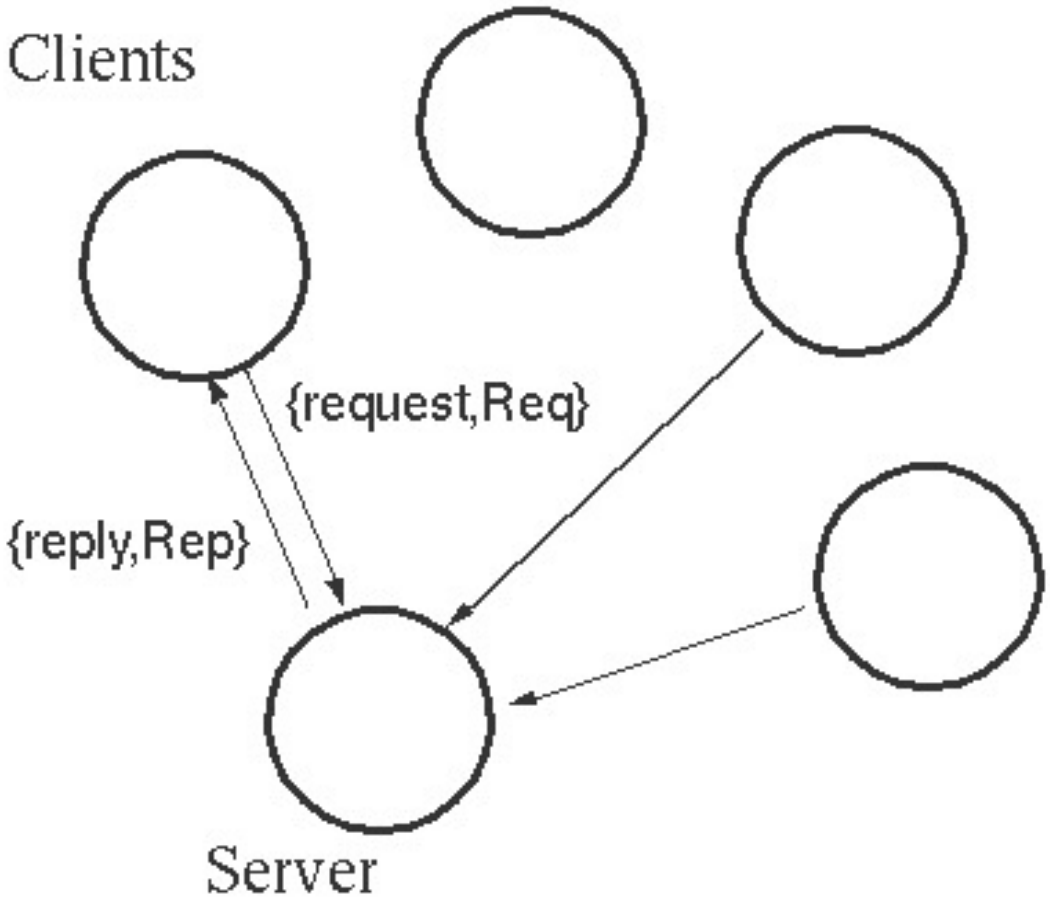
register(Alias, Pid) Registers the process Pid with the name Alias.

```
start() ->
        Pid = spawn(num_anal, server, [])
        register(analyser, Pid).

analyse(Seq) ->
        analyser ! {self(),{analyse,Seq}},
        receive
                {analysis_result,R} ->
                        R
        end.
```


Any process can send a message to a registered process.

- unregister/1
- registered/0
- whereis(Alias) - returns a PID.

Client-Server Model



Protocol

- Protocol

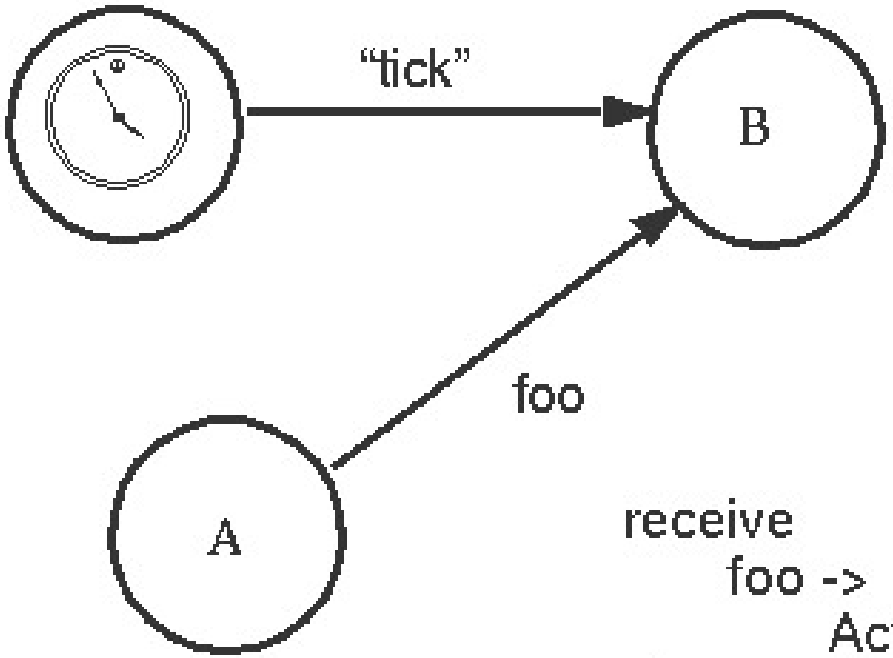
Server code

```
-module(myserver).

server(Data) ->
        receive
                {From,{request,X}} ->
                        {R, Data1} = fn(X, Data),
                        From ! {myserver,{reply, R}},
                        server(Data1)
        end.
```

Interface Library

```
-export([request/1]).

request(Req) ->
        myserver ! {self(),{request,Req}},
        receive
                {myserver,{reply,Rep}} ->
                        Rep
        end.
```

Timeouts




```
receive
      foo ->
            Actions1;
after
      Time ->
            Actions2
end
```

If the message `foo` is received from `A` within the time `Time` perform `Actions1` otherwise perform `Actions2`.

Uses of Timeouts I

- `sleep(T)` – process suspends for `T` ms.

```
sleep(T) ->
        receive
        after
                T ->
                        true
        end.
```

Uses of Timeouts II

- `suspend()` – process suspends indefinitely.

```
suspend() ->
        receive
        after
                infinity ->
                        true
        end.
```

Uses of Timeouts III

- `alarm(T, What)` - The message `What` is sent to the current process in `T` milliseconds from now

```
set_alarm(T, What) ->
        spawn(timer, set, [self(),T,What]).

set(Pid, T, Alarm) ->
        receive
        after
                T ->
                        Pid ! Alarm
        end.
```

Current process code: `receive Msg -> ... ; end`

Uses of Timeouts IV

- `flush()` – flushes the message buffer

```
flush() ->
        receive
                Any ->
                        flush()
        after
                0 ->
                        true
        end.
```


A value of 0 in the timeout means check the message buffer first and if it is empty execute the following code.

- There is a shell command `flush()`