SCTP daemons' User Application Guidelines :
Using the sctpdlib

Sim Woon Chiat,
ICM N PG U SE D2
Siemens AG

1. Introduction

This document describes how to make use of the daemon and how to program a User Application (UA) that makes use of the daemon.

2. Rationale

The main rationale of having a daemon for the SCTP library is due to the fact that there cannot be more than one instance of the SCTP library running on a host machine. Hence, an intermediary is needed to allow multiple processes to use the services provided by the SCTP library. The daemon 'multiplexes' the function calls from all the UAs, and 'demultiplexes' the return values from these calls to the respective UA. The daemon is also useful for cases when the UA is a GUI-based application and needs to run its own event loop to handle the GUI elements. Using the daemon allows the separation of the event loop of the GUI and that of the SCTP library.

The daemon comes with its own library (which will henceforth be referred to as **sctpdlib**), which provides functions that map closely to those provided by the SCTP library. To the user UA, calling these functions produce similar results to calling those functions provided by the SCTP library. For the notifications, the aim is to remain compatible with the callback API defined for the SCTP library.

3. Background

3.1. Overview

The SCTP daemon and its User Applications (UAs) run in separate processes, and they communicate through UDP messages. The messages consist of request and response pairs as well as notifications of SCTP events from the daemon to the UAs. This document will not go into the specifics of the format and fields of the messages that are passed between the two entities.

Figure 1 gives an overview of what is exchanged between the two entities. It shows the main types of messages that pass between the two.

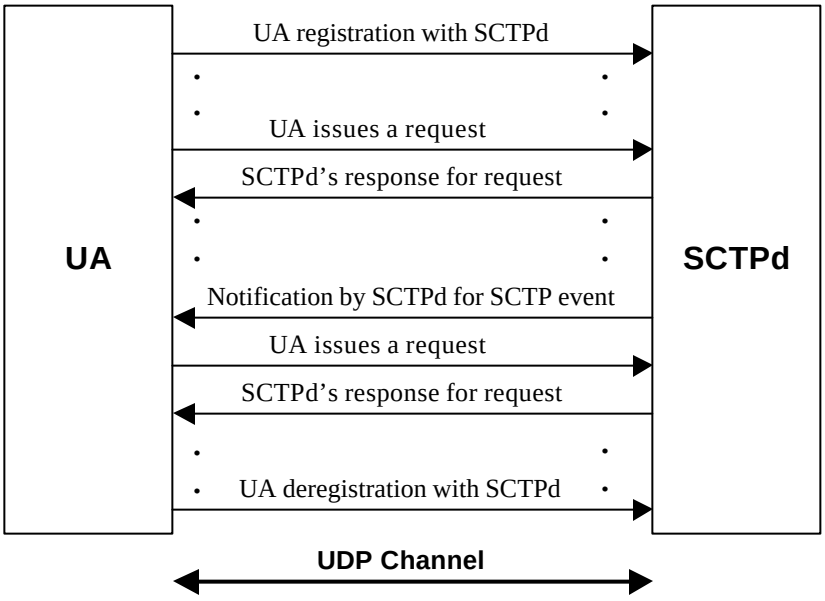

Figure 1.: Message exchanges between the User Application and the Daemon.

The first thing that the UA needs to do is to register itself with the daemon. Certain information (e.g. UDP ports information used by the UA) has to be passed to the daemon to allow the latter to communicate with it subsequently. The registration itself is through a UDP message.

After it has registered with the daemon, it can send requests to the daemon to execute certain functions calls. These functions calls are typically those that are made to the SCTP library. In this case, the UA makes calls to the **sctpdlib**, which in turns sends the relevant UDP messages to the daemon. Upon receipt of these messages, the daemon would call the SCTP library function on behalf of the UA.

Once the SCTP library function completes, the daemon would construct a message containing the results of the call. The message will than be passed back to the calling UA.

Besides the request and response messages, there are notification messages that are sent from the daemon to the UA. These notifications are mostly the SCTP events notifications that the SCTP library raises (e.g. communication up notification), which the daemon will send to the relevant UA.

The SCTP event notifications from the daemon are handled in the same way as those of the SCTP library. A callback mechanism is provided and the UA is expected to provide an event handler for each of the SCTP event that it wishes to handle. It will also need to register these callback handlers with the **sctpdlib**. [See section 5.3.]

As mentioned above, the **sctpdlib** provides a set of functions that 'mimic' those provided by the SCTP library (see **sctpdlib/ua.h** for the entire list). These functions take care of building the UDP messages and hence, the UA need not concern itself with how to

construct the appropriate messages. The return messages are also converted to the return values of these functions. In other words, the UA just need to call the functions provided by **sctpdlib** as if the application was using the SCTP library and calling those functions provided by it. For example, instead of calling,

```
associationID=SCTP_associate(sctpInstance,
                    MAXIMUM_NUMBER_OF_OUT_STREAMS,
                    destinationAddress, remotePort, NULL);
```

you now call,

```
associationID=sctpd_associate(sctpInstance,
                    MAXIMUM_NUMBER_OF_OUT_STREAMS,
                    destinationAddress, remotePort, NULL);
```

3.2. UDP Communication

This section gives an explanation of the UDP communication mechanism used.

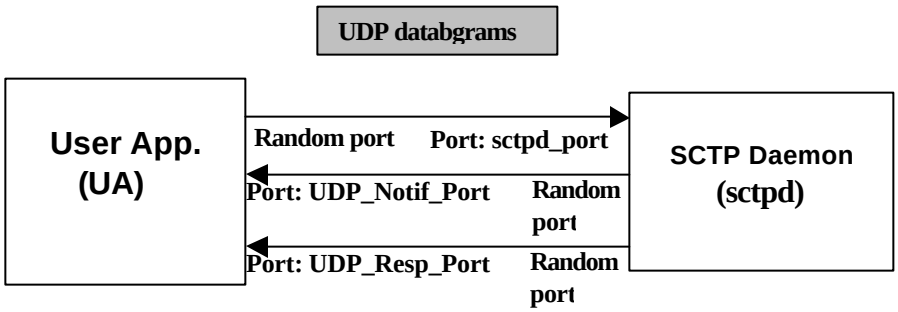

Figure 2: UDP ports needed by a UA.

The daemon binds to a UDP port (**sctpd_port**) and waits for messages from UAs. The messages that it is waiting for are of two types:

1. Administrative messages, e.g. registration messages from a new UA.
2. Function call messages, e.g. a call for sctp_receive( ).

Each UA needs to use two distinct UDP ports for its communication with the daemon. These ports should be unique for each UA.

1. **UA_Notif_Port**: UDP port to which the daemon should send notifications of SCTP event messages (e.g. communication up notification).
2. **UA_Resp_Port**: UDP port to which the daemon should send return values (responses) of function calls (requests) made by the UA to the daemon.

Note: There is a separation of responsibility in terms of who is monitoring the two UDP ports used by the UA. In this implementation, the **sctpdlib** will be monitoring the **UA_Resp_Port** after the UA has made a call to it for a function call request to be sent to

the daemon. However, the responsibility of monitoring the **UA_Notif_Port** is that of the UA, as it probably has its own event loop, and the arrival of notifications from the daemon is asynchronous.

4. **Running the daemon.**

This section covers how to use the daemon.

4.1. Building the daemon

The daemon and **sctpdlib** are part of the SCTP library distribution is built as part of the SCTP library. One point to note is that the flag **–enable-sctpd** must be included in the configure command for the SCTP library in order for the daemon and the **sctpdlib** to be built. In other words, type:

```
./configure –enable-sctpd
```

as part of the configuration and build process for the SCTP library.

4.2. Configuration

The configuration file sctpd.conf can be edited with basic configuration options, which will be loaded upon start-up of the daemon. Currently, it allows only two configuration options.

An example of the file:

MAX_UA=10
MAX_ASSOC=50

MAX_UA: Maximum number of User Applications that this daemon is to support.
MAX_ASSOC: Maximum number of associations that this daemon is to support.

4.3. Starting the daemon.

The daemon can be started in the directory sctpd/ by:

```
./sctpd
```

Note: The daemon must be started as root.

5. Using the SCTP Daemon Library

This section describes what are the steps a UA is required to perform in order to use the daemon. The example program daytime_simple.c is used to illustrate the concepts.

5.1. Header File

The UA must include the header file **ua.h** provided by the library. This header file defines all the functions that are available to the UA through which the functionality of the daemon can be accessed.

5.2. Initializing the **sctpdlib**

In order to begin using the daemon, the UA must initialize the **sctpdlib**. This is done through the call

```
sctpd_initLibrary (UDP_Notif_Port, UDP_Resp_Port);
```

The UDP_Notif_Port is the port to which the daemon should send notifications to, and the UDP_Resp_Port is the port the library should use to receive responses from the daemon for the requests it sends out (see section 3.2.). The initialization will register the UA with the daemon, informing it of the port numbers that the UA is using. After the registration, the UA can send requests to the daemon, but not yet receive any notifications from the daemon. As per the SCTP library, an application needs to register an instance with the **sctpdlib** before any notifications will be sent.

5.3. Registering an Instance

In parallel to how an application, which uses the SCTP library, needs to register an instance with the library, so a UA must register an instance with the **sctpdlib**. This is done with a call to ***sctpd_registerInstance ( )***. The parameters are the same as those which is provided to **sctp_registerInstance( )** in the SCTP library, except for the last parameter which is of type **UDP_UA_CB** (defined in the **sctpdlib**) instead of type **SCTP_ulpCallbacks**. Besides other information, the data structure **UDP_UA_CB** holds the pointers to the callback functions that the UA wished to be called upon the arrival of notifications from the daemon, so it is used identically as **SCTP_ulpCallbacks**.

In the daytime_simple.c example,

```
UDP_UA_CB udp_callbacks;

/* Register the UDP callbacks with the dispatch mechanism */
udp_callbacks.communicationUpNotif = &communicationUpNotif;
udp_callbacks.shutdownCompleteNotif = &shutdownCompleteNotif;
udp_callbacks.networkStatusChangeNotif = &networkStatusChangeNotif;
```

The example is only interested in the communication up, shutdown and network status change events. Hence, it defined three event-handling functions and assigned them to a data structure UDP_UA_CB. Note that these are the same steps taken when using the SCTP library.

5.4. Listening for notifications from the daemon

There are two ways in which the UA can deal with the notifications from the daemon.

5.4.1. Using **sctpd_eventLoop()**

The **sctpdlib** provides a function, **sctpd_eventLoop()** that listens on the specified **UDP_Notif_Port** for incoming notifications from the daemon. It will call a dispatch function in the **sctpdlib** to call the appropriate user-registered callback functions. This is very similar to using the **sctp_eventLoop()** function provided by the SCTP library.

5.4.2. Using a custom event loop.

For some applications, there is more than one source of input, e.g. a GUI-based program. For such a case, the UA can use its own event loop and it is the UA's responsibility to listen for UDP data arriving on **UDP_Notif_Port**.

When UDP data arrives on the port, it should be read and passed to the dispatch function (dispatchUAMesg()) provided by **sctpdlib.** This function parses the data into a message data structure used by the **sctpdlib**, and calls the appropriate user-registered callback functions provided by the UA.

In the case of the **sctpdaytime** program (source code is in **daytime.c**), the gdk event loop is used to monitor the arrival of UDP data and a callback function is registered with the gdk event loop for the event that UDP data has arrived for it:

```
gdk_input_add(udpfd,GDK_INPUT_READ,udp_recv_callback,NULL);
```

where udpfd is the file descriptor of the socket bound to the **UDP_Notif_Port**, and **udp_recv_callback** is the UA-defined routine to be called upon the arrival of UDP data. In **daytime.c** :

```
void udp_recv_callback (gpointer data,gint source,GdkInputCondition
condition) {

  char recvbuff[256];
  socklen_t len;
  struct sockaddr_in taddr;
  ssize_t mesglen;
```

```
  mesglen = recvfrom(source,recvbuff,sizeof(recvbuff),0,(struct
sockaddr *) &taddr,&len);
  if (mesglen>0) {
    /* call the dispatcher provided by sctpdlib which will
       call the registered UDP callback functions. */
    dispatchUAMesg(recvbuff);
  }
}
```

Notice that the function dispatchUAMesg(recvbuff) provided by the **sctpdlib** is called within the callback function. This passes the received UDP data to the dispatch function for processing. By examining the received UDP data, the respective callback functions that the UA has registered with the **sctpdlib** will be called.

5.5. Using the functions provided by the sctpdlib

As mentioned, the **ua.h** header file defines all the function calls that the library provides to the UA to send messages to the daemon. These functions are very similar to those provided by the SCTP library. Any differences will be commented in **ua.h**.

5.6. Heartbeats

Since the daemon and the UAs are loosely bound, there needs to be a mechanism through which the daemon may detect the abnormal termination of the UA without a proper deregistration (e.g. when a UA dies unexpectedly). This is necessary so that the daemon can maintain a clean state and continue to run stably.

The mechanism chosen is that of a heartbeat mechanism. This mechanism requires no action of the part of the UA and is wholly contained within the **sctpdlib**.

Periodically, the daemon will send out heartbeat messages to the UAs that are registered with it, and it expects these UAs to respond with acknowledgments. A callback in the library provides these acknowledgements. If no acknowledgements are received before timeout, the state of the UA will be cleared within the daemon, and all outstanding associations that the daemon has on behalf of the offending UA will be shut down.

6. **Using the example programs.**

A set of example programs has been provided with the sctpdlib under the directory sctpd_programs/. All the test programs need not be started as root.

6.1. daytime_simple

This is a simple non-graphical daytime server. It provides the current system day and time to any SCTP client that connects to it. It is started by:

```
./daytime_simple –s <source IP address> ...
```

6.2. sctpd_echo_tool

This is the echo tool from the SCTP library, converted to using the daemon. Use it as you would the original.

6.3. sctpdaytime_server

This is a simple graphical daytime server. It provides the current system day and time to any SCTP client that connects to it. It is started by:

```
./sctpdaytime_server –s <source IP address> ...
```

6.4. sctpd_terminal

This is a graphical terminal program that allows the user to connect to a remote SCTP server. It is started by:

```
./sctpd_terminal –s <source IP address> ...
```

6.5. echo_server_monitor

This is an echo_server with basic monitoring capabilities. It monitors the association and path information and provides notifications if any paths go inactive. It is started by:

```
./echo_server_monitor –s <source IP address> ...
```