TCPIP

COLLABORATORS

	TITLE : TCPIP	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

TCPIP

1.1 TCPIP.guide

```
TCP/IP
******

   This is an introduction to TCP/IP programming with Miami, and part of
the Miami Native Development Kit. The text is not a complete,
self-contained manual, but instead is an add-on to an existing TCP/IP
textbook for Unix. This documentation describes the differences between
Unix TCP/IP programming and TCP/IP programming for Miami, and gives the
Amiga TCP/IP programmer some additional hints.

   This text assumes that the reader has access to the book "Unix
Network Programming. Networking APIs: Sockets and XTI", a very
up-to-date and excellent textbook on TCP/IP programming. For more
information on this book please see Unix Network Programming.


  Textbook reference                Information on the reference text
  General                           General remarks and hints
  Chapter 1                         Chapter 1: Introduction
  Chapter 2                         Chapter 2: The Transport Layer: TCP and UDP
  Chapter 3                         Chapter 3: Sockets Introduction
  Chapter 4                         Chapter 4: Elementary TCP Sockets
  Chapter 5                         Chapter 5: TCP Client-Server Example
  Chapter 6                         Chapter 6: I/O Multiplexing: The select and  ↩
     poll Functions
  Chapter 7                         Chapter 7: Socket Options
  Chapter 8                         Chapter 8: Elementary UDP Sockets
  Chapter 9                         Chapter 9: Elementary Name and Address  ↩
     Conversions
  Chapter 10                        Chapter 10: IPv4 and IPv6 Interoperability
  Chapter 11                        Chapter 11: Advanced Name and Address  ↩
     Conversions
  Chapter 12                        Chapter 12: Daemon Processes and inetd  ↩
     Superserver
  Chapter 13                        Chapter 13: Advanced I/O Functions
  Chapter 15                        Chapter 15: Nonblocking I/O
  Chapter 16                        Chapter 16: ioctl Operations
  Chapter 17                        Chapter 17: Routing Sockets
```

```
  Chapter 18                          Chapter 18: Broadcasting
  Chapter 19                          Chapter 19: Multicasting
  Chapter 20                          Chapter 20: Advanced UDP Sockets
  Chapter 21                          Chapter 21: Out-of-Band Data
  Chapter 24                          Chapter 24: IP Options
  Chapter 25                          Chapter 25: Raw Sockets
  Chapter 26                          Chapter 26: Datalink Access
  Chapter 27                          Chapter 27: Client-Server Design Alternatives
```

Chapter 14 (Unix Domain Protocols), chapter 22 (Signal-Driven I/O), chapter 23 (Threads) and chapters 28-34 (XTI: X/Open Transport Interface) contain information that is not applicable to the current AmigaOS BSD-Socket API.

1.2 TCPIP.guide/NODE_TEXTBOOK

```
Textbook reference
******************
```

The textbook required to use this manual is:

```
  Unix Network Programming
  Networking APIs: Sockets and XTI
  Volume 1, Second Edition

  Author: W. Richard Stevens
  Prentice Hall PTR, 1998
  ISBN 0-13-490012-X
```

This book is the long-awaited new edition of the author's very popular earlier book 'Unix Network Programming'. When you buy the book be sure that you buy the new second edition (volume 1), NOT the older book 'Unix Network Programming'. That older book has by now become rather obsolete.

The author, W. Richard Stevens, is one of the most reknowned experts on TCP/IP. In addition to the two books mentioned above he is also author and co-author of the "TCP/IP Illustrated" book series, and author and co-author of several RFCs, Internet draft documents and scientific papers on TCP/IP and Internet issues.

This new textbook not only covers the 'usual' topics, such as UDP and TCP client/server programming, raw sockets, ICMP, broadcasting etc., but also contains detailed information on many new topics, such as multicasting, T/TCP, IPv6 and protocol-independent software development.

The book is highly recommended to anyone who is serious about writing professional TCP/IP applications. Parts of it might be too technical and in-depth for the casual hobby programmer, though. For those people a more basic book about TCP sockets with lots of examples (e.g. from Sybex or Addison Wesley), without the more complex information regarding multicasting or IPv6 might be more advisable.

For more information on the text please visit the author's homepage:

http://www.kohala.com/~rstevens

That page also contains the source code for all example programs in the book, plus the current errata.

1.3 TCPIP.guide/NODE_GENERAL

General
****

The following differences between Amiga TCP/IP programming and Unix TCP/IP programming are important. The textbook uses the Unix conventions, so you need to manually convert example programs to the corresponding Amiga conventions:

* Amiga BSD sockets are, unlike Unix sockets, NOT valid level-1 file descriptors, i.e. the level-1 file I/O functions of your C compiler's library CANNOT be used with sockets. Functions that are not allowed on sockets include 'read', 'write', 'close', 'ioctl' and several others. If you are porting code from Unix, and that code uses these functions to access sockets, then you need to use the following replacement functions (ONLY when the program accesses sockets, NOT when it accesses real level-1 files):

  - Replace 'read(a,b,c)' with 'recv(a,b,c,0)'.

  - Replace 'write(a,b,c)' with 'send(a,b,c,0)'.

  - Replace 'close(a)' with 'CloseSocket(a)'.

  - Replace 'ioctl(a,b,c)' with 'IoctlSocket(a,b,c)'.

* For the same reason it is NOT possible to convert a socket into a level-2 file descriptor (fdopen()), or to use socket functions (like select()) on level-1 file descriptors. The number spaces of sockets and level-1 file descriptors are NOT related.

* The text makes extensive use of socket wrapper functions defined by the author, which perform some additional error checking, as described in section 1.4 of the text. Usually the names of the wrapper functions are derived from the original function names, by capitalizing the first character (e.g. the wrapper for 'socket' is named 'Socket'). One problem with this in AmigaOS is that AmigaOS function names also start with a capital letter, and some AmigaOS function names clash with the names of socket wrapper functions (e.g. 'Close', which is already used by dos.library). One way to avoid this problem is by adding a prefix to those wrapper function names that cause clashes, e.g. 's_Read' instead of 'Read', 's_Write' instead of 'Write' and 's_Close' instead of 'Close'. This is the method that was used to get the example programs to compile with SAS/C.

* In Unix programs you will often find that an errno code EINTR is ignored, i.e. that programs explicitly check for (errno==EINTR), and in this case 'continue' back to the beginning of the loop, or bypass error processing. In Unix this is correct, because EINTR indicates that the function call was interrupted (a transient condition). However in AmigaOS (errno==EINTR) indicates that the user has pressed Ctrl-C. This is something you should NOT ignore in your program, i.e. you need to remove that 'if(errno==EINTR) continue;' code from any Unix programs you port. Either have your program exit gracefully or handle this condition in some other way, but do NOT ignore it. Users will be very unhappy if your program does not react to Ctrl-C.

* The text makes use of several Unix-specific functions regarding signals, e.g. alarm(), sigsetjmp() etc. These functions do not exist in AmigaOS, so the corresponding programs have to be rewritten.

1.4 TCPIP.guide/NODE_CHONE

```
Chapter 1. Introduction
**********************

1.1 Introduction
================
```

The text mentions IPv6. At the time of this writing no AmigaOS protocol stack with IPv6 support exists. However current versions of the Miami API have been prepared for future IPv6 plug-ins into Miami and/or Miami Deluxe, and by following the guidelines in this manual it is possible for you to write TCP/IP applications now in such a way that they will automatically be IPv6-compatible later, once IPv6 becomes available for AmigaOS, without recompiling the applications again.

```
1.2 A Simple Daytime Client
===========================
```

The example given in this section (and many other examples in the book) use 'read' instead of 'recv' for socket I/O. This is not possible in AmigaOS. Please see General Information for more information on this.

The indented text on page 8 mentions a function 'inet_pton()'. This function is NOT part of the standard AmigaOS BSD socket API. However it IS supported by Miami and the miami.library API for Miami 3.0 and higher. For more information on how to make use of the new functions in Miami's API without sacrificing compatibility to other TCP/IP protocol stacks (e.g. AmiTCP/IP) please see the enclosed README file.

```
1.3 Protocol independence
=========================
```

The IPv6 example given in the text can be compiled with the Miami SDK, but will not work with Miami until IPv6 support becomes available.

1.4 Error Handling: Wrapper Functions

Please see General Information for more information on name clashes with some of the wrapper functions.

1.5 A Simple Daytime Server

The function 'snprintf' mentioned in the indented text is not part of the standard SAS/C runtime library, but the author's implementation that comes with the example programs can be used instead.

The paragraph 'Terminate connection' mentions the Unix 'fork' function to create concurrent servers. This cannot be done in AmigaOS. Unlike processes in Unix, processes in AmigaOS do NOT share resources (e.g. sockets), making the use of concurrent servers in AmigaOS through separate processes much more complicated. Please read the README file (on multiple bsdsocket openers) and see the description of 'ObtainSocket' and 'ReleaseSocket' in Socket.doc for more information how sockets can be passed between AmigaOS processes.

1.6 Road Map to Client-Server Examples in the Text

Some of the examples listed here use techniques like fork() and are thus not applicable to AmigaOS.

1.8 BSD Networking History

Amiga protocol stacks fit into this history in the following way:

* AS-225 R1 is based on 4.2BSD.

* AS-225 R2 is probably based on the same code, but there are some indications that at least parts of the code have been upgraded to 4.3BSD Tahoe (Net/1).

* AmiTCP/IP <=4.x seems to be based partly on 4.3BSD Tahoe and partly on 4.3BSD Reno. It still has the old routing scheme, but already uses the new sockaddr format.

* TermiteTCP: unknown, probably <= 4.3BSD Reno.

* Miami: Very early beta versions were based on 4.4BSD-Lite (Net/3). Version 1.0 and higher were based on 4.4BSD-Lite2. Version 2.1 and higher are based on FreeBSD. Current versions of Miami (3.0 and higher) are primarily based on FreeBSD 2.2.5, but also use code from 4.4BSD-Lite2 and OpenBSD.

1.9 Test Networks and Hosts

The diagnostic programs 'netstat', 'ifconfig' and 'ping' are also available for Miami. They are called 'MiamiNetStat', 'MiamiIFConfig' and 'MiamiPing' and located in the directory 'Miami:'. They are used as

described in the text.

1.10 Unix Standards
==========

   AmigaOS is for the most part NOT Posix-compliant. DNI/XTI is not supported, neither is AF_LOCAL or AF_UNIX. Many of the new Posix.1.g functions for networking are available in the API of Miami 3.0 though.

1.11 64-bit Architectures
==========

   AmigaOS is currently a 32-bit architecture, following the ILP32 model.

1.5 TCPIP.guide/NODE_CHTWO

Chapter 2. The Transport Layer: TCP and UDP
****

2.2 The Big Picture
==========

   'tcpdump' is also available for Miami. The name is 'MiamiTCPDump'. It is based on BPF (implemented in miamibpf.library). DLPI is not supported by Miami.

2.7 Port numbers
==========

   Miami currently uses the BSD model for allocation of port numbers (see figure 2.6).

   The 'rresvport' function mentioned in the indented text is not currently part of the AmigaOS BSD socket API.

2.10 Standard Internet Services
==========

   Instead of a file '/etc/services' Miami uses a built-in 'Services' database. See also section 9.9.

1.6 TCPIP.guide/NODE_CHTHREE

Chapter 3. Sockets Introduction
****

3.1 Introduction
==========

The AmigaOS BSD socket API only supports the functions 'inet_addr' and 'inet_ntoa'. The more advanced functions 'inet_aton', 'inet_pton' and 'inet_ntop' are supported by the Miami API though. Also see sections 3.6 and 3.7.

3.2 Socket Address Structures
==========

The AmigaOS BSD socket API supports the sa_len/sin_len member with all compatible protocol stacks, including Miami. However AS-225 (using the old 'socket.library' API) does NOT support this member.

The sockaddr_un structure is not supported, because it refers to the unsupported 'Unix' socket domain.

3.4 Byte Ordering Functions
==========

AmigaOS running on MC680x0 CPUs uses big-endian byte ordering, i.e. the conversion functions (htonl() etc.) are no-ops. You should still be careful to add those functions where required, though, or otherwise your code will break if AmigaOS is ever ported to little-endian machines.

3.9 readn, writen, and readline Functions
==========

Note that the EINTR behavior described in the text does NOT apply to AmigaOS. Please see General Information for more information on this.

3.10 isfdtype Function
==========

This function does not exist in AmigaOS, and would be useless anyway, because the number spaces of level-1 file descriptors and sockets are separate.

1.7 TCPIP.guide/NODE_CHFOUR

Chapter 4. Elementary Sockets
****

4.2 socket Function
==========

Miami does not support AF_LOCAL, AF_UNIX, AF_KEY, AF_NS, AF_ISO, or (at the moment) AF_INET6. AF_ROUTE is only supported by Miami, not by older protocol stacks. SOCK_SEQPACKET and SOCK_PACKET are not supported either.

4.5 listen Function
==========

Miami currently uses a listen backlog policy similar to the one used

by SunOS 4.1.4, with one exception: For applications that specify a backlog parameter of 5 the corresponding 'real' listen backlog is configurable in Miami (MiamiSysCtl parameter 'socket.maxqlen'). The default is 7.

This allows 'normal' servers, which have been compiled with listen(...,5) to use larger listen backlogs, controlled by the administrator.

4.7 fork and exec Functions
==========

This section is not applicable to AmigaOS.

4.8 Concurrent Servers
==========

This section is not directly applicable to AmigaOS. It is possible to write concurrent servers for AmigaOS, but not using the methods described here. To service an incoming connection in a separate process you need to create a new process (CreateNewProcTags()), open 'bsdsocket.library' and 'miami.library' (if needed) again from that process, and then pass the socket from the main process to the child, using the functions 'ReleaseSocket' and 'ObtainSocket'. You may NOT share library bases or socket numbers between the main process and the child.

4.9 close Function
==========

You need to use CloseSocket() for sockets, NOT close().

1.8 TCPIP.guide/NODE_CHFIVE

Chapter 5. TCP Client-Server Example
****

5.6 Normal Startup
==========

The description of the output of "ps -l" is not applicable to AmigaOS.

5.8-5.10
==========

These sections are not applicable to AmigaOS. If you spawn child processes then you need to set up your own notification system.

5.13 SIGPIPE Signal
==========

The SIGPIPE signal does not exist in AmigaOS. If a process tries to write to a socket after the socket has received a RST then EPIPE is

returned in errno.

5.14 Crashing of Server Host
============================

   The last paragraph in this section and the discussion in section 5.15 (about how clients discover when the server has crashed) is VERY important. The majority of Amiga networking software is broken in this respect. Please check your software for this condition.

5.15 Shutdown of Server Host
============================

   Server shutdowns in AmigaOS are usually initiated by sending a Ctrl-C signal to all servers. Servers need to watch out for this, and exit gracefully when receiving a Ctrl-C signal.

1.9 TCPIP.guide/NODE_CHSIX

Chapter 6: I/O Multiplexing: The select and poll Functions
****

6.2 I/O Models
==============

   'blocking I/O', 'nonblocking I/O' are both possible with AmigaOS.

   'I/O multiplexing' is also possible, but only using the function select() (and its AmigaOS extension WaitSelect()), not using poll().

   'signal driven I/O' is possible as well, but in a different way than described in the text. Signal driven I/O in AmigaOS uses exec.library signal bits and the Wait() function, not signal handlers.

   'asynchronous I/O' is not currently possible. Mind the terminology: in AmigaOS and BSD-Socket documentation you will often find the term 'asynchronous socket I/O' as a synonym for what this text describes as 'signal driven I/O'. True 'asynchronous I/O' with the meaning described in this text is not currently possible with AmigaOS.

   'signal-driven I/O' in AmigaOS works in one of two ways:

   First method:

   * You specify which exec signal you want to receive for socket events by setting SBTC_SIGIOMASK using SocketBaseTags.

   * You set the FIOASYNC flag on the socket.

   * Now you will receive the specified signal every time an event occurs on the socket.

   Second method:

* You specify which exec signal you want to receive for socket events by setting SBTC_SIGEVENTMASK using SocketBaseTags.

* You use SO_EVENTMASK to define which events you want to receive for the socket.

* Now you will receive the specified signal every time an event occurs on the socket. Whenever you receive the signal you need to retrieve all events by calling GetSocketEvents().

Note that the second method does not work with all versions of all Amiga protocol stacks.

For more information please see the detailed description of all functions in Socket.doc.

6.3 select Function

The select() function can be used in AmigaOS as described, with a few exceptions:

* Only socket descriptors can be used in the fd sets, not level-1 file descriptors.

* NEVER restart select() after (errno==EINTR). That condition indicates that the user has hit Ctrl-C, NOT that the function should be restarted.

* You might want to use the function WaitSelect() instead of select(). WaitSelect() allows you to not only wait for socket events, but also for exec signals.

The maximum number of descriptors for an fd set is defined in the preprocessor constant FD_SETSIZE. You can change this value in the following way:

* #define FD_SETSIZE to the value you need before including sys/socket.h into your program.

* Then call SocketBaseTags with SBTC_DTABLESIZE and your chosen value as the FIRST thing in your program.

6.4 str_cli Function (Revisited)

The technique described in this section and in section 6.7 (using select() for both file descriptors AND sockets) does not work in AmigaOS.

6.8 TCP Echo Server (Revisited)

This section describes how a server can serve multiple client connections with only a single process. This technique also works with AmigaOS, and is often easier to implement than a server that spawns child tasks.

```
6.9-6.11
========

   These sections are not applicable to AmigaOS.
```

1.10 TCPIP.guide/NODE_CHSEVEN

```
Chapter 7. Socket Options
*************************

7.1 Introduction
================

   The 'fcntl' function does not exist for sockets in AmigaOS.  Instead
of 'ioctl' you need to use 'IoctlSocket' for sockets.  Please see
figure 7.15 for more information.

7.2 Checking if an Option is Supported and Obtaining the Default
================================================================

   Using '#ifdef' to check if an option is supported, as described in
the text, is NOT sufficient, because different protocol stacks support
different options. Instead you need to check at runtime whether an
option is supported, by examining the return code of getsockopt() or
setsockopt() (-1 and errno==ENOPROTOOPT for unsupported options).

   You will find that Miami supports a much larger set of options than
most other protocol stacks. If you want your program to work with
other, older protocol stack then your program should not rely on
certain options being supported by all protocol stacks.

7.5 Generic Socket Options
==========================

   Miami supports all options described in this section (including
SO_REUSEPORT), except that SO_DEBUG has no useful effect.

7.6 IPv4 Socket Options
=======================

   Miami supports all options described in this section except for
IP_RECVIF.

7.7-7.8
=======

   These sections are not currently applicable to any AmigaOS protocol
stack.

7.9 TCP Socket Options
======================

   Miami supports all options described in this section except for
```

TCP_MAXRT and TCP_STDURG.

7.10 fcntl Function
==========

   This section is not applicable to AmigaOS.

1.11 TCPIP.guide/NODE_CHEIGHT

Chapter 8. Elementary UDP Sockets
****

8.11 connect function with UDP
==========

   The DNS example given mentions the file '/etc/resolv.conf' for Unix. In Miami the corresponding configuration mechanism is the built-in 'DNS servers' database.

1.12 TCPIP.guide/NODE_CHNINE

Chapter 9. Elementary Name and Address Conversions
****

9.2 Domain Name System
==========

   Instead of the files '/etc/resolv.conf' and '/etc/hosts' Miami uses the built-in databases 'DNS servers' and 'Hosts'. Other mechanisms (e.g. NIS) are not supported. gethostbyname() and gethostbyaddr() automatically check all applicable tables, databases and DNS servers.

9.4 RES_USE_INET6 Resolver Option
==========

   Miami does not currently support IPv6, so this option is not supported, even though it appears in the header files. If you want to use it anyway, for future compatibility, then use the following calling sequence:

```
     if(MiamiSupportsIPV6()) {
         /* save a flag in your application to remember that
            IPv6 is used */
         ipv6=TRUE;

         MiamiResSetOptions(MiamiResGetOptions()|RES_USE_INET6);
     }
```

   For AmigaOS TCP/IP programs the resolver data (variable _res) is in a shared library, not in a linked library, i.e. that variable and

resolver functions cannot be accessed in the way described in the text.

   The other methods described (global options or environment variables) are not supported.

9.5 gethostbyname2 Function
==========

   gethostbyname2 is not supported by the standard AmigaOS BSD-socket API, but by the Miami API.

9.7 uname Function
==========

   The uname function is not currently supported in AmigaOS.

9.9 getservbyname and getservbyport Functions
==========

   With Miami these two functions refer to the built-in database 'Services' instead of the Unix file '/etc/services'.

9.10 Other Networking Information
==========

   The use of the getXXXent, setXXXent and endXXXent functions is possible with Miami, but deprecated. The keyed lookup functions (see figure 9.9) should be used instead.

1.13 TCPIP.guide/NODE_CHTEN

Chapter 10. IPv4 and IPv6 Interoperability
****

   IPv6 is not currently supported by Miami. However the comments in this chapter will be applicable once IPv6 gets available.

1.14 TCPIP.guide/NODE_CHELEVEN

Chapter 11. Advanced Name and Address Conversions
****

   The functions getaddrinfo, gai_strerror, freeaddrinfo and getnameinfo described in this chapter are not supported by the standard AmigaOS BSD-socket API, but by the Miami API.

   Currently these functions only support IPv4, but they will be transparently extended to support IPv6 once IPv6 becomes available. If your application uses these functions correctly now then it will not have to be changed when IPv6 becomes available later.

```
11.5 getaddrinfo Function: IPv6 and Unix Domain
===============================================

   Miami does not support Unix domain sockets, i.e. hostnames '/local'
or '/unix' are not supported either.

11.14-11.15
===========

   AmigaOS does not support multithreading and therefore does not need
reentrant versions of the resolver functions.
```

1.15 TCPIP.guide/NODE_CHTWELVE

```
Chapter 12. Daemon Processes and inetd Superserver
**************************************************

12.1 Introduction
=================

   With Miami INetD is built in to Miami, and therefore automatically
started when Miami is started.

12.2 syslogd Daemon
===================

   The syslogd Daemon is built in to Miami and always active.  It is
configured in Miami's user interface.

12.3 syslog function
====================

   Miami's syslogd is not as extensively configurable as described in
this section. If you need more control over Miami's logging functions
then please install Petri Nordlund's 'Syslog' package and enable 'Use
syslog.library' in Miami.

12.4 daemon_init function
=========================

   This section is not applicable to AmigaOS.

12.6 daemon_inetd function
==========================

   This section is not applicable to AmigaOS.
```

1.16 TCPIP.guide/NODE_CHTHIRTEEN

```
Chapter 13. Advanced I/O Functions
**********************************

13.2 Socket Timeouts
====================

   Method 1 (calling alarm and using SIGALRM) is not possible with
AmigaOS. Instead you would need to set up a timeout using timer.device,
and wait for the IORequest to return in WaitSelect().

13.5 recvmsg and sendmsg Functions
==================================

   The flags MSG_BCAST and MSG_MCAST are currently not supported by any
AmigaOS protocol stack.

13.9 T/TCP: TCP for Transactions
================================

   Miami is currently the only AmigaOS protocol stack that supports
T/TCP, and only for registered users. This means it is advisable to
perform run-time tests in your software to check if T/TCP is available.
Please see the program 'examples/ttcptest.c' as an example how to do
this.
```

1.17 TCPIP.guide/NODE_CHFIFTEEN

```
Chapter 15
**********

   No additional comments on this chapter.
```

1.18 TCPIP.guide/NODE_CHSIXTEEN

```
Chapter 16
**********

16.2 ioctl function
===================

   Instead of 'ioctl' the function 'IoctlSocket' has to be used.

16.3 Socket Operations
======================

   'sockatmark' is not supported by the standard AmigaOS BSD-socket
API, but by the Miami API.

16.8 ARP Cache Operations
```

```
=========================

   Miami's ARP cache is integrated with routing tables and accessible
through AF_ROUTE sockets, not through the obsolete SIOCxARP operations.
```

1.19 TCPIP.guide/NODE_CHSEVENTEEN

```
Chapter 17. Routing Sockets
***************************

   Routing sockets are only supported by Miami, not by older protocol
stacks.

17.4 sysctl Operations
======================

   The 'sysctl' function is not supported by the standard AmigaOS
BSD-socket API, but the Miami API has the equivalent function
'MiamiSysCtl'. The objects MiamiSysCtl operates on (and the resulting
tree structure) are slightly different from the one described in the
text. Please see the C header files for lists of all supported objects.

17.6 Interface Name and Index Functions
=======================================

   The functions defined in this section are not supported by the
standard AmigaOS BSD-socket API, but by the Miami API.
```

1.20 TCPIP.guide/NODE_CHEIGHTEEN

```
Chapter 18. Broadcasting
************************

18.2 Broadcast Addresses
========================

   Commenting on the indented text: Miami converts 255.255.255.255 to
the subnet-directed broadcast address of the outgoing interface.  In
the case of a multi-homed hosts (which will only be an issue with Miami
Deluxe) broadcasts are sent to the primary interface.

18.5 Race Condition
===================

   The problem described in this section is specific to Unix signal
handling only, and of no concern for AmigaOS programmers.
```

1.21 TCPIP.guide/NODE_CHNINETEEN

```
Chapter 19. Multicasting
************************
```

   Multicasting is only supported by the registered version of Miami, not by older protocol stacks.

1.22 TCPIP.guide/NODE_CHTWENTY

```
Chapter 20. Advanced UDP Sockets
********************************
```

   No additional comments on this chapter.

1.23 TCPIP.guide/NODE_CHTWENTYONE

```
Chapter 21. Out-of-Band Data
****************************

21.2 TCP Out-of-Band Data
=========================
```

   AmigaOS does not have a SIGURG flag. Out-of-band data can be detected by using the exception fd set in select().

```
21.3 sockatmark Function
========================
```

   'sockatmark' is not supported by the standard AmigaOS BSD-socket API, but by the Miami API.

1.24 TCPIP.guide/NODE_CHTWENTYFOUR

```
Chapter 24. IP Options
**********************
```

   No additional comments on this chapter.

1.25 TCPIP.guide/NODE_CHTWENTYFIVE

Chapter 25. Raw Sockets
****

25.7 An ICMP Message Daemon
==========

   This daemon would have to be rewritten to be suitable for AmigaOS,
because it uses Unix domain sockets, which are not available in AmigaOS.

1.26 TCPIP.guide/NODE_CHTWENTYSIX

Chapter 26. Datalink Access
****

26.2 BPF: BSD Packet Filter
==========

   Miami supports BPF (through miamibpf.library, documented
separately), but only for reading, not to write packets to the datalink
layer. Other, older protocol stacks do not support BPF.

26.3 DLPI: Data Link Provider Interface
==========

   DLPI is not supported by any AmigaOS protocol stack.

26.4 Linux: SOCK_PACKET
==========

   SOCK_PACKET is not supported by any AmigaOS protocol stack.

26.5 libpcap: Packet Capture Library
==========

   Miami supports libpcap (through miamipcap.library, documented
separately). Miami's libpcap implementation is built on top of BPF.
Other, older protocol stacks do not support libpcap.

26.6 Examining the UDP Checksum Field
==========

   The very dubious hack described in this section does not work with
Miami, because Miami's BPF implementation does not support the sending
of packets. Applications would need to use either UDP sockets or raw
sockets, or directly access the underlying SANA-II device (for
non-serial connections).

1.27 TCPIP.guide/NODE_CHTWENTYSEVEN

Chapter 27. Client-Server Design Alternatives

27.2 TCP Client Alternatives

The Fork()-based client implementation described in this section and in section 27.3 cannot be used in AmigaOS unless very extensive modifications are made.

27.5-27.12

All servers described in these sections use fork() and cannot be used in AmigaOS unless very extensive modifications are made.