```
-module(total).
-export([start/3]).

start(Id, Master, Jitter) ->
    spawn(fun() -> init(Id, Master, Jitter) end).

init(Id, Master, Jitter) ->
    receive
        {peers, Nodes} ->
            server(Id, Master, seq:new(Id), seq:new(Id), Nodes, [], [], Jitter)
    end.

server(Id, Master, MaxPrp, MaxAgr, Nodes, Cast, Queue, Jitter) ->
receive
    {send, Msg} ->
        Ref = make_ref(),
        request( Ref ✔ , Msg ✔ , Nodes ✔ , Jitter ✔ ),
        NewCast = [{ Ref ✔ , length(Nodes) ✔ , seq:null()}|Cast],
        server( Id ✔ , Master ✔ , MaxPrp ✔ , MaxAgr ✔ , Nodes ✔ , NewCast ✔ ,
Queue ✔ , Jitter ✔ );
    {request, From, Ref, Msg} ->
        NewMaxPrp = seq:maxIncrement(MaxPrp,MaxAgr) ✔ ,
         From ✔ ! { proposal ✔ , Ref ✔ , NewMaxPrp ✔ },
        NewQueue = queue( Ref ✔ , Msg ✔ , propsd ✔ , NewMaxPrp ✔ , Queue ✔ ),
        server( Id ✔ , Master ✔ , NewMaxPrp ✔ , MaxAgr ✔ , Nodes ✔ , Cast ✔ ,
NewQueue ✔ , Jitter ✔ );
    {proposal, Ref, Num} ->
        case proposal( Ref ✔ , Num ✔ , Cast ✔ ) of
            {agreed, MaxNum, NewCast} ->
                agree( Ref ✔ , MaxNum ✔ , Nodes ✔ ),
                server( Id ✔ , Master ✔ , MaxPrp ✔ , MaxAgr ✔ , Nodes ✔ ,
NewCast ✔ , Queue ✔ , Jitter ✔ );
            NewCast ->
                server( Id ✔ , Master ✔ , MaxPrp ✔ , MaxAgr ✔ , Nodes ✔ , NewCast ✔ ,
Queue ✔ , Jitter ✔ )
        end;
    {agreed, Ref, Num} ->
        NewQueue = update( Ref ✔ , Num ✔ , Queue ✔ ),
        {AgrMsg, NewerQueue} = agreed( NewQueue ✔ ),
        deliver( Master ✔ , AgrMsg ✔ ),
        NewMaxAgr = seq:max(MaxAgr,Num) ✔ ,
        server( Id ✔ , Master ✔ , MaxPrp ✔ , NewMaxAgr ✔ , Nodes ✔ , Cast ✔ ,
NewerQueue ✔ , Jitter ✔ );
    stop ->
        ok;
    Error ->
        io:format("Process ~w: unsupported message: ~w~n", [Id, Error])

end.

%% Sending a request message to all nodes
request(Ref, Msg, Nodes, 0) ->
    Self = self(),
```

```
    lists:foreach(fun(Node) ->
                      Node ! {request, Self, Ref, Msg}
                  end,
                  Nodes);
request(Ref, Msg, Nodes, Jitter) ->
    Self = self(),
    lists:foreach(fun(Node) ->
                      T = rand:uniform(Jitter),
                      timer:send_after(T, Node, {request, Self, Ref, Msg} )
                  end,
                  Nodes).

%% Sending an agreed message to all nodes
agree(Ref, Num, Nodes)->
    lists:foreach(fun(Node)->
                      Node ! {agreed, Ref, Num}
                  end,
                  Nodes).

%% Delivering messages to the master
deliver(Master, Messages) ->
    lists:foreach(fun(Msg)->
                      Master ! {deliver, Msg}
                  end,
                  Messages).

%% Update the set of pending proposals
proposal(Ref, PrpNum, [{Ref, 1, Max}|Rest])->
    {agreed, seq:max(PrpNum, Max), Rest};
proposal(Ref, PrpNum, [{Ref, N, Max}|Rest])->
    [{Ref, N-1, seq:max(PrpNum, Max)}|Rest];
proposal(Ref, PrpNum, [Entry|Rest])->
    case proposal(Ref, PrpNum, Rest) of
        {agreed, AgrNum, NewRest} ->
            {agreed, AgrNum, [Entry|NewRest]};
        Updated ->
            [Entry|Updated]
    end.

%% Remove all messages in the front of the queue that have been agreed
agreed([{_Ref, Msg, agrd, _Agr}|Queue]) ->
    {AgrMsg, NewQueue} = agreed(Queue),
    {[Msg|AgrMsg], NewQueue};
agreed(Queue) ->
    {[], Queue}.

%% Update the queue with an agreed sequence number
update(Ref, AgrNum, [{Ref, Msg, propsd, _}|Rest])->
    queue(Ref, Msg, agrd, AgrNum, Rest);
update(Ref, AgrNum, [Entry|Rest])->
    [Entry|update(Ref, AgrNum, Rest)].

%% Queue a new entry using Number as key
queue(Ref, Msg, State, Number, []) ->
    [{Ref, Msg, State, Number}];
queue(Ref, Msg, State, Number, Queue) ->
    [Entry|Rest] = Queue,
    {_, _, _, Next} = Entry,
    case seq:lessthan(Number, Next) of
        true ->
            [{Ref, Msg, State, Number}|Queue];
        false ->
            [Entry|queue(Ref, Msg, State, Number, Rest)]
    end.
```