Linux Developer's Manual

Technologic Systems, Incorporated

16610 East Laser Drive, Suite 10
Fountain Hills, AZ 85268

480-837-5200
FAX 837-5300

info@embeddedx86.com

http://www.embeddedx86.com/

This revision of the manual is dated

February xx, 2004

All modifications from previous versions are listed in the appendix.

Table Of Contents

1 Introduction

This manual is a brief introduction to the use of the Linux on a Technologic Systems' Single Board Computer (SBC). Many technical questions are answered within this document, along with some example code in C. This manual is not meant as a tutorial to Linux (or Linux Development). However, this manual was written to help those who are becoming familiar with Linux, and not for those who are 'experts' with Linux. More information for those programs mentioned can be gleaned form man pages, or by doing a Google search. The books listed under section 6.1, particularly the book Understanding Redhat Linux, should be read by those who have *no* familiarity with Linux before even attempting to boot up a Single Board Computer with Linux. That said, an understanding of Linux would not be possible without having a Linux desktop to work on.

Having a Linux desktop is essential to using Linux on an SBC. Customers can either install Linux on a new desktop, install Knoppix, or simply buy a secondary hard-drive for their PC and install Linux on it, thus having a dual-booted system. As a Microsoft operating system won't even recognize the Linux partitions on a compact flash, to even edit a a configuration file, a Linux operating system is needed.

The Linux distribution used by the staff of Technologic Systems on their workstations is Mandrake or Debian (stable). Other customers use Suse, Redhat, and even Gentoo. For those new to Linux and Linux development, it is suggested that they purchase a copy of Mandrake or Redhat. Both have been found to be the easiest to install, work with, and manage. Both distributions offer a surfeit of documentation and support packages.

Currently, Linux kernel 2.4.23 is used on all Single Board Computers. It is possible to use other versions of the Linux kernel, such as those from the 2.2 kernel tree, or 2.6 tree. Some customers have felt more comfortable with a distribution kernel, such as a Debian or Redhat kernel. As long as an appropriate version of modutils are used, there should be little, if any problems with using a different kernel. Those programs that could be affected by a kernel switch would be HotPlug scripts, PCMCIA card service utilities, and the devfs daemon.

1.1 Booting

Compact Flash (CF) cards use Syslinux as the boot-loader to invoke and run Linux. Syslinux only understands the FAT file system. Both the kernel (bzImage) and Syslinux's configuration file (syslinux.cfg) must reside on a FAT filesystem. The CF card is divided into a 2 Megabytes DOS partition, with the remainder of the card dedicated to Linux.

Disk On Chips, on the other had, use GRUB as the boot-loader to boot Linux. GRUB recognizes Linux partitions, so the Disk On Chip holds only a single Linux partition. The kernel is located on the Disk On Chip under */boot* and GRUB's configuration files are located in */boot/grub*.

In the past, Disk On Chip and compact flash cards loaded Linux by first booting into DOS, then invoking loadlin.exe to boot the Linux kernel. This is no longer true. DOS is completely removed from the picture, and Linux is natively booted from the device.

1.2 TSLinux

Technologic Systems currently uses an internally developed flavor of Linux called TSLinux. Previously, a PeeWee Linux based distribution was used, and after many struggles of its use and its use of an antiquated kernel and OS, a decision was made to develop a new GNU/Linux operating system, TSLinux.

TSLinux was developed from “Linux From Scratch” and Mandrake. Many of the configuration scripts and administration files are similar to Redhat and Mandrake. It is also an unsecured distribution. While add-on tarballs of ssh and other utilities are available, securing the distribution to adequately handle the security needs one may have is entirely up to the customer. Although TSLinux is fairly generic, it is custom made to be used on a Technologic System's Single Board Computer, and is unsupported in any other use.

TSLinux is an open source project based on GPL and GPL like licensed applications. Keeping with the open source culture, development is an ever-going process and community support is appreciated. Feedback to this document, TSLinux, and the use of Linux (regardless of the distribution used) on the Technologic Systems SBC are welcomed and appreciated.

2 Configuration

The default password for a root login on TSLinux is *redhat*. By default, logins are always enabled on COM2. The default terminal for kernel boot messages is also on COM2. If you are using a VGA daughter board and fail to see any kernel boot messages or a login prompt, then try logging in via COM2 before assuming the kernel failed to boot.

The configuration files of TSLinux are similar to those of Mandrake. All configuration files are located in the /etc directory, except for the apache configuration file. Apache's configuration file is located at */var/www/conf/httpd.conf*. The apache web-site (www.apache.org) has plenty of documentation on its configuration. O'Reilly, Sams, and Wrox publish excellent books about Apache that can be purchased at a local bookstore or on line. Section 6.1 lists some books that have been found helpful.

All on-board, wired LAN networking settings are stored in files located at */etc/sysconfig*. Each network interface is found in a file named 'ifcfg-' followed by the interface name. To change the IP address of the on-board network interface, the file *ifcfg-eth0* would be edited. The default IP address of eth0 is the Class C network address of 192.168.0.50. The meaning of each line in the ifcfg-* file and its appropriate values can be found by checking Redhat's web-site on how to configure a network interface in Redhat.

The TS-5500 and other SBCs with a PCMCIA slot support a wireless network interface. As is typical with any system that uses pcmcia_cs scripts, the default network settings for the pcmcia device is not in */etc/sysconfig*, rather, it is in */etc/pcmcia/network.opts*.

The installed ftp server is ProFTPd, a “highly configurable GPL-licensed FTP server software”. The configuration file for it is */etc/proftpd.conf*, and is structured similar to Apache's configuration file. ProFTPd as is installed in TSLinux, is a standalone server, though it would be trivial to set it up to use inetd. The default configuration denies all access except for anonymous logins, allows downloads only in the */var/ftp/downloads* directory, and allows uploads only to the */var/ftp/uploads* directory. The ProFTPd website (www.proftpd.net) contains plenty of documentation for those who wish to customize the default configuration file.

Other basic system configuration is done similar to that of Mandrake. For example, to change the default baud-rate for getty logins of the serial port, or to enable/disable logins via VGA terminals, one must edit the file */etc/inittab*. The run level scheme is similar to any System V system, such as Mandrake.

Boot-up scripts should be placed in */etc/init.d* and symlinks to those scripts should exists in the appropriate run level directory under */etc/rc.d*. The default run-level is run-level 3.

3 Development

3.1 Needed Tools

The purchase of a developer's kit for the SBC is recommended. Included in the developer's kit for those boards that support Compact Flash (CF), is a CF to USB dongle. Available for purchase is a compact flash PCMCIA adapter, which has faster read/write speeds than the USB dongle. Even if one is planning on using Disk On Chip, using CF cards during development is high recommended. CF cards are easier to work with and simplifies the storage and transfer of files to and from the SBC.

The PCMCIA CF card is seen by a desktop's kernel as a regular hard disk when installed in the PC's PCMCIA slot. The CF card is typically seen as /dev/hde. The typical partition scheme is used, thus the first partition is /dev/hde1 and the second partition is /dev/hde2. The USB dongle, however, emulates the CF card as a SCSI device. Thus, the CF card is seen as the next available SCSI device, which is typically the first on those systems without a SCSI or SCSI emulated device. The CF card in this situation would be /dev/sda, with the first partition seen as /dev/sda1. Mounting the CF card is done the same way as mounting any other device within Linux. The following command, ran as root, will suffice.

```
Mount /dev/sda1  /mnt
```

3.2 Compact Flash Imaging and Restoration

Backup of a compact flash card is done by imaging it, often referred to as “ghosting”. Restoring ghosted images of the compact flash card to a saved state is done with the dd command. 'dd' is a standard Linux tool installed on most desktop distributions. To back up the CF card, the followed command is used:

```
dd if=/dev/sda of=my_backup_file
```

'if' is the input file, and 'of' is the output file.

To restore the CF card, the compact flash disk is specified as the output:

```
dd if=my_backup_file  of=/dev/sda
```

Restoring a CF cad to its default shipping state is a three step process. First, one must download these three files from the Technologic Systems' TSLinux Kernel web-page:

1) A blank CF image for the size of the target CF card. This image sets up the 2MB FAT12 partition used by Syslinux, the installation of Syslinux in the Master Boot Record, with the rest of the card devoted to Linux. (e.g. TS-base-32.cf)
2) A tarball for the desired version of TSLinux to be installed (e.g. OS-3.05.tar.gz).
3) A tarball for the sample kernel for the target Single Board Computer.

Then, using the dd utility, apply the blank CF image for the target CF card. The TSLinux Kernel page hosts CF images for various SanDisk CF cards. These images are only guaranteed to work on a SanDisk compact flash card of a size that matches the file. When the dd command has finished, remove the compact flash card from the adapter and reinsert it to ensure that the partition tables are re-read correctly. After mounting the card's second partition, the next step is to place the OS onto the CF card. The following command demonstrates this (assuming the second partition of the compact flash card is mounted to /mnt/cfp2):

```
tar -C /mnt/cfp2 -xvzf OS-3.05.tar.gz
```

If you have a PCMCIA embedded SBC, a symbolic link must be made from */etc/init.d/pcmcia* to an appropriate run level, typically run-level three. This can be accomplished by

```
ln -s /etc/init.d/pcmcia /etc/rc.d/rc3.d/S30pcmcia
```

Once this has completed, the compact flash card is ready for the next step, placing the board-specific kernel onto the compact flash card. Unpack the tarball, which will spit out two directories, */lib* and */boot*. The */boot* directory contains kernel specific files, one of which is the bzImage-<kernel-name> file. This file should be copied to the Compact Flash card's first partition as bzImage. The second directory, */lib*, should be recursively copied to the root of the compact flash card. This directory contains the kernel specific modules.

A detailed discussion of the kernel naming schemes and package formats are discussed later in this document.

3.3 Disk On Chip restoration

There is no easy way to restore Disk On Chips. Since Disk On Chips are now booted natively via GRUB, TSLinux now requires that the M-systems Disk On Chip firmware be kept in place. The recommended method to put TSLinux onto a Disk On Chip is to boot a SBC from a Compact Flash card, and handle the Disk On Chip within TSLinux. The following steps outline how to setup a Disk On Chip.

1. Fdisk and format (mke2fs) the DiskOnChip (/dev/fl/fla1/disc)
2. Put TSLinux OS onto the DiskOnChip.
3. Mount the DiskOnChip to the filesystem.
4. Place SBC specific kernel into the DiskOnChips's /boot directory.
5. Unpack the kernel modules into the DiskOnChip
6. Unpack the patched GRUB files onto the DiskOnChip (thereby populating /boot/grub)
7. Run GRUB and install GRUB onto the DiskOnChip's Master Boot Record.
8. Mount and edit the DOC's */etc/fstab*, changing */dev/hda2* to */dev/fla/fla1/part1*

3.4 Compiling binaries

The recommended method for programming and compiling binaries for a SBC is to do all development on a Linux Desktop, and copy or ftp over the compiled binary to the target board. Source code should be written in any tool that a developer is comfortable in, such as EMACS or Vim. Almost all major Linux desktop distributions include EMACS or Vim. Both are text editors, with features to simplify the life of programmers. Indeed, many claim either can be seen as an integrated Development Environment (IDE). The use of either editor, along with the gcc compiler, is the standard method of programming.

One problem developers run into is missing libraries. The complete set of unstripped glibc 2.2.5 libraries used by all TSLinux SBCs, can be found on the Technologic Systems website. The file is named entire.libs.dir.tar.bz2. Another common problem is mismatched library versions, between the developer's desktop and the target SBC. In these cases, the problem can be solved by explicitly linking against the unpacked entire.libs.dir.tar.bz2 tarball. The GCC manual details on how this can be accomplished. Alternatively, the use of a virtual TSLinux box can be used instead. The virtual TSLinux box is the same virtual box culled from to create TSLinux version 3.xx The virtual linux box can be either chrooted() into or booted into via User Mode Linux (http://user-mode-linux.sourceforge.net)

When compiling programs with the gcc compiler, the explicit use of the '-march=i386 -mcpu=i386' options should be used. If compiling a downloaded applicationk, set the CFLAGS environment variable

to include those options before running the application's configure and make scripts. This can be accomplished from a bash shell by typing

```
export CFLAGS='-march=i386 -mcpu=i386'
```

Many programs are compiled with optimization (with -O or -O3 passed as an option to gcc during compilation). By default, gcc will optimize to the running CPU. The compiler should be told to optimize for the 386, thus guaranteeing that executables will run on the SBC.

GNU has a complete manual for gcc on their website. There are many online tutorials available on the Internet, along with many published books about gcc and its use.

For those who prefer a GUI and full featured IDE, Metrowerks offers a possible solution. Metrowerks' Code Warrior for Linux can be purchased from their website. See Metrowerks' website for more information (www.metrowerks.com)

3.5 Customizing the file-system for production use

The default shipping of TSLinux is tailored for developers. The file-system boots up read-write on top of an ext3 file-system.

The problem with ext3 is that it does not 100% guarantee that the file-system is uncorrupted after unclean shutdowns. In fact, no file-system, no matter how robust, can give a 100% guarantee that there will be no problems after a failure. After each improper shutdown, the file-system must be checked. The checking of the system can be a long process, and often times, needs to be manually intervened due to some error that can not be fixed on its own. Obviously, this is not good in an embedded application.

However, one can obtain such a guarantee with Linux. The solution is the same as it is with any OS. Don't write to disk. If the file-system is mounted read-only, an unexpected shutdown has no chance of corrupting the file-system. However, there are some applications that need to write out to disk, such as logs, process id numbers, or other various bits of information that needs to be stored onto a 'scratch-pad' area. The two most common locations for such information is typically in */var* and */tmp*. For instance, ProFTPd needs to write out its process ID to disk when used in daemon mode.

The solution to keeping the file-system read-only and having a 'scratch-pad' area of disk, is to use a virtual ram-disk for these areas. The following steps detail how to convert the TSLinux OS shipped by Technologic Systems into a system that can withstand the abuse of an embedded environment where typing *shutdown -h now* is not an option. All modifications to the files should be done from your desktop, and not from within the running Single Board Computer.

First, a few files must be edited in order to put the file-system into read-write mode.

Remove */etc/mtab*, and make mtab a symbolic link to */proc/mounts*

```
rm etc/mtab
ln -s /proc/mounts etc/mtab
```

edit etc/rc.d/rcS.sysinit to NOT remount / (root) partition read-write.
The following line

```
mount -n -o remount,rw /
```

should be commented out like so

```
#mount -n -o remount,rw /
```

Now two small virtual read-write ramdisk partitions will be setup by adding the following to etc/fstab:

```
tmp_log /var/log tmpfs size=512K 0 0
tmp_var /var/run tmpfs size=100K 0 0
```

```
#tmp_tmp /tmp tmpfs size=100K 0 0
```

tmpfs, the file-system used for virtual ramdisk, has an advantage of shrinking and growing to accommodate the needs of the mounted partition. We specify an upper limit to the size it can grow to, otherwise, it would grow to about half of the available memory.

By default, Apache places its logs in */var/www/logs*. We can specify it to place its process id and its logs to the virtual disk by removing */var/www/logs* directory and sym-linking it to /var/log

```
rm /var/www/log
ln -s /var/log /var/www/log
```

4 Device Drivers

The following sections detail how to use the various device drivers that are custom to the Single Board Computer. Both the Analog to Digital driver and the Watch Dog Timer drivers refer to non-standard header files that are only found in a TSLinux kernel source tree. When writing user-space code that makes use of either driver, one can either compile against a TS-specific kernel, or glean the needed header files from the Technologic Systems kernel patch.

4.1 SBC info

Basic information about the SBC, such as model number and options installed, can be read from the */pro/SBC/info*. This proc entry is read only and exists only with patched kernels that are built with a chosen processor type for a Technologic Systems Single Board Computer.

4.2 Accessing DIO

The DIO lines from the SBC can all be accessed through I/O space. Using the DIO lines requires no device driver, and can be done entirely from your own application. Accessing I/O space does require root privileges, requiring applications that do so to be either run by super-user root. This can also be accomplished by setting the binaries file permissions sticky bit to execute with super-user privileges. Obviously, this weakens the overall security of the system, so care must be taken when doing anything with super-user privileges.

First, the application must ask the kernel for permission to manipulate the desired I/O locations. A failure will typically mean that another program or device driver is utilizing the requested I/O space. Having gained permission, the application has control over the requested I/O space. Attempting to access any I/O without being granted permission to do so will result in a segmentation fault. inb(), outb() and friends are used to read and write to I/O. All I/O out functions reverse the argument order used by Turbo C's outb(). The following demonstrates how to access the TS-5300's DIO1 from Linux.

```
#include <stdio.h>
#include <sys/io.h>
int main() {
   if (ioperm(0x7A,4,1)<0){
        printf("Error\n");
        return -1;
   }
   //set pins0 to 3 to inputs
   ioval=inb(0x7a);
   ioval= ioval & 0xFE;
   outb(ioval, 0x7A);
```

```
    return 0;
}
```

For more information about DIO, its use, and I/O locations, refer to the SBC's manual. For more information regarding the outb() family of system calls, see the man page for outb or inb.

4.3 Using the Watch Dog Timer

Technologic Systems offers Linux device drivers for utilizing the SBC's on-board Watch Dog Timer (WDT). The WDT on AMD ElanSC520 based SBCs is accessed through */dev/amdwdt*, and on SBCs with an Intel 386EX processor, through */dev/tswdt*. The WDT is configured and started through an ioctl call. Servicing the WDT is done with a write to the WDT device. Once started, the WDT can not be stopped. Further information about the WDT, such as appropriate time-out values, can be found in the SBC's manual.

The following is a quick example code for using the watchdog timer.

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include "AmdElanWDT.h"

short Mode ;
int main (void)
{
        unsigned int timeout_value = 0x10;
        char *devname = "/dev/amdwdt" ;
        int fd = open (devname, O_RDWR);
        if (fd < 0) {
            printf("AMDSC520 Watch Dog Timer did not open\n");
            printf ("Errno: %s \n", strerror(errno)    );
            return -1;
        }

        ioctl ( fd, WDT_BEGIN, timeout_value);
        while ( 1) { write (fd, "foo", 3); }
        return -1;
}
```

4.4 Using the Analog to Digital Converter

Some SBCs offer an Analog to Digital Converter (A/D). Typically, the Maxim 197 chip is used to do the conversion. This ICU offers 8 channels and 2 ranges. The SBC's manual has detailed information on the possible ranges, resolutions, and how to interpret the values read back from the A/D device.

Technologic Systems offers a device driver to independently configure, read, and write each channel. This generic A/D driver is named a2d_driver.o The device is accessed under */dev/AtoD/0* through */dev/AtoD/7*, with each number representing a separate channel. An ioctl() call invoked on a channel configures that channel. A write to that channel initiates a conversion, and the converted value can be

obtained with a read command. The following c code demonstrates the use of channel 1. Note the inclusion of the file a2d_driver.h, which is found only in a kernel source tree patched with Technologic System's kernel patch.

The TS-9700 and TS-5600 do not make use of the generic A/D driver. The TS-9700 has its own driver for both A/D and D/A conversions. This module is named ts9700.o. It exports device nodes under */dev/SBC/TS9700*.

The TS-5600 has its own driver code to access the A/D and D/A devices connected to the Elan SC520's SPI bus. The kernel code is not a loadable module, it is statically compiled into the kernel core itself. When building your own kernel for a TS-5600, choose a processor of 486 with the Technologic Systems add-ons enabled. The TS-5600 exports its device nodes under */dev/SBC/TS5600*.

All three drivers share the same generic interface: Open the device node, write() to A/D channels to initiate a conversion, then do a read() call to obtain the results. D/A device nodes are write-only. The following is example code against the generic a2_driver.o device driver.

```
#include <stdio.h>
#include <unisted.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#include <a2d_driver.h>

int main () {
    usigned char command = 0;
    int response;
    signed int datavalue;
    int fd= open ("/dev/AtoD/0", O_RDWR);
         //make sure that this device
         //has been created.
    if (fd <0) {
         printf("failed opening /dev/AtoD/0\n");
         return -1;
    }
    command = (A2D_UNIPOLAR| A2D_RANGE1);

    ioctl( fd, TSA2D_CONFIG, command);
    response = ioctl(fd, TSA2D_SHOWCONFIG);
    printf("/dev/AtoD/0 is %spolar, and uses Range %d\n",
         response & A2D_BIPOLAR ? "bi" :"uni",
         response & A2D_RANGE2 ? 2: 1);
    //test write operation
    datavalue = 0;
    response = write (fd, "trigger", sizeof("trigger");/*string
"trigger" is arbitrary. Any string will do*/
    if (response < 0 ){
         printf(" write returned %d\n", response);
         perror("error:");
    }
    response = read (fd, &datavalue, sizeof(signed int));
    printf("Channel 0: 0x%x\n", datavalue);
    /* do another conversion*/
    write (fd, "trigger", sizeof("trigger"));
    response = read(fd, &datavalue, sizeof(signed int));
    close (fd);
    return 0;
}
```

4.5 RS485

The “Serial Programming HOWTO” (www.linuxdocs.org/HOWTOS/Serial-Programming-HOWTO/) is an excellent guide to those who are unfamiliar with programming serial ports from within a POSIX compliant system, such as Linux. However, the HOWTO fails to demonstrate how to toggle the RTS line. Also, timing the toggling in a non-realtime, non-preemptible kernel such as Linux can be frustrating. Starting with TS-5300 rev C, the TS-5400, and TS-5500, the toggling of the RTS line for RS485 can be done automatically. The SBC's manual can reveal if the board is capable of this feature. An extra ioctl() command is added to the kernel to enable automatic RS485 mode and RTS mode. The board defaults with RTS mode turned off, which translates to any attempts to write data out the RS485 lines being ignored. If an application wishes to not use automatic RS485 mode but still utilize the RS485 chip, then RTS mode must be set. Applications must be built against kernel headers from Technologic System's kernel patches. The following snippet of code demonstrates both setting the RTS mode and setting automatic RS485 mode.

```
#define RTSMODE 2
#define AUTO485   1
```

<snip>

```
mcr = AUTO485;
ioctl(fd, TIOC_SBCS485, &mcr);/*use TIOC_SBCC485 to clear */
mcr = RTSMODE;
ioctl(fd, TIOC_SBCS485, &mcr);
ioctl(fd, TIOC_SBCC485, &mcr); /*disable RS485 */
```

4.6 QVGA FrameBuffer

Technologic Systems offers Quarter VGA daughter boards. Each QVGA SBC has its own device driver. However, they are all typical frame buffer drivers. This means that they support generic ioctl() system calls used by all Linux frame buffer drivers. They also support the mmap() system call, so that one may write directly to the video device's video memory. Any windowing application that does not use X11, but accesses the Linux frame-buffer directly, will work. Examples of powerful window management tools, would include QT/Embedded from Troll Tech, picoGUI, and microGUI.

4.7 Matrix Keypad

Many of the Single Board Computers offered by Technologic Systems can support a matrix keypad, also sold by Technologic Systems. The kernel module for the matrix keypad is ts_keypad.o, and exports the device node */dev/SBC/TSKeypad*. When the device node is opened, the driver begins its keypad scan routine, storing key-presses into a buffer. When the device node is read from, if any key-presses were made, they are sent back to the user. The following example code demonstrates the use of the matrix keypad:

```
 /*
  * readkeypad.c
  * (c) Technologic System's (info@embeddedx86.com)
  *      Liberty Young (liberty _AT__ embeddedx86 /DOT/com)
  *
  * Technologic Systems makes no warranties with this code;
  * it is provided AS-IS. This code was developed for use in
  * a Technologic Systems' Single Board Computer of the
  * 3x00 and 5x00 series.
  *
  * Technologic Systems' sample application code
  * This sample user application makes use of the
  * ts_keypad.o device driver
  * It simply reads from the keypad, printing out the results,
  * until the Button '#' is pressed .
  */
 #include <fcntl.h>
 #include <stdio.h>
 #include <unistd.h>
 #include <errno.h>
 #include <string.h>
 #include <sys/time.h>

 int main (void)
 {
    char buf[0];
    int rtrn = 0;
    buf[0] = 0;
    char *devname = "/dev/SBC/TSKeypad" ;
    int requestedlength;
    struct timeval tv1, tv2;
    struct timezone ignore;

    int fd = open (devname, O_RDWR);
    requestedlength = sizeof(char);

    printf("Starting Technologic Systems' Matrix Keypad Example
Program\n");

    if (fd < 0) {
         printf("the TS MatrixKeypad device can not be found..is
the driver loaded?\n");
         printf("error val: %d ", fd);
         printf ("Errno: %s \n", strerror(errno)    );
         return errno;
    }

    printf("testing continous read of device\n");
```

```
    printf("press '#' to end this stage\n");
    while (buf[0] != '#') {
         rtrn = read (fd, &buf, sizeof(char));
         if (rtrn < 0 )
              printf ("Errno: %s \n", strerror(errno));

         if (!rtrn) {
              buf[0] = '\0';
              continue;
         }
         printf("Found Key: %c\n", buf[0]);
    }

    printf("Now we'll test the 'buffered' feature of the
driver\n");

    //pseudo sleep for 3 second
    gettimeofday(&tv1, &ignore);

    printf("Remeber, the driver will recognize any and all
keypress\n");
    printf("As long as the device file is in OPEN\n");

     do {
        gettimeofday(&tv2, &ignore);
     }while ( tv2.tv_sec  < (3+tv1.tv_sec));

    rtrn = read (fd, &buf, sizeof(char)) ;

    while (rtrn > 0)  {
         printf("Found Key: %c\n", buf[0]);
         rtrn = read (fd, &buf, sizeof(char)) ;
    }

    printf("goodbye!");
    return 0;
  }
```

4.8 Disk On Chip Drivers

Disk On Chip drivers are supplied by M-systems. While we endorse the official drivers supplied by M-systems, we do not distribute their patches. Information, as well as downloads of, Disk On Chip driver patches for a Linux kernel are available from the M-systems website. However, Technologic Systems precompiled kernels do contain the doc.o driver in module form. If a Disk On Chip is installed on the target SBC, doc.o will successfully load. The Disk On Chip can be accessed through /dev/fl/fla1

5 Kernel Development

5.1 TSKernel

While an un-patched Linux Kernel can be used on a Single Board Computer, Technologic Systems has developed a patches for the Linux Kernel. These patches increase performance and add SBC specific device drivers.

TSKernel is the official name for the line of kernel patches from Technologic Systems. TSKernel patches standalone, and do not stack with other TSKernel patches. TSKernels with a version of 2.xx are all built against the 2.4 line of Linux kernels. The TSKernel patches add features and driver support for the various products of Technologic Systems. Patches are available on the Technologic Systems website, as well as pre-compiled binary packages for TSKernels.

The TSKernel has its own naming scheme for pre-compiled binaries. They are used to differentiate kernels built for a specific line of SBCs.

- Parthenope
  kernels that have the name of Parthenope are built specifically for the 3x00 SBC series.
- Ligeia
  kernels that have the name of Ligeia are built specifically for the 5x00 SBC series without PCMCIA support
- Leucosia
  kernels that have the name of Leucosia are built specifically for the 5x00 SBC series with PCMCIA support. PCMCIA modules are found in a separate file.

Each TSKernel version has three major parts. The first part is the actual kernel patch itself. The second major part to a TSKernel is the pre-built kernel binaries. The third major part is the kernel header files. Each part is downloaded separately from the other, but as a whole, makes up a TSKernel distribution.

The kernel patch is generated in the typical context diff style against a clean, vanilla copy of a Linux kernel. The kernel patches include saved .config files. They are placed in the top directory of the Linux kernel and when loaded, will configure a kernel exactly like the pre-built kernels of the same version.

The pre-built kernels are available in three packaged formats. A gzipped tarball, an RPM, and a Debian package .deb. The tarball and RPM are typically created from a Debian .deb with the tool 'alien'. The tarball, being the universal format, is one used for discussions in this paper. The tarball, when unpacked, will generate a */boot*, */lib*, and */user* directory. The */boot* directory contains a saved System.map file as well as a bzImage file (the kernel itself). The bzImage is saved in as **bzImage-<kernel-version>-ts.<build-name>**. The kernel should be moved from that location to the Compact Flash card's first partition with the name of bzImage, dropping the extraneous letters in the kernel name. The */lib* directory contains the kernel modules, and is copied recursively to the root of the second partition of the Compact Flash card. The */usr* directory contains nothing of importance.

The kernel header files are also available in three package formats (.tar.gz, RPM, and Debian .deb). These files simply contain the kernel header files. Unpacking the tarball will generate /usr/src/kernel-headers-<kernel version name>/include. This is quite useful for developers who need kernel header files that are specific to Technologic System's drivers, without forcing them through the process of patching a kernel simply to generate those header files. These should not be placed onto the Compact Flash card or Disk On Chip. They are only useful for developers and should be placed on the developer's workstation.

5.2 Building the TS kernel

It is advised that all kernel development work not be done as super-user. The book Linux Device Drivers is highly recommended for those who are new to the subject or with kernel development in general.

Compiling your own kernel is not a complicated task; at the same time, it is not a trivially easy task. Visiting such sites as www.kernelnewbies.org, and reading the first two chapters of Linux Device Drivers should be done first. A basic understanding to compiling applications under Linux is also needed. This would include familiarity with the patch command. The following guide-line for installing a pre-built kernel assumes one has a Compact Flash card (or DiskOnChip) already partitioned and has a boot-loader installed.

Download the appropriate version of the Linux kernel that the TSKernel patch is for, which should already be downloaded from our site and saved locally to your machine. Kernels can be downloaded from www.kernel.org. Unpack that kernel into a local directory. Apply the TSKernel patch against the unpacked kernel. If one needs DiskOnChip, then download the DiskOnChip Linux drivers from M-systems and apply them to the locally saved Linux kernel. The next step is to configure the kernel. TSKernel patches also generate saved .config files that can be loaded from the kernel configure menu. By doing so, one will start off with a kernel configured exactly like the pre-built kernel binaries on the Technologic Systems web-site. After saving the kernel configuration, simply go through the build process and install the kernel modules to a compact flash card. Then copy the *arch/i386/boot/bzImage* file to the compact flash card in a location expected by the boot-loader.

5.3 Installing pre-built TSKernels

After one has pre-partitioned the Compact Flash card into two partitions (fat16 and ext3), loaded the boot-loader and the TSLinux Operating System, one is ready to install a kernel binary package onto the Compact Flash card. Download the kernel binary for the correct SBC (see section 5.1 for which code-name is right for one's SBC). It should be unpacked, resulting in three directories. Recursively copy the */lib* to the / (root) partition of the Compact Flash card. Copy the bzImage file found in the */boot* directory to the DOS partition of the Compact Flash card.

Kernel binaries that support PCMCIA have an extra tarball to download, the pcmcia-cs modules. The pcmcia-cs kernel modules are packaged separately. Without the pcmcia-cs modules, the */lib/modules/*<kernel-name>*/pcmcia* directory will be empty. The pcmcia-cs tarball is named pcmcia-modules-<kernel version>-ts.leucosia-<pcmcia-cs version>.tar.gz. When unpacked, it will generate a */lib* directory. Just like the other kernel binary packages, it must be copied recursively to the / (root) directory of the operating system.

6 Recommended Reading

6.1 Books

Rubini, Allesandro and Jonathan Corbet. Linux Device Drivers 2nd edition
Oreilly, June 2001.
Bovet, Daniel P and Marco Cesati. Understanding the Linux Kernel 2nd edition

Oreilly, December 2002.
Frish, Aeleen. Essentail System Administration 3rd edition
Oreilly, August 2002
Hallobaugh, Craig Ph.D. Embedded Linux Hardware, Software, and Interfaceing
Addison Wesley, November 2002
Welsh, Matt and Matthias Kalle Dalheimer and Terry Dawson and Lar Kaufman. Running Linux 4th edition
Oreilly, December 2002.

6.2 Technologic Systems Documents

Getting Started with TS-Linux Version 3.0

Target SBC's Developers Manual.

6.3 Websites

www.embeddedx86.com
www.kernelnewbies.org
www.google.com
www.linuxdocs.org/HOWTOS/Serial-Programming-HOWTO/
www.linuxdocs.org
www.m-systems.com
www.embeddedx86.com/~mike
www.embeddedx86.com/~grimiore

7 Revision History

Date	*Revision Notes*
2002.12.30	Initial Release
2003.5.15	Plethora of new information added
2004.2.19	added driver sections (QVGA, keypad), new kernel install procedures, and added to Section 3.4