

Lantronix SDK User Guide

For the Lantronix Software Developers Kit

The information in this guide may change without notice. The manufacturer assumes no responsibility for any errors which may appear in this guide.

UNIX is a registered trademark of The Open Group. Ethernet is a trademark of XEROX Corporation. Portions of Portable Universal C are derived from EiC, an online bytecode C interpreter (www.anarchos.com/eic), and are used with permission.

Copyright 1999, Lantronix. All rights reserved. No part of the contents of this book may be transmitted or reproduced in any form or by any means without the written permission of Lantronix. Printed in the United States of America.

The revision date for this manual is **November 3, 1999**.

Part Number: 900-171
Rev. A

WARNING

This equipment has been tested and found to comply with the limits for a Class A digital device pursuant to Part 15 of FCC Rules. These limits are designed to provide reasonable protection against such interference when operating in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy, and if not installed and used in accordance with this guide, may cause harmful interference to radio communications.

Operation of this equipment in a residential area is likely to cause interference in which case the user, at his or her own expense, will be required to take whatever measures may be required to correct the interference.

Changes or modifications to this device not explicitly approved by Lantronix will void the user's authority to operate this device.

Cet appareil doit se soumettre avec la section 15 des statuts et règlements de FCC. Le fonctionnement est soumis aux conditions suivantes:

- (1) Cet appareil ne doit pas causer une interférence malfaisante.
- (2) Cet appareil doit accepter n'importe quelle interférence reçue qui peut causer une opération indésirable.

Contents

1: Introduction.....	1
1.1 What is the Lantronix SDK?	1
1.1.1 Who Would Use the Lantronix SDK?	1
1.1.2 What Can the Lantronix SDK Do?	1
1.2 What is PUC?.....	2
1.2.1 Differences from C	2
1.2.2 Limitations	2
1.2.3 Non-Standard Behaviors.....	3
1.2.4 Known Bugs	3
1.2.5 Important Limits	3
1.2.6 Operating Requirements	4
2: Installing the PUC Files.....	5
2.1 What's Included in the Distribution.....	5
2.2 MSS Optimization	5
2.3 UNIX Installation.....	6
2.4 Windows Installation	6
3: Getting Started.....	7
3.1 How to Run PUC	7
3.2 Example 1: Simple Hello World.....	8
3.3 Example 2: More Interactive Mode	9
3.4 Example 3: Serial Output.....	11
3.5 Example 4: Network Socket Connection	12
3.6 Example 5: Network/Serial Combination	13
3.7 Example 6: Autorun Mode.....	14
3.7.1 Autorun Overview	14
3.7.2 Autorun Example.....	15
4: Debugging.....	17
4.1 Errors.....	17
4.1.1 Common C Language Errors	17
4.1.2 Common Linking Errors	17
4.2 Interactive Mode Help	18
4.2.1 Return Values	18
4.2.2 Show Command.....	18
4.2.3 Trace Command.....	18
4.3 Alternate Compiling Environment.....	19
5: Sample Code	21
5.1 Miscellaneous Samples	21
5.2 PUC Network Samples	22
5.3 Stevens' Network Samples	23

6: API Reference	25
6.1 Header files	25
6.2 Standard Library Functions.....	26
6.3 String Functions	27
6.4 Math Functions	28
6.5 Character Type Functions	28
6.6 I/O Interfaces (File and Serial)	29
6.7 Network Socket Functions	33
6.8 Directory Read Functions	34
6.9 NVR/Flash	34
6.10 Time Functions	34
6.11 Debugging Functions	35
6.12 SNMP Functions	36
A: Technical Support.....	37
A.1 The SDK Support Plan.....	37
A.2 Problem Report Procedure	37
A.3 Other Contact Information	38
B: Disk Management.....	39
B.1 Overview	39
B.2 The Filesystem	39
B.3 Using Disks in PUC	40
B.4 Disk Commands	40
C: References.....	43
C.1 Stevens' Network Examples	43
C.2 EiC Documentation	43
C.3 Other Useful Sources	43
Function List	45
Index.....	47

1: Introduction

1.1 What is the Lantronix SDK?

The Lantronix Software Developers Kit (SDK) allows you to customize the behavior of your MSS in more ways than are available via the standard command set. You can write programs for the MSS that handle serial and network data, and store the finished programs on the MSS Flash ROM so they always run when the MSS boots. For example, you could write an application that monitors industrial or medical equipment and sends email when certain conditions are met.

Example

A hospital blood sugar meter has a serial port, and it is able to send sugar level updates out of its serial port at regular intervals.

You attach the meter to an **MSS**. You then write an application that checks for low sugar levels, and creates and sends an email to alert hospital staff of the low sugar condition.

You can also write an application that keeps track of sugar level data over time, runs statistics on it, or saves it to a file. A hospital worker or lab technician could transfer the data from the MSS to his or her own PC via FTP for further analysis.

The most important part of the Lantronix SDK is a set of software in the MSS called PUC (Portable Universal C) that provides a C language environment for programming. The user writes standard C code, with some limitations. The MSS compiles the C code into a bytecode and then executes the bytecode.

The Lantronix SDK also includes documentation, sample code, and an OEM customization guide to assist developers in writing their own applications for the MSS.

1.1.1 Who Would Use the Lantronix SDK?

The SDK is for anyone who needs customized software that is not available on the MSS, whether it be a network wrapper for a serial device, a custom protocol, or a non-standard means of access or control such as generating email, transferring a file via FTP, or creating an SNMP MIB.

Using the SDK comfortably requires some experience with programming in C. SDK users should have experience using file I/O operations under C, because that is how the serial and network data is handled. If you have used sockets libraries under Unix or Windows, that knowledge will help.

1.1.2 What Can the Lantronix SDK Do?

The Lantronix SDK is intended to take over the MSS' basic function: moving data from serial to network and vice-versa. The user's C code can read from or write to the serial port, accept network connections, make decisions based on serial or network data, and more. Applications have access to most of the built-in functionality of the MSS.

Note: *Some built-in functions on the MSS, such as hostlist support and modem mode, are not available from the SDK.*

Amazing things can be accomplished with application level programming. You do not need to learn, implement, modify or debug any of the OS-Level code. Instead, you can rely upon the proven technology of the MSS to provide a TCP/IP protocol stack, serial flow-control, NVR and filesystems, network time, timer, FTP server, web server, and SNMP functionality, all in a familiar UNIX-like way. You can use these core building blocks to implement a wide variety of functionality.

The SDK does have its limits. First, it has no hard real-time abilities. This means that you cannot write an SDK application to control high-speed or high-precision serial devices that need microsecond-level decision-making capability. Second, there is no direct access to any of the MSS hardware. You could not, for example, write a custom PC card driver for the MSS-VIA. Third, there is no access to the core of the MSS operating system. You would not be able to implement your own non-IP protocol stacks.

1.2 What is PUC?

Portable Universal C (PUC) is the interactive online bytecode interpreter that executes the C code you write for the MSS. PUC is an implementation of Extensible Interactive C (EiC), a command-line interpreted C. PUC shares some of EiC's limitations, but also adds APIs that were not included in EiC such as a Berkeley sockets library and SNMP functions.

Note: *For more information about EiC, see Appendix C.*

1.2.1 Differences from C

PUC generally functions the same way as regular C: functions, file I/O, header files, pointers, and sockets are all available. Chapter 6, *API Reference*, lists all of the functions that are currently available.

The products that currently support the SDK (the MSS models) are all big-endian hardware. However, future implementations of the SDK will be available on little-endian hardware. SDK developers should use the supplied functions (htonl, ntohl, etc.) to ensure that communications with external devices will work, regardless of the byte-ordering of the hardware.

1.2.2 Limitations

There are a few significant limitations to PUC.

- ◆ PUC does not support float and double data types, forking, child processes, labels, signals, or gotos due to size limitations.
- ◆ Some older prototype definition styles are not understood. Use ANSI standards.
- ◆ There are no makefiles, but you can `#include` source files from other source files so you can make them automatically load in the right order.
- ◆ Some of the standard library functions are not implemented. See Chapter 6 for a list of implemented functions and their prototypes.

1.2.3 Non-Standard Behaviors

There are some important non-standard behaviors to be aware of:

- ◆ **read()** and **recv()** return -1 on remote disconnect, whereas most flavors of unix expect a 0.
- ◆ Terminal settings require both a newline (\n) and a carriage return (\r) to move to the beginning of the next line. **fgets()** terminates at either character.
- ◆ In general, files are opened in binary mode rather than text mode.
- ◆ **setbuf()** can only be changed to NULL from its default setting.
- ◆ **sscanf()** doesn't terminate when the input fails to match the control specification; instead it assigns 0 values to the unmatched variables.
- ◆ When **accept()** copies socket information, it does not copy blocking status. Blocking status must be set explicitly after the accept. **fcntl()** is only used to change blocking status and has no other behaviors.
- ◆ UDP sockets never block on **recvfrom()** or **sendto()**.
- ◆ Files can only be fopened as read-only, write-only, or append-only. To use a file in read/write mode, it must be opened twice using two different file pointers:

```
FILE *fpr=fopen("dev", "r");
FILE *fpw=fopen("dev", "w");
```

Files opened with **fopen** should never be set to nonblocking modes; **open** should be used if nonblocking modes are required.

1.2.4 Known Bugs

When initializing a static structure, function pointers and **extern** forward references may not be set to the correct value. Try to avoid using the addresses of functions or unallocated variables when statically initializing global structures..

1.2.5 Important Limits

RAM

Depends on MSS model, but at least 512 KB.

Maximum RAM disk file size

Depends on MSS model, but at least 64 KB.

Maximum simultaneous file opens

32

Maximum filename size

40 characters

1.2.6 Operating Requirements

You need four things to run PUC:

- 1 A Lantronix MSS100 or MSS-VIA thin server running code v3.6/2 (991020) or later, or an MSSLite with hardware revision X.X or later.

To check you current software version, enter the **Show Version** command.

- 2 A text editor, used to write the code.
- 3 Telnet functionality on your development host, used to access the MSS.

Note: *You cannot use the serial terminal to control the MSS in PUC mode.*

- 4 A TFTP server or FTP client, used to download code into the MSS.

You can develop applications on any host that can access the MSS and its TFTP loadhost. The next chapter will step you through a sample development cycle.

2: Installing the PUC Files

2.1 What's Included in the Distribution

The PUC files are distributed partially on a CD-ROM and partially on the MSS internal ROM disk.

The PUC distribution CD-ROM contains a subdirectory called `/tftpboot/puc`. You should place the contents of this directory on your TFTP loadhost in a directory called `/tftpboot/puc`. You do not need to copy the `/include` subdirectory, but it may be helpful to refer to the files during development.

The MSS operating system contains a ROM filesystem in which it stores all of the necessary include files. This lets PUC access the include files without referring to the TFTP loadhost.

<code>/rom/puc</code>	<code>/rom/puc/include</code>			<code>/rom/puc/include/netinet</code>	<code>/rom/puc/include/sys</code>
<code>unp.h</code>	<code>assert.h</code>	<code>math.h</code>	<code>string.h</code>	<code>in.h</code>	<code>socket.h</code>
	<code>ctype.h</code>	<code>netdb.h</code>	<code>termios.h</code>		<code>stat.h</code>
	<code>dirent.h</code>	<code>stdarg.h</code>	<code>time.h</code>		<code>stdtypes.h</code>
	<code>errno.h</code>	<code>startpuc.h</code>	<code>unistd.h</code>		<code>types.h</code>
	<code>fcntl.h</code>	<code>stdio.h</code>	<code>unp.h</code>		
	<code>limits.h</code>	<code>stdlib.h</code>			

Note: *The values of constants in the header file are subject to change in future releases of the SDK.*

`unp.h` is the header file used by all of Stevens' code, with definitions for some common TCP sockets. Adding this file to the beginning of your program allows you to code and test without figuring out which specific include files you need or in which order you need to specify them. Include this file any time you want to use sockets.

2.2 MSS Optimization

There is only one required configuration item for SDK development on the MSS: you must configure your loadhost with the Change Loadhost command. The following example uses a TFTP loadhost. The rest of the configuration in this section is optional, or only required for certain situations.

```
Local>> CHANGE LOADHOST 192.0.1.123
```

- ◆ To use the functions `gethostname()` and `gethostbyname()`, you must set a nameserver. It is also a good idea to set a secondary nameserver (in case the primary one is unavailable) and a default domain name.

```
Local>> CHANGE NAMESERVER 192.0.1.55
local>> CHANGE SECONDARY NAMESERVER 192.0.1.56
Local>> CHANGE DOMAIN "supercorp.com"
```

- ◆ To use the function `time()`, you must set a timeserver. NTP servers give date and time information. Day-time servers give only time-of-day information. Choose one of the following sets of commands:

```
Local>> CHANGE TIMESERVER NTP 192.0.1.101
Local>> CHANGE TIMESERVER GMTOFFSET -1
Local>> CHANGE TIMESERVER DAYTIME 192.0.1.102
```

- ◆ To speed reboots of the MSS, you may wish to *temporarily* disable BOOTP and RARP queries. However, keep in mind that EZWebCon cannot set the IP address of a server that has BOOTP disabled.

```
Local>> CHANGE BOOTP DISABLED
Local>> CHANGE RARP DISABLED
```

- ◆ To prevent hosing up attached serial devices, turn off boot messages.

```
Local>> CHANGE SILENTBOOT ENABLED
```

2.3 UNIX Installation

This section assumes familiarity with the UNIX operating system and MSS setup. If you have trouble following any of these steps, contact your system administrator.

- 1 Create a directory on your TFTP loadhost named **/tftpboot/puc**.
- 2 Make sure the permissions for **/tftpboot/puc** are set to 775 (full permissions for owner and group, read and execute permissions for world).
- 3 Copy the PUC files to the **/tftpboot/puc** directory.
- 4 Make sure the permissions of all of the files in **/tftpboot/puc** are set to 664 (read/write permissions for owner and group, read permissions for world).
- 5 Make sure your MSS is configured to use the proper loadhost.

Note: *Instructions for configuring and checking the TFTP loadhost are located in your MSS Installation Guide and the MSS Reference Manual.*

Filenames on UNIX TFTP loadhosts are typically case-sensitive. Keep this in mind when specifying filenames for files that are located on your TFTP loadhost.

2.4 Windows Installation

This section assumes familiarity with the Windows operating system. If you have trouble following any of these steps, contact your system administrator.

- 1 Create a directory on your TFTP loadhost named **<tftp>:\tftpboot\puc**. You will need to consult your TFTP documentation to find out where the download directory is located.
- 2 Copy the PUC files to the **<tftp>:\tftpboot\puc** directory.
- 3 Make sure your MSS is configured to use the proper loadhost.

Note: *Instructions for configuring and checking the TFTP loadhost are located in your MSS Installation Guide and the MSS Reference Manual.*

3: Getting Started

This chapter contains several development examples that will teach you how to use PUC. The explanations and examples in this chapter assume that you have followed the instructions in Chapter 2 for installing the PUC files on your loadhost.

- ◆ Example 1: Simple Hello World (page 8)
- ◆ Example 2: More Interactive Mode (page 9)
- ◆ Example 3: Serial Output (page 11)
- ◆ Example 4: Network Socket Connection (page 12)
- ◆ Example 5: Network/Serial Combination (page 13)
- ◆ Example 6: Autorun Mode (page 14)

3.1 How to Run PUC

You can use PUC in three different modes: interactive, command line, and autorun. The examples in this chapter detail all three modes:

- ◆ You start interactive mode by typing `cc` at the `MSS Local>` prompt (See Examples 1-2). Then you enter instructions one at a time at the `PUC>` prompt. PUC will execute each instruction as it is entered, and show return values, which is useful for debugging. Use this mode for general development.
- ◆ In command line mode, you execute an existing C-code file by typing a single command line at the `Local>` prompt (see Examples 3-4). PUC will execute the program immediately, and only once. Use this mode to test nearly-finished applications.

Command line arguments can follow the source filename. They will be passed to the program as standard `argc` and `argv` arguments to the `main()` function.

- ◆ In autorun mode, you tell the MSS to run specific C-code files at every startup (see Example 5). The command is similar to the one used for command line mode, except for an `-auto` flag. Use this mode to run finished applications on the MSS.

Autorun mode will start the application 5 seconds after the MSS boots. If the program exits or aborts, the MSS will restart it 5 seconds later. Compile-time delays will add a few seconds to these times, depending on program size.

You must be the privileged user to run PUC, regardless of which mode you choose. In addition, you must be connected to the MSS via a network connection; you cannot run PUC from the serial port.

3.2 Example 1: Simple Hello World

This is a basic interactive mode example that uses an include file and a printf statement. User entries are bolded; if you wish to follow along, enter the bold items into your Telnet window. If you need help, type `:help` at the PUC> prompt.

- 1 Telnet into your MSS, enter a username, and become the privileged user.

```
% telnet mymss100
Trying 192.0.1.125
Connected to mymss100
Escape character is '^]'.

Lantronix MSS100 Version V3.6/2(991020)
Type HELP at the 'Local>' prompt for assistance.

Username> Bob
Local> set priv
Password: system (not echoed)
Local>>
```

Note: *You cannot start PUC from the serial port.*

- 2 Enter PUC's interactive mode by typing `cc`.

```
Local>> cc
PUC: Interactive mode - type :help for help, or :exit to exit.
PUC 1>
```

Note: *When you enter the `cc` command, the prompt changes to “PUC n>” where n represents the number of the current user entry.*

- 3 At the PUC> prompt, enter compiler directives (# commands) or C commands. For this example, enter the following items:

```
PUC 1> #include <startpuc.h>
returned: (void)
PUC 2> printf("Hello world.\n\r");
Hello world.
returned: 14
PUC 3>
```

The `#include` command causes PUC to load the SDK header file `<startpuc.h>`. PUC will look through the `/flash`, `/ram`, `/rom`, and `/tftpboot/puc` directories, in that order, until it finds the file. In this case, `<startpuc.h>` is a standard header file, so it is found with other include files on the MSS `/rom` directory.

After loading the header file, PUC returns the next prompt. When you enter the `printf()` statement, PUC executes it, displays the appropriate output, and displays a return value. In this case, `printf()` returns the value 14, which is the number of characters printed. The return value of every statement is reported in interactive mode.

- 4 Exit PUC by typing `:exit` at the PUC> prompt.

If this does not work or your code hangs, you can usually abort PUC by typing **Ctrl-C**. Aborting a running program using **Ctrl-C** may not work if your code is blocked waiting for input or output (which is one of the reasons Lantronix recommends using non-blocking I/O). If you cannot use **Ctrl-C**, you may have to cycle power on the MSS.

3.3 Example 2: More Interactive Mode

This example expands your understanding of interactive mode by looking at return values and teaching you how to write and execute your own code snippets. User entries are bolded; if you wish to follow along, enter the bold items into your Telnet window.

- 1 Telnet into your MSS, enter a username, and become the privileged user.
- 2 Enter PUC's interactive mode by typing cc at the Local> prompt. You will see the PUC> prompt for the remainder of this example.
- 3 Include the header file <startpuc.h>.

```
PUC 1> #include <startpuc.h>
returned: (void)
PUC 2>
```

Notice that a “returned” line is displayed below the command line before the next prompt. Normally, “returned” displays the return value of the item entered. In this case, (void) means that there is no return value for the #include entry.

- 4 Declare an integer named t.

```
PUC 2> int t;
returned: (void)
PUC 3>
```

- 5 Assign integer t a value of 7.

```
PUC 3> t=7;
returned: 7
PUC 4>
```

PUC will display the value of t before the next prompt. Since you just assigned t's value as 7, PUC returns 7. After you assign a value to an integer, you can check the value by entering the integer name followed by a semicolon.

```
PUC 3> t;
returned: 7
PUC 4>
```

- 6 Enter the following printf() statement, which also shows you the current value of t.

```
PUC 4> printf("t=%d\n\r", t);
t=7
returned: 5
PUC 5>
```

In this case, t is still 7, so the printf statement causes PUC to display “t=7.” The following line is the return value of the printf statement. There are 5 characters printed, including the \n and \r newline characters, so the number 5 is displayed.

- 7 Use the PUC **:show** command to get more information about the `printf` statement.

```
PUC 5> :show printf
printf -> Builtin Func (
    fmt: * constchar,
    ...
) returning int
(void)
PUC 6>
```

PUC displays the prototype definition of the function.

- 8 Take a break from PUC for a moment. Create a file on your loadhost that contains the following code and save it as `example1.c` under `/tftpboot/puc`. Make sure the file has 664 permissions.

```
#include <startpuc.h>

void main()
{
    int t;
    t = 15;
    printf("t = %d\n\r", t);
}
```

You can use any text editor to create the file. If you use a word processor, be sure to save the file as plain text, otherwise the formatting commands and other spurious characters will confuse PUC.

- 9 Now go back to your PUC session and read in `example1.c`.

```
PUC 6> #include example1.c
returned: (void)
PUC 7>
```

PUC will load the file from your TFTP loadhost and interpret the file, but will not execute `main()` yet.

- 10 Execute the main function.

```
PUC 7> main();
t = 15
returned: (void)
PUC 8>
```

The return value of `t` is 15, because that is the value you defined in the `main()` function of `example1.c`

- 11 Try checking the value of `t` again.

```
PUC 8> t;
returned: 7
PUC 9>
```

Notice that `t` is not 15 as you might expect. It is 7, because the scope of `int t` in `main()` is internal to `main()` — it has nothing to do with your earlier globally declared `int t`. Once `main()` finishes, its local variables disappear. From the command line, you can only see global variables.

- 12 Exit PUC by typing **:exit** at the `PUC>` prompt.

```
PUC 8> exit
Local>>
```

3.4 Example 3: Serial Output

This example opens the serial port, sends characters to it, and then closes it. User entries in examples are bolded; if you wish to follow along, enter the bold items into your Telnet window.

- 1 Place the sample `spin.c` file on your loadhost in the `/tftpboot/puc` directory. The contents of the file are included here for reference.

```
#include <startpuc.h>

void main()
{
    int  fd;
    char c = ' ';
    int  i;

    /* open serial port 0 */
    fd = open("tt0:", O_WRONLY);

    /* spin through printable chars from <spc> to <~> 50 times */
    for(i=0; i < 50; i++ ) {
        while( c <= '~' ) {
            write(fd, &c, 1);
            c++;
        }
        c=' ';
    }

    /* always close any files you open */
    close(fd);
}
```

- 2 Connect a display device, such as a serial terminal, to the serial port of your MSS. The display device should match the MSS serial port settings: 9600 baud, 8 bits, no parity, and one stop bit.

If you boot your MSS and hit return a few times on the serial terminal, you should see a response from the MSS on your terminal.

- 3 Log into your MSS and become the privileged user.
- 4 Run the `spin.c` file in PUC's command line mode.

```
Local>> cc spin.c
```

You should see characters on the terminal.

3.5 Example 4: Network Socket Connection

This example show how to connect to a remote host using network sockets. User entries are bolded; if you wish to follow along, enter the bold items into your Telnet window.

The `tcp_connect` function is contained in the file `tcp_connect.c`; this function handles the actual socket connection.

- 1 Place the sample **`timecli.c`** and **`tcp_connect.c`** files on your loadhost in the `/tftpboot/puc` directory. The contents of **`timecli.c`** are included here for reference:

```
#include <unp.h>
/* automatically include needed c files in PUC.  Note that these
   files must be in the search path. */
#ifdef NO_PUC
#include "tcp_connect.c"
#endif

void
main(int argc, char **argv)
{
    int    sockfd, n;
    long   seconds;
    char   line[MAXLINE];

    if (argc != 2) {
        printf("usage: a.out <IPaddress>");
        exit(1);
    }

    /* Time server client */
    if ((sockfd = tcp_connect(argv[1], SOCK_TIMESERVER)) > -1) {
        while ((n = recv(sockfd, (char *) &seconds, MAXLINE, 0)) > 0) {
            printf("seconds since 1900: %u\n\r", seconds);
            close(sockfd);
        }
    }

    /* Daytime client */
    if ((sockfd = tcp_connect(argv[1], SOCK_DAYTIME)) > -1) {
        while ((n = recv(sockfd, line, MAXLINE, 0)) > 0) {
            line[n] = 0; /* null terminate */
            printf("The time is %s\n\r", line);
        }
        close(sockfd);
    }
}
```

- 2 Log into your MSS and become the privileged user.
- 3 Run the `timecli.c` file in PUC's command line mode. You must include the name of the host you wish to connect to as an argument in your command line. In this case, the desired host is *delphi*.

```
Local_2>> cc timecli.c delphi
PUC: Compiling <timecli.c>...
PUC: looking for <puc/timecli.c> on TFTP host...
seconds since 1900: 3150123680
The time is Thu Oct 28 11:21:20 1999

PUC: exit(0)
```


3.6 Example 5: Network/Serial Combination

This example accepts a TCP client, and bidirectionally echoes data back and forth between the client and the MSS serial port. Anything presented on the serial port gets sent to the client; anything sent from the client gets pushed out the serial port.

The code is divided up into three files, all of which must be loaded in order to run.

- ◆ network/tcpserv.c

This file sets up the MSS as a TCP server listening for connections on port 9877. You would connect to this server from UNIX with a command like **telnet <mss name> 9877** or **nc -v <mss name> 9877**.

- ◆ wrapper.c

tcpserv.c automatically loads wrapper.c, which includes a series of error-trapping wrapper functions for many common commands. All of the wrapper functions are named for the command they wrap with the first letter capitalized. For example, Close() wraps the built-in command close().

- ◆ network/do_buffer.c

tcpserv.c also requires the inclusion of function do_socket. It calls this function whenever a new client connects to the server. In this case, you would load the file network/do_buffer.c, which opens the serial port in nonblocking mode, sets the network socket to nonblocking mode, and then watches them both for incoming data. Incoming network data is sent out the serial port immediately, while incoming serial data is buffered until a specified stop character is read or a certain amount of time has passed with no data received.

- 1 Put the 3 files listed above into /tftpboot/puc
- 2 Load the files into PUC. Note that you do not have to load wrapper.c, it is loaded automatically by tcpserv.c.

```
Local_2>> cc
PUC: Interactive mode - type :help for help, or :exit to exit.

PUC 1> #include tcpserv.c
PUC: Looking for <puc/tcpserv.c> on TFTP host...
returned: (void)

PUC 2> #include do_buffer.c
returned: (void)

PUC 3>
```

- 3 Start your new server running on the MSS.

```
PUC 3> main();
waiting for connection
```

- 4 Connect to the new server from a UNIX host.

```
% telnet mymss100 9877
```

Back on the PUC session, you will see the message “Accepted socket.”

- 5 On the UNIX terminal, type some data and hit Return. You should see the data on the serial terminal. You should also see some status messages on your PUC session.
- 6 On the serial terminal, type some data and hit Return. You should see the data on the UNIX terminal. You should also see some status messages on your PUC session.
- 7 Quit the Telnet session. Your MSS should display the following:

```
Transferred xx bytes
closing socket
waiting for connection
```

At this point, the server is waiting for new connections.

3.7 Example 6: Autorun Mode

3.7.1 Autorun Overview

Once you have verified that your application works, you can tell the MSS to run the code at each startup. Autorun mode is similar to running a file in command line mode, but you must add the `-auto` flag.

- 1 Specify the file you wish to run. Place the `-auto` flag between `cc` and the name of your `.c` file.

```
Local>> cc -auto myapp.c
PUC: Autorun is Enabled.
Local>>
```

The code must be stored either on the MSS /flash disk or on the loadhost in /tftpboot/puc. If you place code on the MSS /ram disk, it will be cleared before the MSS reboots.

Note: Put a `-DDEFINEME` between `cc` and the filename to define the `DEFINEME` macro. You can also specify arguments after the filename, if your function is defined to accept arguments. Arguments are passed using the standard C `argc` and `argv` variables.

Once you enable autorun mode, `stdin`, `stdout`, and `stderr` will be directed to a null device. Further `printf` functions won't display anything, and attempts to read from `stdin` will return EOF (end of file).

- 2 Reboot the server.

```
Local>> init delay 0
MSS Rebooting in 0 seconds...
```

To disable autorun mode, enter the following command.

```
Local>> cc -noauto
PUC: Autorun is Disabled.
Local>>
```

3.7.2 Autorun Example

The C file `example2.c` shows a simple network/serial interaction: the MSS queries a remote network daytime server, and sends the result to the serial port. User entries are bolded; if you wish to follow along, enter the bold items into your Telnet window.

- 1 Connect a display device, such as a serial terminal, to the serial port of your MSS. The display device should match the MSS serial port settings: 9600 baud, 8 bits, no parity, and one stop bit.
- 2 Connect to the MSS via FTP using *root* as your username and *system* as your password.

```
/tftpboot/puc> ftp mymss100
Connected to mymss100.
220 FTP Version 0.1, Punix version V3.6/2(991020) on ip0:
Name (mymss100:myname): root
331 Password required.
Password: system
230 logged in with privs, proceed.
Remote system type is MSS100.
ftp>
```

- 3 Load the file `example2.c` into the MSS Flash disk.

```
ftp> cd /flash
250 directory is now /flash
ftp> put example2.c
local: example2.c remote: example2.c
250 PORT set to <ip address>,(20,7522)
150 Opened port 7522.
226 transfer complete, 1344 bytes transferred
1344 bytes sent in 0.00 seconds (685.38 Kbytes/s)
ftp>
```

- 4 Exit FTP mode.

```
ftp> close
221 Goodbye

/tftpboot/puc>
```

- 5 Log into your MSS and become the privileged user.
- 6 Set the file `example2.c` for autorun mode. You must include the name of the desired daytime server after the code file name. In the example belowe, the daytime server is `delphi`.

```
Local>> cc -auto example2.c delphi
PUC: Autorun is Enabled.
Local>>
```

- 7 Reboot the MSS.

```
Local>> init delay 0
Initializing server...
```

- 8** When your MSS reboots, you will see the following on the serial port terminal:

```
%% Lantronix MSS100
%% Ethernet Address: 00-80-a3-xx-xx-xxInternet Address <ip address>

Thu Oct 28 12:29:38 1999
Thu Oct 28 12:29:48 1999
Thu Oct 28 12:29:59 1999
Thu Oct 28 12:30:09 1999
...
```

Note the repetition of the time display. The `-auto` switch will re-execute the program if it exits. You could use an infinite loop like `while (1) sleep(10);` inside the program to only print the time once.

- 9** Disable autoboot mode.

```
Local>> cc -noauto
PUC: Autorun is Disabled.
Local>>
```

4: Debugging

This chapter includes tips on debugging your code and interpreting errors received from the MSS or PUC. Please read this section before contacting Lantronix Technical Support. Support contact information is noted in *Appendix A*.

Note: *To view a list of available MSS commands, type `Help` at the **Local**> prompt. For help when PUC is running in interactive or command line mode, type `:help` at the **PUC n**> prompt.*

4.1 Errors

4.1.1 Common C Language Errors

Flow reaches end of non-void function

You will get an error if there is no return statement in `main()` when it has been declared to return a value.

```
Local_2>> cc test.c
PUC: Compiling <test.c>...
PUC: looking for <puc/test.c> on TFTP host...
Warning: in test.c near line 13: Flow reaches end of non-void function `main'
```

Incorrect function usage

You will get an incorrect function usage error if you forget to `#include` the proper header file (such as `<startpuc.h>`) before calling a function (such as `printf()`).

```
PUC 2> printf("hello");
Error in ::PUC:: near line 2: Incorrect function usage: printf
```

4.1.2 Common Linking Errors

Couldn't find

You will get a couldn't find error if the file (such as `test.c`) was not found in the search path.

```
Local_2>> cc test.c
PUC: Compiling <test.c>...
PUC: looking for <puc/test.c> on TFTP host...
%Error: Couldn't find <test.c>.
```

Access violation

You will get an access violation error if the file you specify (such as `test.c`) does not have world-read permissions.

```
Local_2>> cc test.c
PUC: Compiling <test.c>...
PUC: looking for <puc/test.c> on TFTP host...
%Error: TFTP error code 5 message <Access violation>.
%Error: Couldn't find <test.c>.
```

4.2 Interactive Mode Help

There are three PUC features that can help you debug your code in interactive mode: return values, the **:show** command, and the **:trace** command. In addition, you can type **:help** for general PUC help.

4.2.1 Return Values

Each line will return its runtime value when entered at the command prompt in interactive mode. Use this feature to check the values of your variables and whether your functions were able to execute.

```
PUC 2> t = 7;  
returned: 7
```

4.2.2 Show Command

The PUC **:show** command displays the expected function prototypes of any function, whether built-in or user defined.

```
PUC 3> :show printf  
printf -> Builtin Func (  
          fmt: * const char ,  
          ...  
          ) returning int  
returned: (void)
```

4.2.3 Trace Command

The **:trace** command shows when and where a new function is called. If your code is getting stuck somewhere, you can probably find out where by using this command.

```
Local_2>> cc  
PUC: Interactive mode - type :help for help, or :exit to exit.  
  
PUC 1> #include example3.c  
PUC: looking for <puc/example3.c> on TFTP host...  
returned: (void)  
  
PUC 2> :trace  
2,  
returned: (void)  
  
PUC 3> main();  
3,  
[main] 11,  
[foo] 5,hello  
6,  
[main] 12,  
[::PUC::] 3,  
returned: (void)  
  
PUC 4>
```

4.3 Alternate Compiling Environment

Because PUC C is almost identical to ANSI C, you can set up an alternate compiling environment and compile SDK code on your PC or UNIX host. That way, you may see different compile-time error messages for problems in your code, and it may be faster to try out different code snippets.

These instructions assume you are running UNIX and have the gcc compiler available.

- 1 Set up a directory under /tftpboot/puc called localinc.
- 2 Put alternate versions of the header files <startpuc.h> and “unp.h” in /localinc. You may have to modify these files slightly to reflect different header files in your environment.
- 3 Add the line `#define NO_PUC` to your <startpuc.h> file, and use this definition within your source files if you need to make any environment-related changes.
- 4 Add the definitions for PUC's special features to <startpuc.h>, as desired. See the NON-ANSI sections of the *API Reference* chapter for more information.
- 5 When you are ready to compile in your custom environment, add the new path to use to look for include files:

```
unix%> gcc -I /tftpboot/puc/localinc test.c
```

When you compile, you'll get a file a.out that should do the same thing your MSS would do if you were to compile on it, but a.out will run on your PC or UNIX host instead of on the MSS.

You will probably have to do something special to read from the serial port in your custom environment, since “tt0:” will fail. Add a section to your code that is similar to the following to open the host's serial port.

```
#ifdef NO_PUC
    int fd = open("/dev/ttyq1", O_RDWR);
#else
    int fd = open("tt0:", O_RDWR);
#endif
```

5: Sample Code

The samples described in this chapter and contained on the Lantronix SDK distribution disk allow you to do many useful things with the MSS SDK. For example, you can create an SMTP mail client, a simple FTP client, an SNMP agent, a UDP server, and more. Lantronix recommends basing your custom applications on these samples.

Note: *Some of the examples won't work correctly unless a Nameserver and Domain have been defined on your MSS. See your Installation Guide for information about configuring these items.*

The files are grouped into subdirectories on the distribution CD-ROM. However, they must be placed at the top level of /tftpboot/puc in order to work properly. This chapter may not be all-inclusive. For a list of the samples that are included in your distribution, please refer to the readme on the distribution disk.

5.1 Miscellaneous Samples

spin.c	Spins characters out of the serial port.
echo.c	Shows all of the possible ways to read and write to the serial port, using file descriptors and file pointers and all of the I/O commands.
log.c	Saves incoming serial data to a file on the ramdisk. Enter data on the serial port, ending with Z . Check the results from the MSS command line: <pre>Local>> DISK MORE /ram/outf</pre>
timecli.c	Simple TCP client examples using timeservers.
param.c	Provides param_read, param_write, and param_commit to store parameters in a local Flash disk file. Parameters can be arbitrary length.
fractoa.c	Convert a fraction into a decimal string. Useful since the PUC does not use the <i>float</i> data type (this is intended to reduce the size requirements of the software).
utils.c	dprintf is a printf specialized for debugging. upTime reports uptime in seconds since it was first called. fdprintf acts like fprintf but takes a filedescriptor instead of a file pointer.
getip.c	Gets the MSS's IP address via gethostbyname .
ioctl.c	Shows usage of platform-specific IOCTL calls.
dirwatch.c	Examines modification times for all files in all disk subdirectories.

5.2 PUC Network Samples

tcpserv.c	Genericized TCP server running under PUC. The function do_socket() services the connection. It is based on Stevens' figure 5.2.
tcpcli.c	Genericized TCP client running under PUC. The function do_socket() services the connection.
tcp_connect.c	Opens a TCP connection to a remote network server. This file is called by many of the examples.
do_buffer.c	tcpserv.c or tcpcli.c along with do_buffer.c collects incoming serial data until one of three conditions is met: buffer full, "send flag" encountered, or no data received for a specified time delay. This prevents the MSS from sending single-character packets for slow serial data, which reduces network traffic.
do_filebuf.c	Buffers a network file into a local ramdisk file, then sends it out the serial port in one piece when the network connection closes.
udpserv.c	Queryable UDP server. Responds to simple commands from multiple simultaneous clients.
mailcli.c	An SMTP mail client that connects to an SMTP mailserver and sends email.
ftpcli.c	An FTP file transfer client. This limited implementation can only use PASV mode for file get, put, and directory listings.
snmp.c	Fairly complete and complex SNMP agent. Requires mib.c , asn1.c , agent.c , and mymib.c (which should be customized for your specific purposes). Supports traps but doesn't handle tables.
directed.c	Accepts multiple TCP clients to the same port, and sends all incoming client messages out the serial line one at a time. Each incoming message needs a "return address" identification header; serial incoming messages are scanned for matching identification headers and sent back to the original sender only. Input is buffered to send only full messages.
rs491.c	Implements local MSS control of an RS491 serial protocol device so that raw files can simply be sent from a remote network client. Allows you to use the SDK to buffer network data for a timing-critical serial protocol.
pipec.c	Acts as a kind of wrapper for generalized serial and network I/O. Users can intercept the data streams to customize behavior using mypipe.c without having to open any sockets or serial lines. The user functions serIn and netIn are repeatedly called with any pending data. Outgoing data can be sent to serOut or netOut .

5.3 Stevens' Network Samples

The examples in this section are taken from *Unix Network Programming, Volume 1, 2nd Ed.* by W. Richard Stevens. Full bibliographic information can be found in *Appendix C*.

Some of the examples are modified from the original Stevens examples in order to comply fully with PUC. The higher-level functions were modified as little as possible; the wrappers were modified more significantly. Differences are noted.

wrapper.c	Error-trapping wrappers for socket I/O functions. The wrappers are mainly useful for debugging since they exit program execution on failures.
daytimetcpcli.c	<p>A daytime client that queries a remote daytime server using <code>inet_pton</code>. Returns a formatted time string. See Stevens' Figure 4.5.</p> <p>The client establishes a TCP connection with a server and the server sends back the current time and date in a human-readable format.</p>
inet_pton.c	Converts dotted quad (presentation format) IP addresses to network format. Required by <code>daytimetcpcli.c</code> .
tcpserv01.c	TPC echo server using the unassigned port 9877. Modified to run under PUC by removing the fork. See Stevens' Figure 5.2.
udpserv01.c	UDP echo server that handles multiple clients simultaneously, echoing back any incoming data to each specific client. See Stevens' Figure 8.3.

6: API Reference

This chapter lists the functions that are built into the Lantronix SDK environment. You will be able to see each function's calling convention, purpose, and return values. Functions are arranged by the include file in which they are found, and further divided into those that adhere to the ANSI standard calling convention (ANSI), and those that differ from the standard C API/runtime library (Non-ANSI). Some Non-ANSI functions are simply implementation differences, others are custom API functions created by Lantronix.

The functions are grouped into major sections as follows:

- ◆ Standard Library Functions, page 26
- ◆ String Functions, page 27
- ◆ Math Functions, page 28
- ◆ Character Type Functions, page 28
- ◆ I/O Interfaces (File and Serial), page 29
- ◆ Network Socket Functions, page 33
- ◆ Directory Read Functions, page 34
- ◆ NVR/Flash, page 34
- ◆ Time Functions, page 34
- ◆ Debugging Functions, page 35
- ◆ SNMP Functions, page 36

Note: *An alphabetical index of functions can be found on page 45.*

6.1 Header files

The required header files are stored in the MSS ROM disk (/rom):

/rom/puc	/rom/puc/include			/rom/puc/include/netinet	/rom/puc/include/sys
unp.h	assert.h	math.h	string.h	in.h	socket.h
	ctype.h	netdb.h	termios.h		stat.h
	dirent.h	stdarg.h	time.h		stdtypes.h
	errno.h	startpuc.h	unistd.h		types.h
	fcntl.h	stdio.h	unp.h		
	limits.h	stdlib.h			

Note: *Do not rely on the values of any constants in the header files. They may change in future releases of the SDK.*

The header files “**unp.h**” and **<startpuc.h>** include other header files. Therefore, many of the functions described in this chapter can be gained from including either “**unp.h**” or **<startpuc.h>** in your program. Basically **<startpuc.h>** includes all non-socket files and “**unp.h**” includes **<startpuc.h>** and the socket-related files.

“unp.h” includes	<startpuc.h> includes		
startpuc.h ----->	stdio.h	string.h	time.h
netdb.h	stdlib.h	limits.h	unistd.h
/netinet/in.h	ctype.h	assert.h	termios.h
/sys/socket.h	dirent.h	errno.h	
	fcntl.h	sys/stat.h	

Note: *The error “Incorrect Function Usage” usually means that the function hasn't been prototyped, which means that you haven't included the necessary header files. For sockets and general usage, you should only have to #include “unp.h”; it will include everything you need.*

6.2 Standard Library Functions

<stdlib.h> Standard Library Functions - ANSI		
abort	void abort(void);	Abort program without running atexit functions.
abs	int abs(int);	Return absolute value of int.
atexit	int atexit(void f(void));	Set a function to run at exit time.
atoi	int atoi(const char *s);	String to integer.
atol	long atol(const char *s);	String to long.
bsearch	void *bsearch(const void * _key, const void * _base, size_t _nelem, size_t _size, int _cmp(const void * keyval, const void * datum));	Binary search a sorted list.
calloc	void *ptr = calloc(size_t nelem, size_t elsize);	Allocate memory, init to zero.
exit	void exit(int eval);	Exit your program, returns eval.
free	void free(void *ptr);	Free memory block.
labs	long labs(long);	Return absolute value of long.
malloc	void *ptr = malloc(size_t size);	Allocate uninitialized memory.
qsort	void qsort(void *_base, size_t _n, size_t _size, int (*_cmp)(const void *, const void *));	Quicksort function.
realloc	void *ptr = realloc(void *ptr, size_t size);	Resize memory block.
rand	int rand(void);	Generate a random number from 1 to RAND_MAX.
srand	void srand(unsigned int seed);	Seed the random number generator.
strtol	long strtol(const char *str, char **endptr, int base);	String to long int.
strtoul	unsigned long strtoul(const char *str, char **endptr, int base);	Str to ulong.

<stdlib.h> Standard Library Functions - Non-ANSI		
itoa	<code>char *itoa(int n, char *str, int radix);</code>	Integer to string.
ltoa	<code>char *ltoa(long n, char *str, int radix);</code>	Long int to string.
ultoa	<code>char *ultoa(ulong n, char *str, int radix);</code>	Unsigned long to string.
utoa	<code>char *utoa(uint n, char *str, int radix);</code>	Unsigned int to string.

6.3 String Functions

<string.h> String Functions - ANSI		
memchr	<code>void *memchr(const void *s, int c, size_t n);</code>	Find a character in memory.
memcmp	<code>int memcmp(const void * s1, const void * s2, size_t n);</code>	Compare memory contents.
memcpy	<code>void *memcpy(void * dst, const void * src, size_t n);</code>	Copy data.
memmove	<code>void *memmove(void * dst, const void * src, size_t n);</code>	Copy overlapping data.
memset	<code>void *memset(void *s, int c, size_t n);</code>	Set memory to value.
strcat	<code>char *strcat(char * s1, const char * s2);</code>	Concatenate string s2 to end of string s1.
strchr	<code>char *strchr(const char *s, int c);</code>	Find character in string.
strcmp	<code>int strcmp(const char * s1, const char * s2);</code>	Compare strings.
strcpy	<code>char *strcpy(char * dst, const char * src);</code>	Copy a string.
strcspn	<code>size_t strcspn(const char *s, const char *reject);</code>	Return length until first character in string reject.
strerror	<code>char *strerror(int n);</code>	Generate error message string from error number n.
strlen	<code>size_t strlen(const char *s);</code>	Return length of string.
strncat	<code>char *strncat(char * s1, const char *s2, size_t n);</code>	Concatenate up to n characters from string s2 to the end of string s1.
strncmp	<code>int strncmp(const char * s1, const char *s2, size_t n);</code>	Compare strings to n characters.
strncpy	<code>char *strncpy(char * dst, const char * src, size_t n);</code>	Copy a string of n or fewer characters.
strpbrk	<code>char *strpbrk(const char *s, const char *accept);</code>	Find any character of string accept.
strrchr	<code>char *strrchr(const char *s, int c);</code>	Return last instance of character c in string s.
strspn	<code>size_t strspn(const char *s, const char *accept);</code>	Return the length of the initial segment of string s, which consists entirely of characters from (not from) string accept.
strstr	<code>char *strstr(const char *haystack, const char *needle);</code>	Find needle in haystack.

<string.h> String Functions - Non-ANSI

strrev	<code>char * strrev(char * s);</code>	Reverse a string.
--------	---------------------------------------	-------------------

6.4 Math Functions

<math.h> Math Functions - Non-ANSI

random	<code>int random(void);</code>	Returns rand(), not a different algorithm as in ANSI.
srandom	<code>int srandom(int);</code>	Calls srand(int), not a different algorithm.

6.5 Character Type Functions

<ctype.h> Character Type Functions - ANSI

isalnum	<code>isalnum(c);</code>	Is alphanumeric.
isalpha	<code>isalpha(c);</code>	Is alphabetic.
iscntrl	<code>iscntrl(c);</code>	Is control.
isdigit	<code>isdigit(c);</code>	Is a digit.
isgraph	<code>isgraph(c);</code>	Is a graphics character.
islower	<code>islower(c);</code>	Is lowercase.
isprint	<code>isprint(c);</code>	Is printable.
ispunct	<code>ispunct(c);</code>	Is punctuation.
isspace	<code>isspace(c);</code>	Is space or whitespace.
isupper	<code>isupper(c);</code>	Is uppercase.
isxdigit	<code>isxdigit(c);</code>	Is a hexadecimal digit (0-9, a-f, A-F).

<ctype.h> Character Type Functions - Non-ANSI

tolower	<code>tolower(c);</code>	If uppercase, convert to lowercase (macro).
toupper	<code>toupper(c);</code>	If lowercase, convert to uppercase (macro).

6.6 I/O Interfaces (File and Serial)

Files can be read, written, and/or created on the /flash disk (non-volatile RAM) or on the /ram disk (temporary). The serial port can also be opened for read, write, or both at “tt0:”.

<fcntl.h> I/O Interfaces (File and Serial) - ANSI		
creat	<code>int fd = creat(const char *path, unsigned short amode);</code>	
open	<code>int fd = open(const char *path, int oflag, ... mode_t mode);</code>	Opens a file.
	Ex: <code>int fd = open("tt0:", O_RDWR);</code>	Opens the serial port, read/write mode.
	Ex: <code>int fd = open("filename", O_WRONLY O_CREAT, 0664);</code>	Returns an error or a file descriptor.

<fcntl.h> I/O Interfaces (File and Serial) - Non-ANSI		
fcntl	<code>int ret = fcntl(int fd, F_GETFL);</code>	Always returns zero. See Section 1.2.3.
	<code>int ret = fcntl(int fd, F_SETFL, O_NONBLOCK);</code>	Enables non-blocking I/O.
	<code>int ret = fcntl(int fd, F_SETFL, 0);</code>	Disables non-blocking I/O.
	NOTE: No other fcntl commands are allowed.	

<unistd.h> I/O Interfaces (File and Serial) - ANSI		
chdir	<code>int chdir(const char *path);</code>	Change current directory.
close	<code>void close(int fd);</code>	Close a file.
dup	<code>int dup(int fd);</code>	Copy a file descriptor.
link	<code>int link(const char *oldpath, const char *newpath);</code>	Create new path linked to old.
lseek	<code>off_t curpos = lseek(int fd, off_t offset, int whence);</code>	Move the file pointer.
read	<code>ssize_t nread = read(int fd, void *buf, int len);</code>	Read from a file.
rmdir	<code>int rmdir(const char *path);</code>	Remove empty directory.
unlink	<code>int unlink(const char *fname);</code>	Delete a file.
write	<code>ssize_t nwritten = write(int fd, const void *buf, int len);</code>	Write to a file.

<unistd.h> I/O Interfaces (File and Serial) - Non-ANSI		
ioctl	<code>int rv = ioctl(int fd, IO_INPPEND, NULL);</code>	Input pending.
	<code>int rv = ioctl(int fd, IO_OUTPEND, NULL);</code>	Output pending.
	<code>int rv = ioctl(int fd, IO_GSTATUS, int *ret);</code>	Status of DSR, CD, RI, flow.
	<code>int rv = ioctl(int fd, IO_GTTY, int *ret);</code>	Get current baud/parity/charsize.
	<code>int rv = ioctl(int fd, IO_STTY, int *ret);</code>	Set baud/parity/charsize, etc.
	NOTE: <termios.h> contains constants for IO_GTTY/IO_STTY.	
sync	<code>void sync(void);</code>	Write all pending data to all files on all devices.

For `IO_GTTY`, you should AND the return values with the status constants to determine if they are set. For `IO_STTY`, you should OR the constants together to define your settings. The example below sets the baud rate to 19200, enables CTS/RTS flow control, enables parity (and sets it for Even), sets the character size for 8 bits, and sets the stop bit count to 1.

```
int newset=B19200|CRTSCTS|PARENB|CS8;
int fd=open("tt0:",O_RDWR);
ioctl(fd,IO_STTY,&newset);
```

<termios.h> IO_GTTY/IO_STTY Constants

B300, B600, etc.	Sets the baud rate. The possible values are: B300, B600, B1200, B2400, B4800, B9600, B19200, B38400, B57600, B115200, B230400. <i>AND</i> the result of <code>IO_GTTY</code> with <code>CBAUD</code> to get the baud rate field.
CS7, CS8	Sets the character size. CS8 is the default. <i>AND</i> the result of <code>IO_GTTY</code> with <code>CSIZE</code> to get the character size field.
CSTOPB	Sets the MSS for two stop bits (one stop bit is the default).
CRTSCTS	Enables CTS/RTS (hardware) flow control.
CXONXOFF	Enables XON/XOFF (software) flow control.
CDTRDSR	Enables DTR/DSR (hardware) flow control.
SER_ECHO	Automatically echoes serial input.
SER_PASSFLOW	Adds XON/XOFF characters to stream.
PARENB	Enables parity and sets it for Even, unless <code>PARODD</code> is also set. [<code>PARENB</code> alone = Even]
PARODD	Changes to Odd parity. <code>PARENB</code> must also be set. [<code>PARENB</code> + <code>PARODD</code> = Odd]
PAREXT	Makes <code>PARODD</code> toggle between Mark and Space. <code>PARENB</code> must also be set. [<code>PARENB</code> + <code>PAREXT</code> = Space; <code>PARENB</code> + <code>PAREXT</code> + <code>PARODD</code> = Mark]

AND any of the serial status constants with the “ret” value.

```
int ret;
int fd=open("tt0:",O_RDWR);
int rv=ioctl(fd,IO_GSTATUS,&ret);
if(ret & SERIAL_IDTR)
    printf("DSR is asserted\n\r");
```

<termios.h> IO_GSTATUS Constants

SERIAL_CD	Carrier Detect is asserted.
SERIAL_IDTR	Input on the DSR pin is asserted.
SERIAL_IGAG	Input is flow-controlled (we have stopped receiving).
SERIAL_ISTS	Our CTS Input is asserted.
SERIAL_ODTR	Our current DTR output is asserted.
SERIAL_OGAG	Output is flow-controlled (the device attached has stopped rcv).
SERIAL_ORTS	Our RTS output is asserted.

<stdio.h> I/O Interfaces (File and Serial) - ANSI		
clearerr	<code>void clearerr(FILE *fp);</code>	Clear any errors on file stream.
fclose	<code>int fclose(FILE *fp);</code>	Close this file.
feof	<code>int feof(FILE *fp);</code>	True if end of file reached.
ferror	<code>int ferror(FILE *fp);</code>	True if there's an error on that file stream.
fflush	<code>int fflush(FILE *fp);</code>	Flush any pending output to the device/file.
fgetc	<code>int fgetc(FILE * fp);</code>	Get next character from file (blocking input).
fgetpos	<code>int fgetpos(FILE *fp, long *off);</code>	Retrieve current file position.
fgets	<code>char * fgets(char *s, int n, FILE *fp);</code>	Read line from file (up to \n or \r). See Section 1.2.3.
fileno	<code>int fileno(FILE *fp);</code>	Return file descriptor for this FILE.
fputc	<code>int fputc(int c, FILE * fp);</code>	Send a character to file (blocking output).
fputs	<code>int fputs(const char *s, FILE *fp);</code>	Send a string to file (blocking output).
fread	<code>size_t fread(void *buf, size_t elsize, size_t nelems, FILE * fp);</code>	Read a block of data from file (blocking input).
ftell	<code>long ftell(FILE *fp);</code>	Get current position in file.
fseek	<code>int fseek(FILE *fp, long off, int origin);</code>	Set file position from beginning or end of file.
fsetpos	<code>int fsetpos(FILE *fp, long off);</code>	Set file position.
fwrite	<code>size_t fwrite(const void *buf, size_t elsize, size_t nelems, FILE * fp);</code>	Write a block of data to a file (blocking output).
getchar	<code>int getchar();</code>	Get character from stdin (blocking input).
perrorr	<code>void perror(const char *msg);</code>	Print message followed by strerror(errno) to stderr.
putchar	<code>int putchar(char c);</code>	Put character to stdout (blocking output).
rename	<code>int rename(const char *oldname, const char *newname);</code>	Rename a file.
rewind	<code>void rewind(FILE *fp);</code>	Set file pointer to beginning of the file.

<stdio.h> I/O Interfaces (File and Serial) - Non-ANSI		
fopen	FILE *fopen(const char *name, const char *mode);	Open a file. NOTE: our fopen only supports a single character mode (r , w or a), and files are always opened in binary mode. No text translation takes place. See Section 1.2.3.
fprintf	int fprintf(FILE * fp, const char *fmt, ...);	Formatted print to file stream. NOTE: None of the printf / scanf functions support float or double variables.
getc	int getc(FILE * fp);	Get character from file, NOT implemented as a macro.
printf	int printf(const char *fmt, ...);	Formatted print to console.
putc	int putc(char c, FILE *fp);	Put character to file, NOT a macro.
puts	int puts(char * str);	Print string to console (automatically adds \n\r).
sscanf	int sscanf(const char *str, const char *fmt, ...);	Formatted scan from string. See Section 1.2.3.
setbuf	int setbuf(FILE *fp, char *buf);	Can only be used set buffer to NULL. See Section 1.2.3.
sprintf	int sprintf(char *buf, const char * fmt, ...);	Formatted print to string.
vfprintf	int vfprintf(FILE * fp, const char *fmt, va_list args);	Print formatted output of varargs to file-varargs.
vprintf	int vprintf(const char *fmt, va_list args);	Print formatted output of varargs.
vsprintf	int vsprintf(char * str, const char *fmt, va_list args);	Print formatted output of varargs to string-varargs.

<sys/stat.h> I/O Interfaces (File and Serial) - Non-ANSI		
chmod	int chmod(const char *path, mode_t mode);	Change mode of file.
fstat	int fstat(int fd, struct stat *buf);	File status, from file descriptor.
mkdir	int mkdir(const char *path, mode_t mode);	Make a directory, with a specified mode.
stat	int stat(const char *file_name, struct stat *buf);	Get file status.

Note: Only world read matters, since PUC can only support two levels of privilege: root and anonymous. As such, although you can set other modes on files, only read world and read/write/execute root permissions will be obeyed by the filesystem.

6.7 Network Socket Functions

These functions are Berkeley Sockets functions. For more information, see Stevens (bibliographic information is included in Appendix C) or the UNIX man pages.

<sys/socket.h> Network Socket Functions - Non-ANSI		
accept	<code>int accept (int fd, struct sockaddr_in *addr, int *addrlen);</code>	Allocate a new file descriptor for first pending connection.
bind	<code>int bind (int fd, struct sockaddr *name, int namelen);</code>	Assign name to unnamed socket.
connect	<code>int connect (int fd, struct sockaddr *name, int namelen);</code>	Make a connection to another socket.
gethostbyname	<code>HOSTENT *gethostbyname (char *name);</code>	Look up hostent in nameserver.
gethostname	<code>int gethostname (char *name, int namelen);</code>	Get local host name.
getprotobyname	<code>PROTOENT *getprotobyname (const char *protoname);</code>	Look up protocol name.
getservbyname	<code>SERVENT *getservbyname (const char *servname, const char *protoname);</code>	Look up service name.
getsockname	<code>int getsockname (int sockfd, struct sockaddr *name, int *namelen);</code>	Get socket name.
inet_addr	<code>unsigned long inet_addr (char *cp);</code>	ASCII to internet address.
inet_ntoa	<code>char *inet_ntoa (unsigned long /* struct in_addr*/ in);</code>	Internet address to ASCII.
listen	<code>int listen (int fd, int backlog);</code>	Set to accept queue.
recv	<code>int recv (int fd, char *buf, int len, int flags);</code>	Read message from connected socket.
recvfrom	<code>int recvfrom (int fd, char *buf, int len, int flags, struct sockaddr_in *from, int *fromlen);</code>	Read from any socket.
send	<code>int send (int fd, char *msg, int len, int flags);</code>	Write to connected socket.
sendto	<code>int sendto (int fd, char *msg, int len, int flags, struct sockaddr_in *to, int tolen);</code>	Write to any socket.
setsockopt	<code>int setsockopt (int fd, int level, int optname, char *optval, int optlen);</code>	Set socket options.
shutdown	<code>int shutdown (int s, int how);</code>	Shut down a socket.
socket	<code>int socket (int domain, int type, int protocol);</code>	Create a socket endpoint.

<netinet/in.h> Network Socket Functions - Non-ANSI		
htonl	<code>ulong htonl(ulong);</code>	Host to network byte-order long.
htons	<code>ushort htons(ushort);</code>	Host to network byte-order short.
ntohl	<code>ulong ntohl(ulong);</code>	Network to host byte-order long.
ntohs	<code>ushort ntohs(ushort);</code>	Network to host byte-order short.

6.8 Directory Read Functions

<direct.h> Directory Read Functions - ANSI		
closedir	int closedir(DIR *dirp);	Close directory.
opendir	DIR *opendir(const char *filename);	Open a directory to read, Null if error.
readdir	struct dirent *readdir(DIR *dirp);	Read next directory entry.
rewinddir	void rewinddir(DIR *dirp);	Reset to first directory entry.
seekdir	void seekdir(DIR *dirp);	Seek to the results of a previous telldir.
telldir	long telldir(DIR *dirp);	Retrieve current position within directory list.

6.9 NVR/Flash

To keep persistent data across reboots, write files to the Flash disk (/flash/filename). There will be approximately one second of lag time as files are written.

Note: *The Flash disk has a large but limited read/write life cycle.*

6.10 Time Functions

You must configure and enable a timeserver for time functions to give meaningful time information. See the *MSS Reference Manual* for information on how to configure your timeserver options.

If you use an NTP (Network Time Protocol) server, the date and time will be correct, provided the NTP server is on-line when the MSS boots. If not, the MSS will check periodically for it to become available. If you use a daytime server, the time of day will be set, but not the current date. To correctly report both the date and time, use the **Change Timeserver** command to configure your MSS for NTP with the appropriate GMT offset.

```
Local>> CHANGE TIMESERVER NTP BROADCAST
Local>> CHANGE TIMESERVER GMTOFFSET -7
```

In the example above, the **Broadcast** keyword could be replaced with the **IP** keyword and the numeric IP address of the desired timeserver (that is, CHANGE TIMESERVER NTP IP 192.0.1.22).

<unistd.h> Time Functions - ANSI		
sleep	ulong sleep(ulong seconds);	

<unistd.h> Time Functions - Non-ANSI		
sleepMS	ulong sleepMS(ulong milliseconds);	NOTE: The system clock on the MSS doesn't allow precision timing for intervals less than 20msec.

<time.h> Time Functions - ANSI		
asctime	char *asctime(struct tm *ts);	ASCII date/time from time structure.
ctime	char *ctime(ulong *rv);	ASCII date/time.
gmtime	struct tm *ts = gmtime(ulong *rv);	Time structure for current Greenwich Mean.
localtime	struct tm *ts = localtime(ulong *rv);	Time structure for current local time.
mktime	ulong rv = mktime(struct tm *ts);	Time in seconds from a time structure.
time	ulong rv = time(NULL &rv);	Seconds since 1/1/1970, learned from NTP, RTC, daytime, etc. Daytime servers will update the time-of-day, but not the date.

Note: *If you are using NTP and time (NULL) returns a value less than 914544000 (Jan. 1, 1999), then the time should be ignored because it cannot be valid.*

<time.h> Time Functions - Non-ANSI		
clock	ulong rv = clock();	System timeticks since boot. For timeticks, use CLOCKS_PER_SECOND. NOTE: ANSI C specifies microseconds. Since our resolution is currently 10 milliseconds, this gives us much more range before it overflows 32 bits.
difftime	long difftime(time_t t1, time_t t2);	Difference in seconds between two times. NOTE: ANSI C specifies a double return value, but we don't support doubles.

6.11 Debugging Functions

<assert.h> Debugging Functions - ANSI		
assert	assert(expression);	If expression evaluates to false (or zero), prints <i>Assertion failed: expression, file xxx, line n.</i> and aborts the program. If the NDEBUG macro is defined, none of the assert messages will appear, nor will the program abort if the expression evaluates to false.

6.12 SNMP Functions

The MSS operating system includes a handler or *agent* that will reply to queries for some SNMP objects. See the *MSS Reference Manual* for a list of the object databases, or *MIBs*, that the MSS can handle internally.

If the MSS receives an SNMP query for an object that is after the last MSS-native MIB object, an SDK application can obtain and reply to that query. This allows SDK users to implement their own MIBs, as long as the MIBs are outside of the MIBs that the MSS supports locally. If the SDK application acts as a backup SNMP handler, it is responsible for replying to all queries that it receives. This could include generating the appropriate SNMP errors for unknown or malformed objects, enforcing any security needed, and so on. Familiarity with SNMP is needed to implement a custom MIB using the SDK.

Note: See chapter 25 of *Stevens' TCP/IP Illustrated, Vol. 1*, for more information about the SNMP protocol.

The sample SNMP code on the SDK CD-ROM (**snmp.c**) provides basic decoding, encoding, MIB setup, request handling, and trap sending code; you can start with this code and modify it to meet your needs. If you do not need support for the more complex objects, you can simply change the MIB and handler at **mymib.c**, following the example there.

<socket.h> SNMP Functions - Non-ANSI		
snmpDeregister	<code>void snmpDeregister(int fd);</code>	Closes the SNMP connection.
snmpRead	<code>int len = snmpRead(int fd, char *buffer, int max_buffer_len);</code>	Receives incoming SNMP packets. The first byte placed in the passed-in buffer will be the first byte of the snmp portion of the UDP packet. Non-blocking. Returns 0 if nothing is pending. Returns <0 if there is an error; call snmpDeregister(). Returns the length of the data copied to the user buffer.
snmpRegister	<code>int fd = snmpRegister(void);</code>	Registers your application as the handler for all non-built-in OIDs. fd is >=0 for success or -1 for failure.
snmpSendTrap	<code>int snmpSendTrap(char *host, char *trap_pkt, int len);</code>	Allows your application to send traps (alerts) to an arbitrary host. The host is either a text hostname (if there is a DNS available for name resolution) or an ASCII string host IP address. If rv < 0, there was an error.
snmpWrite	<code>int rv = snmpWrite(int fd, char *reply_pkt, int len);</code>	Replies to the last-read packet. If rv < 0, there was an error; call snmpDeregister() and start over.

Note: If your application does not know how to handle a packet, it should be configured to send the appropriate SNMP error frame back to the sender. The MSS will not reply to any SNMP queries accepted by a custom application.

A: Technical Support

A.1 The SDK Support Plan

Lantronix supports the SDK on a fee basis. Please consult your service agreement or contact Lantronix at 949/453-3990 for more information.

SDK Sales: TBD
Internet: sdk-sales@lantronix.com

SDK Support: TBD
Internet: TBD

A.2 Problem Report Procedure

When you report a problem, please provide the following information:

- ◆ Your name, and your company name, address, and phone number
- ◆ Lantronix MSS model number
- ◆ Lantronix MSS serial number
- ◆ Software version (use the **Show Version** command to display)
- ◆ Network configuration, including the information from a **Netstat** command
- ◆ Description of the problem
- ◆ Example source code that demonstrates the problem
- ◆ Debug report (stack dump), if applicable
- ◆ Status of the unit when the problem occurred (please try to include information on user and network activity at the time of the problem)
- ◆ A description of the serial device connected to the MSS serial port, if applicable.

A.3 Other Contact Information

Address: 15353 Barranca Parkway, Irvine, CA 92618 USA

Phone: 949/453-3990

Fax: 949/453-3995

World Wide Web: <http://www.lantronix.com>

North American Direct Sales: 800/422-7055

North American Reseller Sales: 800/422-7015

North American Sales Fax: 949/450-7232

Internet: sales@lantronix.com

International Sales: 949/450-7227

International Sales Fax: 949/450-7231

Internet: intsales@lantronix.com

Technical Support: 800/422-7044 or 949/453-3990

Technical Support Fax: 949/450-7226

Internet: support@lantronix.com

B: Disk Management

B.1 Overview

The Disk commands are very similar to the file management commands in UNIX environments. If you are not familiar with UNIX environments, refer to an introduction to UNIX before you proceed.

Note: *Unlike the similar UNIX commands, each disk command must be preceded by the word `DISK`.*

Each of the MSS disks support all of the commands listed in this appendix, unless otherwise noted. Some of the Disk commands support the various flags used by the same UNIX command. Additionally, flags can be grouped together. For example, `-lt` would be equivalent to `-l` and `-t`.

Some of the Disk commands support the UNIX wildcard (*) convention. When you enter a * into a command, it specifies that the command will react to any file having a similar extension, prefix, suffix, or pattern. For example, the command `DISK LS *.h` will list all of the header files from your current working directory (provided they have the `.h` extension).

Unlike UNIX Disk commands, MSS Disk commands are not case-sensitive. In the examples, required command keywords appear in uppercase, and user-supplied parameters appear in lowercase. However, the commands can be entered in any case.

B.2 The Filesystem

There are four directories that are used by the SDK. The MSS contains three directories, two of which can be used for your custom application files. The fourth directory lives on the MSS's loadhost. The directories are scanned at runtime in the following order:

/ram The RAM disk stores temporary information for the MSS. The MSS will hold information stored on this disk until it is turned off or rebooted. At startup, the RAM disk will be empty. FTP connections automatically use the RAM disk as the default working directory.

/flash Flash is re-writable memory that allows you to customize your MSS. Any data that you want the MSS to save after it is rebooted should be put on the Flash disk. When your applications are stable, you can put them on the MSS Flash disk.

Depending upon your custom settings, the Flash disk may contain new settings for the MSS. At start-up, the MSS will change the MSS settings based on what it finds on the Flash disk. If the MSS does not find any new information on the Flash disk, it will use the files stored in ROM.

Note: *The Flash disk has a large but limited write life.*

/rom The ROM disk is read-only and cannot be modified by users. It contains all of the standard header files (*.h).

/tftpboot/puc/

The loadhost's `/tftpboot/puc` directory is a good place to work on files. Any source or include files that are placed here will be loaded into the MSS automatically.

When looking for include files or source files, the MSS will look at the RAM disk, then the Flash disk, then the ROM disk. If it has not located the files, it will use TFTP to try to look for the files on the configured loadhost. There are no files on the loadhost by default. You must place files there explicitly. You must also make sure the files have world read permissions (the default is no world privileges).

B.3 Using Disks in PUC

Disk files can be read from or written to from PUC using ANSI standard file commands. For example:

```
open("/flash/index.txt", O_RDONLY);
```

Directory access functions are available in `<dirent.h>`.

B.4 Disk Commands

DISK CAT {file}

Allows you to display an entire file in your terminal window.

```
Local>> DISK CAT /flash/index.txt
```

DISK CD {directory}

Allows you to change the current working directory.

```
Local>> DISK CD /ram
```

DISK CHMOD {code} {file}

Allows you to change permissions for a file or directory. To assign permissions, enter a 3-digit number. The first digit represents the owner's permissions. The second digit represents the group's permissions. The third digit represents the world's permissions.

```
Local>> DISK CHMOD 755 /flash/index.txt
```

Digit	Meaning
0	No permissions.
1	Execute permission only.
2	Write permission only.
3	Write and Execute permissions.
4	Read permission only.
5	Read and Execute permissions.
6	Read and Write permissions.
7	All permissions.

DISK CP {file1} {file2}

Allows you to copy or move a file. To copy a file, enter the filename for *file1* and the new file name as *file2*. To move a file, specify the filename as *file1* and the destination directory as *file2*. The following examples rename a file and then move it to a new directory.

```
Local>> DISK CP customapp.txt app.txt
Local>> DISK CP app.txt /ram
```

DISK DF {disk}

Displays the blocks of free disk space on the MSS disks. When you add the *-i* switch, the display includes in the display the number of inodes used versus the number still available.

```
Local>> DISK DF -i /flash
```

DISK FSCK

Checks the MSS filesystem and corrects any problems.

```
Local>> DISK FSCK
```

DISK FORMAT FLASH

Allows you to format or erase the MSS /flash disk.

```
Local>> DISK FORMAT FLASH
```

DISK LN {flag} {file1} {file2}

Creates a hard or soft link for files, thus linking a file or set of files to another file. Using no flag creates a hard link. Adding the *-s* flag creates a soft link.

```
Local>> DISK LN newapp.c unp.h
Local>> DISK LN -s newapp.c unp.h
```

DISK LS {flag} {file}

Lists the contents of a directory. The available flags are:

-l	Returns a list in long form, which includes information about modification date, size, owner, and permissions.
-t	Sorts the list by modification date, with the newest file appearing first.
-r	Reverses the order of the file listing. For example, if <i>-t</i> was also specified, <i>-r</i> would list the oldest file first.

```
Local>> DISK LS -l /rom/puc/include
```

DISK MKDIR {directory}

Creates a new directory on the MSS RAM disk or Flash disk.

```
Local>> DISK MKDIR /flash/customapps/
```

DISK MORE {file}

Displays the contents of a file on the terminal, 24 lines of text at a time. Normally the display pauses after each screen and prints "--MORE--" at the bottom of the screen. To access the next screen, press the Space bar. TO abort, press Ctrl-C

```
Local>> DISK MORE /rom/puc/unp.h
```

DISK MV {file} {target}

Moves files or directories on the MSS RAM and Flash disks. You can also rename files with this command by inserting the new filename for *target*.

```
Local>> DISK MV /rom/puc/unp.h /flash/customapps
```

DISK OD {flag} {file}

Displays the contents of the specified file in bytes format. The possible flags are:

```
-b    Prints the bytes in octal format.  
-ct   Prints the bytes in ASCII format.  
-x    Prints the bytes in hexadecimal format.
```

```
Local>> DISK OD -x unp.h
```

DISK PWD

Shows you the full pathname of your current directory.

```
Local>> DISK PWD
```

DISK RM {flag} {file}

Removes files and/or directories from the MSS RAM and Flash disks.

```
-i    Prompts for a Y (yes) or N (no) before the file is removed.  
-r    Removes an entire directory and all of its subdirectories.
```

```
Local>> DISK RM -i unp.h
```

DISK RMDIR {directory}

Removes a directory on the MSS RAM or Flash disks. This command can only be used if the directory is empty. If the directory is full, you must add the **DISK RM -rf {directory}** command.

```
Local>> DISK RMDIR /flash/customapps
```

DISK SYNC

Forces the MSS to write files to flash immediately. Normally, when the MSS is rewriting files to the Flash disk, it will buffer data before initiating a write sequence. Write sequences are automatically written after 5 seconds of disk inactivity.

```
Local>> DISK SYNC
```

C: References

C.1 Stevens' Network Examples

All example code referred to as *Stevens'* in this manual is taken from the following book:

Stevens, W. Richard
UNIX Network Programming, 2nd Ed.
Volume I, *Networking APIs: Sockets and XTI*
© 1998 Prentice-Hall PTR, Upper Saddle River, NJ USA
ISBN 0-13-490012-X

Source code is available from the following WWW address:

<http://www.kohala.com/start/unpv12e.html>

C.2 EiC Documentation

Portable Universal C (PUC) is based on Extensible interactive C (EiC). The most current version of the EiC documentation can be found on the World Wide Web at:

<http://www.anarchos.com/eic/>

C.3 Other Useful Sources

The following reference materials were not directly used to complete this manual, but are valuable resources for C programmers. You may wish to refer to these books for additional help.

Note: *UNIX users can look for specific terms in the UNIX man pages.*

TCP/IP Illustrated

Stevens, W. Richard
TCP/IP Illustrated
Volume I, *The Protocols*
© 1984 Addison Wesley Longman, Inc., Reading, MA USA
ISBN 0-201-63346-9

The C Programming Language

Kernighan, Brian and Dennis M. Ritchie
The C Programming Language, 2nd Ed.
© 1998 Prentice-Hall PTR, Englewood Cliffs, NJ USA
ISBN 0-13-110362-8 (paperback)
ISBN 0-13-110370-9 (hardcover)

Function List

A

abort 26
abs 26
accept 33
asctime 35
assert 35
atexit 26
atoi 26
atol 26

B

bind 33
bsearch 26

C

calloc 26
chdir 29
chmod 32
clearerr 31
clock 35
close 29
closedir 34
connect 33
creat 29
ctime 35

D

difftime 35
dup 29

E

exit 26

F

fclose 31
fcntl 29
feof 31
ferror 31
fflush 31
fgetc 31
fgetpos 31
fgets 31
fileno 31
fopen 32
fprintf 32
fputc 31
fputs 31
fread 31
free 26

fseek 31
fsetpos 31
fstat 32
ftell 31
fwrite 31

G

getc 32
getchar 31
gethostbyname 33
gethostname 33
getprotobyname 33
getservbyname 33
getsockname 33
gmtime 35

H

htonl 33
htons 33

I

inet_addr 33
inet_ntoa 33
ioctl 29
isalnum 28
isalpha 28
iscntrl 28
isdigit 28
isgraph 28
islower 28
isprint 28
ispunct 28
isspace 28
isupper 28
isxdigit 28
itoa 27

L

labs 26
link 29
listen 33
localtime 35
lseek 29
ltoa 27

M

malloc 26
memchr 27
memcmp 27

memcpy 27
memmove 27
memset 27
mkdir 32
mktime 35

N

ntohl 33
ntohs 33

O

open 29
opendir 34

P

perror 31
printf 32
putc 32
putchar 31
puts 32

Q

qsort 26

R

rand 26
random 28
read 29
readdir 34
realloc 26
recv 33
recvfrom 33
rename 31
rewind 31
rewinddir 34
rmdir 29

S

seekdir 34
send 33
sendto 33
setbuf 32
setsockopt 33
shutdown 33
sleep 34
sleepMS 34
snmpDeregister 36
snmpRead 36
snmpRegister 36

snmpSendTrap 36
snmpWrite 36
socket 33
sprintf 32
srand 26
srandom 28
sscanf 32
stat 32
strcat 27
strchr 27
strcmp 27
strcpy 27
strcspn 27
strerror 27
strlen 27
strncat 27
strncmp 27
strncpy 27
strpbrk 27
strrchr 27
strrev 28
strspn 27
strstr 27
strtol 26
strtoul 26
sync 29

T

telldir 34
time 35
tolower 28
toupper 28

U

ultoa 27
unlink 29
utoa 27

V

vfprintf 32
vprintf 32
vsprintf 32

W

write 29

Index

Symbols

#define	19
#include	2, 5, 8, 9
/flash	8, 14, 15, 29, 34, 39
/ram	8, 14, 29, 39
/rom	8, 25, 39
/tftpboot/puc	5, 6, 8, 10, 11, 13, 14, 19, 40

A

a.out	19
accept()	3
ANSI	19, 25, 40
API	25
Arguments	7, 12, 14, 15
Assigning values to integers	9
auto (-auto) flag	7, 14
Autoloading	2, 13
Autorun mode	7, 14, 15
Disabling	14
Autorun mode, disabling	16

B

Berkeley sockets	33
Bi-directional echo	13
Binary mode	3
Bugs	3
Byte ordering	2

C

C language errors	17
C, differences from	2
cc command	7, 8, 14, 15
Character type functions	28
Checking software version	4
Child process	2
close()	13
Command line mode	7, 11, 12
Compile-time errors	19
Configuration	7–16
Creating files	10

D

Daytime server	15, 34
DDEFINEME macro	14
Debugging	17
Functions	35

Declaring integers	9
Directory read functions	34
Disk commands	39, 40
CAT	40
CD	40
CHMOD	40
CP	41
DF	41
Format Flash	41
FSCK	41
LN	41
LS	41
MKDIR	41
MORE	42
MV	42
OD	42
PWD	42
RM	42
RMDIR	42
SYNC	42
Display terminal	15
do_buffer.c	13
do_socket()	13
Double data type	2

E

EiC	2, 43
EOF	14
Errors	17, 19
Error-trapping functions	13
Examples	
Autorun	14, 15
Hello world	8
Interactive mode	9
Network socket connection	12
Network/serial	13
Serial output	11
Exit command	8, 10

F

fcntl()	3
fgets()	3
File permissions	6, 17
File pointers	3
Filesystem	39
Flash disk	15, 29, 34, 39
Flash functions	34

fopen()	3
Forking	2
FTP	4, 15
Functions	2
Built-in	25
Character type	28
Debugging	35
Directory read	34
I/O interfaces	29
Math	28
Network socket	33
NVR/Flash	34
Prototype	18
SNMP	36
Standard library	26
String	27
Time	34
Wrapper (error trap)	13

G

Global versus local variables	10
Goto	2

H

Header files	2, 19, 25
Help	17

I

I/O functions	29
in.h	5
Include files	5, 25
Integer	9
Interactive mode	7, 8, 9, 18
IO_GTTY constants	30
IO_STTY constants	30

L

Label	2
Lantronix contact information	37
Linking errors	17
Loadhost	4, 5, 6, 10, 40
Local versus global variables	10
Local> prompt	7, 9

M

main()	10, 11, 17
Makefile	2
Math functions	28
MIB	36

N

Network sockets	33
Network Time Protocol	34
Newline character	3, 9
noauto (-noauto) flag	14, 16
Non-blocking	13
NTP server	34
Null device	14
NVR functions	34

O

Opening files	3
Overview	
PUC	2
SDK	1

P

PC host	19
Port 9877	13
printf()	8, 9
Privileged status	7, 8, 11, 12, 15
Prototype definitions	2, 10, 18
PUC	2, 4, 7
Disks	40
Distribution files	5
Executing code	10, 16
Exiting	8, 10
Limitations	2
Non-standard behaviors	3
UNIX installation	6
Windows installation	6
PUC> prompt	7, 8, 9

R

Ramdisk	29, 39
read()	3
recv()	3
recvfrom()	3
Reference sources	43
Return character	3, 9
Return value	9
Return values	8, 18, 30
ROM disk	5, 39
Runtime library	25
Runtime values	18

S

Sample code

daytimetcpcli.c	23
directed.c	22
dirwatch.c	21
do_buffer.c	22
do_filebuf.c	22
echo.c	21
fractoa.c	21
ftpcli.c	22
getip.c	21
inet_pton.c	23
ioctl.c	21
log.c	21
mailcli.c	22
Miscellaneous	21
Network (PUC)	22
Network (Stevens)	23
param.c	21
pipec.c	22
rs491.c	22
snmp.c	22
spin.c	21
tcp_connect.c	22
tcpcli.c	22
tcpserv.c	22
tcpserv01.c	23
timecli.c	21
udpserv.c	22
udpserv01.c	23
utils.c	21
wrapper.c	23
Scope	10
SDK overview	1
Search path	8, 19
sendto()	3
Serial port	7, 11, 13, 15, 19, 29
Serial settings	11, 15
setbuf()	3
Show command	10, 18
Signal	2
SNMP functions	36
SNMP Object database	36
snmp.c	36
Sockets	26
spin.c	11
sscanf()	3
Standard library functions	2, 26
startpuc.h	5, 9, 17, 19, 25, 26
stderr	14

stdin	14
stdout	14
Stevens	33, 43
String functions	27

T

TCP client	13
TCP server	13
tcp_connect()	12
tcpconnect.c	12
tcpserv.c	13
Telnet	4, 8, 9, 11, 13
Terminal	11
Text editor	4
TFTP server	4, 5, 6, 40
Time functions	34
timecli.c	12
Timeserver	34
Trace command	18
tt0	19, 29

U

UDP socket	3
UNIX	13, 14, 19, 39
Installation	6
unp.h	5, 19, 25, 26

V

Values	18
Variables	18
Global versus local	10
void	9

W

Windows installation	6
wrapper.c	13

