Prolog Socket Programming (9A)

Young W. Lim
3/24/14

based on the following document:
http://www.csupomona.edu/~jrfisher/www/prolog_tutorial

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice/OpenOffice.

Socket Programming (9A)

Socket Programming Example (1)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Play tic tac toe with the Java GUI
%%%  using port 54321.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

connect(Port) :-
   tcp_socket(Socket),
   gethostname(Host),  % local host
   tcp_connect(Socket, Host:Port),
   tcp_open_socket(Socket, INs, OUTs),
   assert(connectedReadStream(INs)),
   assert(connectedWriteStream(OUTs)).

:- connect(54321).  % directives: Comment out for Prolog-only testing
```

tcp_open_socket(+SocketId, -InStream, -OutStream)

directive
A rule without a head
:- predicates to be executed.

Specifies the predicate to run at load time

Socket Programming Example (2)

```
ttt :-
  connectedReadStream(IStream),

  read(IStream,(X,Y)),                          %%%% read x's move

  record(x,X,Y),                                %%%% update Prolog's board
  board(B),
  alpha_beta(o,2,B,-200,200,(U,V),_Value),      %%%% look for best next move
  record(o,U,V),                                %%%% update the board

  connectedWriteStream(OStream),                %%%%  tell Java tic tac toe player

  write(OStream,(U,V)),
  nl(OStream), flush_output(OStream),
  ttt.

:- ttt.          % directives: Comment out for Prolog-only testing
```

```
read(IStream,(X,Y)),
write(OStream,(U,V)),
```

infinite loop

directive
A rule without a head
:- predicates to be executed.

Specifies the predicate to run at load time

Socket Library Predicates

tcp_socket	(-SocketId)
tcp_close_socket	(+SocketId)
tcp_open_socket	(+Socket, -Stream)
tcp_open_socket	(+SocketId, -InStream, -OutStream)
tcp_bind	(+Socket, ?Port)
tcp_listen	(+Socket, +Backlog)
tcp_accept	(+Socket, -**Slave**, -Peer)
[deprecated]**tcp_connect**	(+Socket, +Host:+Port)
tcp_connect	(+Socket, +Host:+Port, -StreamPair)
tcp_connect	(+Socket, +Host:+Port, -Read, -Write)
tcp_setopt	(+Socket, +Option)
tcp_fcntl	(+Stream, +Action, ?Argument)
tcp_host_to_address	(?HostName, ?Address)
gethostname	(-Hostname)

Socket C Library (1)

tcp_socket(-SocketId)
Creates an INET-domain stream-socket
and unifies an identifier to it with SocketId.
On MS-Windows, if the socket library is not yet initialised, this will also initialise the library.

gethostname(-Hostname)
Return the canonical fully qualified **name** of this host.
This is achieved by calling gethostname() and
return the canonical name returned by getaddrinfo().

tcp_connect(+Socket, +Host:+Port)
Connect Socket.
After successful completion, tcp_open_socket/3 can be used
to create I/O-Streams to the remote socket.
New code should use tcp_connect/4,
which can be hooked to allow for proxy negotiation.

```
tcp_socket(Socket),
gethostname(Host),  % local host
tcp_connect(Socket, Host:Port),
tcp_open_socket(Socket, INs, OUTs),
```

tcp_connect(+Socket, +Host:+Port, -StreamPair)
Client-interface to connect a socket to a given Port on a given Host.
Port is either an integer or the name of a registered service.

tcp_connect(+Socket, +Host:+Port, -Read, -Write)
Similar to tcp_connect/3, but providing separate streams.
Separate streams are hard to close safely and new code should use tcp_connect/3.

Socket C Library (2)

tcp_open_socket(+Socket, -Stream)
Create streams to communicate to Socket.
If Socket is a master socket (see tcp_bind/2), Stream should be used for tcp_accept/3.
If Socket is a connected (see tcp_connect/2) or accepted socket (see tcp_accept/3),
Stream is unified to a **stream pair** (see stream_pair/3)
that can be used for reading and writing.
The stream or pair must be closed with close/1, which also closes Socket.

tcp_open_socket(+SocketId, -InStream, -OutStream)
Similar to tcp_open_socket/2, but creates **two separate sockets**
where tcp_open_socket/2 would have created a stream pair.
Deprecated because closing a stream pair is much easier to perform safely.

```
tcp_socket(Socket),
gethostname(Host),  % local host
tcp_connect(Socket, Host:Port),
tcp_open_socket(Socket, INs, OUTs),
```

tcp_bind(+Socket, ?Port)
Bind the socket to Port on the current machine.
This operation, together with tcp_listen/2 and tcp_accept/3
implement the server-side of the socket interface.
If Port is unbound, the system picks an arbitrary free port
and unifies Port with the selected port number.
Port is either an integer or the name of a registered service.
See also tcp_connect/4.

tcp_listen(+Socket, +Backlog)
Tells, after tcp_bind/2, the socket to listen for incoming requests for connections.
Backlog indicates how many pending connection requests are allowed.
Pending requests are requests that are not yet acknowledged using tcp_accept/3.
If the indicated number is exceeded, the requesting client will be signalled
that the service is currently not available. A suggested default value is 5.

tcp_accept(+Socket, -Slave, -Peer)
This predicate waits on a server socket for a connection request by a client.
On success, it creates a new socket for the client and
binds the identifier to Slave.
Peer is bound to the IP-address of the client.

thread_create (1)

thread_create(:Goal, -Id, +Options)
Create a new Prolog thread (and underlying C thread)
and start it by executing **Goal**.
If the thread is created successfully,
the thread identifier of the created thread is unified to **Id**.
Options is a list of options.

alias	**(AliasName)**
at_exit	**(:AtExit)**
detached	**(Bool)**
global	**(K-Bytes)**
local	**(K-Bytes)**
c_stack	**(K-Bytes)**
trail	**(K-Bytes)**

thread_create (2)

alias(AliasName)
Associate an `alias name' with the thread.
This name may be used to refer to the thread
and remains valid until the thread is joined (see thread_join/2).

at_exit(:AtExit)
Register AtExit as using **thread_at_exit**/1
before entering the thread goal.
Unlike calling thread_at_exit/1 as part of the normal Goal,
this ensures the Goal is called.
Using thread_at_exit/1, the thread may be signalled or run out of resources before
thread_at_exit/1 is reached.

thread_create (3)

detached(Bool)
If **false** (default), the thread can be **waited** for using **thread_join**/2.
thread_join/2 must be called on this thread
to reclaim all resources associated with the thread.

If **true**, the system will reclaim all associated resources **automatically**
after the thread finishes.

Please note that thread identifiers are freed for reuse
after a detached thread finishes or a normal thread has been joined.
See also **thread_join**/2 and **thread_detach**/1.

If a detached thread dies due to failure or exception of the initial goal,
the thread prints a message using **print_message**/2.
If such termination is considered normal, the code must be wrapped using **ignore**/1 and/or **catch**/3 to ensure successful completion.

thread_create (4)

global(K-Bytes)
Set the limit to which the **global stack** of this thread may grow. If omitted, the limit of the calling thread is used. See also the -G command line option.

local(K-Bytes)
Set the limit to which the **local stack** of this thread may grow. If omitted, the limit of the calling thread is used. See also the -L command line option.

c_stack(K-Bytes)
Set the limit to which the **system stack** of this thread may grow. The default, minimum and maximum values are system-dependent.126

trail(K-Bytes)
Set the limit to which the **trail stack** of this thread may grow. If omitted, the limit of the calling thread is used. See also the -T command line option.

The **Goal** argument is copied to the new Prolog engine. This implies that further instantiation of this term in either thread does not have consequences for the other thread: Prolog threads do not share data from their stacks.

Server Applications

```
create_server(Port) :-
        tcp_socket(Socket),
        tcp_bind(Socket, Port),
        tcp_listen(Socket, 5),
        tcp_open_socket(Socket, AcceptFd, _),
        <dispatch>

dispatch(AcceptFd) :-
        tcp_accept(AcceptFd, Socket, _Peer),
        thread_create(process_client(Socket, Peer), _,
                [ detached(true)
                ]),
        dispatch(AcceptFd).

process_client(Socket, Peer) :-
        setup_call_cleanup(tcp_open_socket(Socket, In, Out),
                        handle_service(In, Out),
                        close_connection(In, Out)).

close_connection(In, Out) :-
        close(In, [force(true)]),
        close(Out, [force(true)]).

handle_service(In, Out) :-
        ...
```

```
tcp_open_socket(+SocketId, -InStream, -OutStream)
tcp_accept(+Socket, -Slave, -Peer)
```

The stream_pool library

add_stream_to_pool(+Stream, :Goal)
delete_stream_from_pool(+Stream)
close_stream_pool
dispatch_stream_pool(+TimeOut)
stream_pool_main_loop

The stream_pool Predicates

add_stream_to_pool(+Stream, :Goal)
Add Stream, which must be an input stream and
---on non-unix systems--- connected to a socket to the pool.
If input is available on Stream, Goal is called.

delete_stream_from_pool(+Stream)
Delete the given stream from the pool.
Succeeds, even if Stream is no member of the pool.
If Stream is unbound the entire pool is emtied but
unlike close_stream_pool/0 the streams are not closed.

close_stream_pool
Empty the pool, closing all streams that are part of it.

dispatch_stream_pool(+TimeOut)
Wait for maximum of TimeOut for input on any of the streams in the pool.
If there is input, call the Goal associated with add_stream_to_pool/2.
If Goal fails or raises an exception a message is printed.
TimeOut is described with wait_for_input/3.

If Goal is called, there is some input on the associated stream.
Goal must be careful **not to block** as this will block the entire pool.2

stream_pool_main_loop
Calls dispatch_stream_pool/1 in a loop until the pool is empty.

```
add_stream_to_pool(In, accept(Socket)),
add_stream_to_pool(In, client(In, Out, Peer)).
delete_stream_from_pool(In).
```

The stream_pool Examples

```
:- use_module(library(streampool)).

server(Port) :-
        tcp_socket(Socket),
        tcp_bind(Socket, Port),
        tcp_listen(Socket, 5),
        tcp_open_socket(Socket, In, _Out),
        add_stream_to_pool(In, accept(Socket)),
        stream_pool_main_loop.

accept(Socket) :-
        tcp_accept(Socket, Slave, Peer),
        tcp_open_socket(Slave, In, Out),
        add_stream_to_pool(In, client(In, Out, Peer)).

client(In, Out, _Peer) :-
        read_line_to_codes(In, Command),
        close(In),
        format(Out, 'Please to meet you: ~s~n', [Command]),
        close(Out),
        delete_stream_from_pool(In).
```

Read from a stream

read_line_to_codes(+Stream, -Codes)
Read the next line of input from Stream and
unify the result with Codes after the line has been read.
A line is ended by a newline character or end-of-file.
Unlike read_line_to_codes/3, this predicate removes a trailing newline character.

On end-of-file the atom end_of_file is returned. See also at_end_of_stream/[0,1].

read_line_to_codes(+Stream, -Codes, ?Tail)
Difference-list version to read an input line to a list of character codes.
Reading stops at the newline or end-of-file character,
but unlike read_line_to_codes/2, the newline is retained in the output.
This predicate is especially useful for reading a block of lines up to some **delimiter**.
The following example reads an HTTP header ended by a blank line:

Write to a stream

The format family of predicates is the most versatile and **portable way to produce textual output**.

format(+Format)
Defined as `format(Format) :- format(Format, []).'. See format/2 for details.

format(+Format, :Arguments)
Format is an atom, list of character codes, or a Prolog string.
Arguments provides the arguments required by the format specification.
If only one argument is required and this single argument is not a list,
the argument need not be put in a list. Otherwise the arguments are put in a list.

format(+Output, +Format, :Arguments)
As format/2, but write the output on the given Output.
The de-facto standard only allows Output to be a stream.
The SWI-Prolog implementation allows all valid arguments for with_output_to/2.100 For example:

```
?- format(atom(A), '~D', [1000000]).
A = '1,000,000'
```

Socket Programming (9A)

References

[1] en.wikipedia.org
[2] en.wiktionary.org
[3] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[4] http://www.learnprolognow.org/ Learn Prolog Now!
[5] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial
[6] www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/intro.html
[7] www.cse.unsw.edu.au/~billw/dictionaries/prolog/negation.html