Martyn Welch

Running and Programming Linux

Introduction

Linux shares very little in common with Microsoft Windows, the operating system that the vast majority of people have been taught to use or program. This report aims to provide the background information required to begin programming under Linux, with an emphasis on interfacing; to point the reader in the direction of further help that is available; as well as describing some of the details of programming for this operating system. This paper expects the reader to already have a fair understanding of the principles of programming in the C language.

History

Linux is an open source[1] operating system that was first written by Linus Torvalds and has been developed and extended, by a mainly volunteer, community on the Internet. Formally Linux is only the operating systems kernel[2] the rest of the "system" draws heavily upon the work of the GNU project[4], run by the Free Software Foundation. The Linux kernel has been designed to comply with the POSIX[3] standards, which were originally written to try and standardise the diverging different brands of Unix[4]. Although a Linux system can not be considered a Unix[5], mainly since it has never been through the ratification process, the two systems share a large amount in common and is often fairly easy to port software between the systems.

The previously mentioned GNU project was founded a few years prior to Linux's conception by another very talented coder named Richard Stallman, who was becoming disillusioned by the "proprietary"[6] route which software industry was taking. The GNU project's aim is to write an entire operating system from scratch, placing it under the GNU's GPL License[6]. This license states that any software placed under it can not be charged for (however the distributer may charge for related services and may cover relevant distribution costs) and if distributed, must always be

---

1 Open Source - "A method and philosophy for software licensing and distribution designed to encourage use and improvement of software written by volunteers by ensuring that anyone can copy the source code and modify it freely." (from the "Free On-Line Dictionary Of Computing"[3]).
2 Hence forth this paper will use the term "Linux" to describe the entire GNU/Linux operating system and "the Linux kernel" when taking specifically about the kernel.
3 POSIX: Portable Operating System Interface developed by PASC[2].
4 Unix – See The Jargon File[1] (http://www.catb.org/~esr/jargon/html/U/Unix.html).
5 In fact the GNU acronym is a recursive acronym standing for "GNU's Not Unix".
6 Proprietary Software - "Software that is not free or semi-free. Its use, redistribution or modification is prohibited, or requires you to ask for permission, or is restricted so much that you effectively can't do it freely."[5] (from GNU "Categories of Free and Non-Free Software" - Please contrast with Free/Open source software and Commercial software).

accompanied by the source code or means of acquiring it (with the above restrictions on charges). By chance this software was also being developed to be compliant with the POSIX standards.

The Linux Model

Linux differs a lot when compared to the most commonly used OS's from Microsoft[7], however they share a strong similarity to the Unix model (and thus the inner workings of Mac OS X). The following sections will describe Linux's main features.

File System

The way that Linux deals with file system seems very odd to those who have derived most of there computer knowledge from DOS or it's derivative operating systems, but it is the standard method in the Unix world. Under DOS each drive associated with a letter and the files and directories on each device are accessible by moving the focus of the operating system between these letters. Rather than each drive being accessible via a letter of the alphabet, Linux uses a predefined drives' contents as a base or "root" directory[8] structure. The directories on this drive are usually laid out in a standard way (as defined by the File system Hierarchy Standard[7]), the system then allows other drives to be accessible from subdirectories or "mount points" under this structure.

This flexible system allows parts of a computers storage to be expanded as and when required by migrating parts of the system, previously part of the root system or small capacity drives, onto higher capacity drives or even utilise storage space on another computer over a network connection.

File Attributes

Linux is a multiuser operating system, it has been designed to be used by many people, even at the same time. Systems such as this require some way of determining who is allowed to do what and to allow the systems users to stop others reading private files and also to allow this same kind of control to be extended to subsets of the users on the system. Each file has three sets of access attributes and one set of special attributes. These can be seen by using the "`ls`"[9] command with the "`-l`" option, which lists each of the files and subdirectories in the current directory with its attributes, as shown in Text 1.

---

7   This is mainly due to Linux deriving it's model indirectly from Unix and Microsoft's operating systems deriving their model from DOS.

8   Usually denoted as "/".

9   ls – list directory contents (from ls man page), similar to dir in DOS.

```
martyn@whiterabbit:~/network$ ls -l
total 96
-rwxr-xr-x    1 martyn   users       12327 May 16 09:17 client1*
-rw-r--r--    1 martyn   users         694 May 16 09:17 client1.c
drwxr-xr-x    2 martyn   users        4096 Jun  5 17:22 perl/
-rwxr-xr-x    1 martyn   users       12400 May 16 09:17 server1*
-rw-r--r--    1 martyn   users         909 May 16 09:17 server1.c
-rwxr-xr-x    1 martyn   users       16239 May 16 16:42 tcpCheck*
-rw-r--r--    1 martyn   users        7399 May 16 16:42 tcpCheck.c
-rw-r--r--    1 martyn   users        1659 May 16 09:17 tcpClient.c
-rwxr-xr-x    1 martyn   users       16133 May 27 16:42 tcpServer*
-rw-r--r--    1 martyn   users        4596 Jun  4 11:14 tcpServer.c
martyn@whiterabbit:~/network$
```
Text 1 : "ls" Listing

At the start of the line is a set of 10 characters which look something like "`-rwxrwxrwx`". The first character is used to show any special attributes, in this case "`-`" shows that the file has no special attributes. In the case of the "`perl`" directory this character is show as a "`d`" which denotes that this item is in fact a directory. There are other options; an "`l`" denotes a symbolic link[10]; a "`b`" identifies the entry as a block device[11] and a "`c`", a character device[11] (other lesser used characters exist, which I won't go into here). The remaining 9 characters are split into 3 blocks (of "`rwx`"), each block shows the state of the 3 access attribute sets. Each of these blocks determine what a certain group of users on the can do with the file. The first block determines what the "owner" of the file can do with it; the second, anyone in the same "group" and lastly anyone else on the system. The first character, shown as "`r`" shows if the file or directory can be read, if not a "`-`" is shown instead. The "`w`" determines whether the file or directory can be written to and finally the "`x`" shows if the file can be executed. In the case of directories the "`x`" denotes whether the directory can be entered[12]. Unlike DOS, which uses special file extensions, such as "`.exe`" and "`.com`" to denote a program, Linux uses this "`x`" attribute to determine whether it can try and execute a file as a program. This gives the system added flexibility, a program can start off as a simple shell script[13] and if it proves useful maybe re-written at a later date into a compiled language such as C, greatly increasing it's efficiency.

---

10 A symbolic link is similar to a "shortcut" in windows, but at a much lower level.
11 Both the "b" and "c" options are reserved for device files, which will be covered later.
12 Note: it is possible that a directory may be allowed to be entered, but not read or written to.
13 Shell script – A file containing commands capable of being run by the shell, similar to batch files in DOS.

There is one exception to this rule, the superuser[14]. The superuser has special privileges that allow him (or her) access to all files, directories and resources. Access to the superuser account is often strictly controlled and for obviously good reasons, it's existence is to enable the system to be administrated.

Device Access

As defined by the File system Hierarchy Standard[7], a special directory called "/dev" is present at the root of the file system. There are a large number of special files in this directory which follow a special naming convention.

Like Unix, Linux aims to be a very flexible system and to achieve this it presents access to the devices which the system is comprised as these special files. All communication with the system are modeled as streams to and from these files. This makes writing data to a serial port, which are present in "/dev" as files beginning with "ttyS", can be simple as accessing and writing to a file. To achieve this a stream is opened to the file and data is written to it. In the case of a normal file, this information would then be stored onto the hard drive, in the case of accessing a serial port, the data would be interpreted and the state of the serial port affected accordingly.

In fact these "files" are not actual files, however they do appear that way to any programs on the system. As with most Unix systems the actual device drivers are a part of the kernel. When a device file is accessed "the kernel recognises the I/O[15] request and passes it a device driver, which performs some operation, such as reading data from a serial port, or sending data to a sound card."[8] As mentioned above, these "files" have some special attributes, they are either set up as a block device or character device. "A block device is usually a peripheral such as a hard drive: data is read and written to the device as entire blocks"[8], a block being made up of a set number of bytes which may be written to any point on the device. A character device on the other hand are "usually read or written sequentially and I/O may be done as single bytes"[8], this is the case with serial ports.

Running Programs

When a program is run, it is given a number of attributes, some of which are provided by the user who caused it to run, others are unique to the running instance of the program. A running program is known as a process, it is possible to find out what processes are running on a computer running

---

14 The superuser account is similar to an "administrator" under Windows.
15 I/O – Acronym for input/output usually related to software communicating with hardware.

Linux by running the "ps"[16] program as shown in Text 2. The "-ef" option shows all processes which are running on the machine and shows the full listing for each.

```
martyn@whiterabbit:/dev$ ps -ef
UID         PID  PPID  C STIME TTY          TIME CMD
root          1     0  0 Jun05 ?        00:00:04 init [3]
root          2     1  0 Jun05 ?        00:00:00 [keventd]
root          3     0  0 Jun05 ?        00:00:00 [ksoftirqd_CPU0]
root          4     0  0 Jun05 ?        00:00:00 [kswapd]
root          5     0  0 Jun05 ?        00:00:00 [bdflush]
root          6     0  0 Jun05 ?        00:00:00 [kupdated]
root          8     1  0 Jun05 ?        00:00:00 [kjournald]
root         34     1  0 Jun05 ?        00:00:00 [khubd]
root         63     1  0 Jun05 ?        00:00:00 /usr/sbin/syslogd
root         66     1  0 Jun05 ?        00:00:00 /usr/sbin/klogd -c 3 -x
root         75     1  0 Jun05 ?        00:00:00 /usr/sbin/crond -l10
root         79     1  0 Jun05 ?        00:00:00 gpm -m /dev/mouse -t ps2 -r 100
root         81     1  0 Jun05 tty1     00:00:00 /sbin/agetty 38400 tty1 linux
martyn       82     1  0 Jun05 tty2     00:00:00 -bash
root         83     1  0 Jun05 tty3     00:00:00 /sbin/agetty 38400 tty3 linux
root         84     1  0 Jun05 tty4     00:00:00 /sbin/agetty 38400 tty4 linux
root         85     1  0 Jun05 tty5     00:00:00 /sbin/agetty 38400 tty5 linux
root         86     1  0 Jun05 tty6     00:00:00 /sbin/agetty 38400 tty6 linux
martyn     1129    82  0 11:44 tty2     00:00:00 ps -ef
martyn@whiterabbit:/dev$
```
Text 2 : "ps" listing

The first item shows the user identification (or "UID") the process is running as, it can be seen in Text 2 that a large number of these are running as "root", the superuser. This is because these are system tasks (in fact those surrounded by square brackets are kernel level processes) which can be considered to be programs that are administrating the computer system.

The 5th column labeled as "TTY" shows the terminal[17] that the process is running on, a question mark means that the process is not linked to a terminal. 5 of the processes are shown as being instances of "/sbin/agetty" in the "CMD" column (the command that was run to create the process) and are each attached to a different terminal. These processes listen to the terminal waiting

---

16 ps – report process status (from ps man page), similar to task manager under Windows.

17 terminal : "<hardware> An electronic or electromechanical device for entering data into a computer or a communications system and displaying data received. Early terminals were called teletypes, later ones VDUs. Typically a terminal communicates with the computer via a serial line." (from the "Free On-Line Dictionary Of Computing"[3]), also short for "virtual terminals" which are accessible when sitting at the machine or over a network via services such as telnet or ssh (secure shell).

for a user, accept and verify a user name and password before allowing access to the computer. Upon verifying a user agetty (or similar program) passed control over to a shell[18] (such as bash as shown to be running on tty2) which runs with the "UID" of the user who logged in.

Each process that is created will receive a unique process identification (or "PID"), which allows the operating system to keep track of the system memory and any devices that it is using. Unless the process is turned into a "daemon"[19], a process will also linked to the process which caused it to run (known as its "parent") and will have the parent's PID as its Parent PID (or "PPID") and will be linked to the same terminal, as can be seen from the "ps -ef" process that was launched to retrieve the list of processes. If a parent exits all of its children will also be force to exit (thusly freeing up the terminal for someone else).

Pipes and Redirection

One of the main philosophies used to build programs for Unix was that each one should be able to be considered as a simple tool, that is fast and efficient at one task. In order to obtain complex functionality the shell has an advanced system for redirecting the output from one program into another and thus allowing the user to effectively do what he or she wants. A simple form of this functionality can be seen in DOS, where the output from a program can be redirected to a file, using the ">" command. Linux, like Unix, has a fairly, fully featured redirection facility. As with DOS, the ">" symbol can be used to redirect the output of a program to a file, however under Linux this will only result in the standard output from the program being recorded in the file. It is good practice to use at least two different output streams in Linux programs, the standard output and standard error steams. The first can be directed to a file as shown above, the second, usually reserved for error messages as the name suggests, can be redirected by using "2>" to send the output to a different file, or ">&" can be used to send both streams to the same file.

This is not the only form of redirection that may be used, Linux allows the output of 1 program to be sent directly as an input for another process, though the use of "pipes". If it was required to find out what the disk usage of the files in the directory used in the "ls" example, Text 1, the result could be worked out by looking at the file sizes using "ls". However a better utility, exists for this, "du", but it doesn't list the contents by size order. This could be achieve using the above redirection utilising the following commands:

---

18 shell : "The command interpreter used to pass commands to an operating system; so called because it is the part of the operating system that interfaces with the outside world." (from the "Free On-Line Dictionary Of Computing"[3]), similar to the Dos "prompt", but more versatile.

19 daemon : A process which runs in the background, usually waiting for specific conditions to occur. These processes are not attached to any specific terminal and run independent of any logged in users.

```
du -s * > du.out
sort -nr du.out
```

This however is not a very elegant solution as it requires the creation of an intermediate file to hold the output from the "du" command. This could instead be achieved but using a pipe "|" as showing in Text 3.

```
martyn@whiterabbit:~/network$ du * | sort -nr
16      tcpServer
16      tcpCheck
16      server1
16      client1
12      perl
8       tcpServer.c
8       tcpCheck.c
4       tcpClient.c
4       server1.c
4       client1.c
martyn@whiterabbit:~/network$
```
Text 3 : Pipe example

Using the pipe simplifies the process and doesn't result in a temporary file, which would need to then be removed at a later date.

Linux Socket Layer

The sockets layer was first seen in the Berkeley versions of Unix[20]. Sockets are used for client/server style of communications between programs. The model allows for two different types of sockets, local and network sockets. Socket communications differ substantially from pipes, pipes only generally cause the standard output of one program to be sent to another as it's standard input, the server/client aspect of sockets allows multiple clients to interact with the same server.

Local sockets exist as a special file, that is created by the server program, which then listens to this socket for a connection. In the case of network sockets, instead of a special file the server listens on a "port" or access point[21]. In the case of TCP/IP communications, there are 65536 virtual ports available, however special privileges are required to access those below 1024 and are generally reserved for certain services[22].

Once the server is listening the server socket, it waits for a client to try and connect. When a client

---

20 The socket layer was in fact designed as an extension of the pipes concept.
21 The term used and the exact syntax is highly dependent on the networking protocol used, for the case of this report TCP/IP will be considered.
22 For example, port 22 is used for secure shell.

connects a second temporary socket is initialised and communication between the connecting client and the server is moved to that socket. This frees up the base socket for another connection from another client.

Help

One of the main points that is held against open source development is that, since many of the developers are writing the programs for free and for fun, they are relaxed about producing documentation, often thought of as a dull chore. Though this may be the case for a number of small projects, the larger project generally see these chores as necessary and usually produce one or more standard sets of documentation and some may have been written by happy users of the software, that do not have the necessary skills to help in the development of code.

man & info pages and other on-line help

Most Unix systems have an on-line help[23] in the form of manual pages, known as "man"[24] pages after the command that is used to access them. These are usually small documents that, among other things, give a short description of a certain command, any options that may be passed to it (and what they do) and a list of other related man pages. It can often be quite difficult to find a man page if you are unsure about what the command is that you seek to fulfill the a required task. Programs such as "apropos"[25] help by allowing users to search the man pages for specified words.

Though manual pages are the classical form of system based help for Unix systems (and were thusly reproduced for Linux), a more modern form of help known as "info"[26] has been developed. This system is like the manual pages, as it works though a terminal, however it includes hyperlinks to allow easy traversing between pages (something which is missing from man). This system was devised by the GNU project[4] and has obtained only mild adoption, although it does provide a rich source of information on their software. Info pages generally provide a greater depth of information usually including examples of how to complete common tasks with the program.

Both of these sources have be devised to work primarily through the console, since a large amount of this software runs in the absence of a GUI[27]. Graphical inclined programs that utilise a GUI

---

23 Note : In this case "on-line" does not refer to on the Internet, but stored digitally on the computer itself.
24 man - "format and display the on-line manual pages" (from man man page).
25 apropos - "search the whatis database for strings" (from apropos man page). The whatis database is created from the installed man pages.
26 info - "read Info documents" (from info man page).
27 Graphical User Interface - "<operating system> (GUI) The use of pictures rather than just words to represent the input and output of a program. A program with a GUI runs under some windowing system (e.g. The X Window System, MacOS, Microsoft Windows, Acorn RISC OS, NEXTSTEP). The program displays certain icons, buttons, dialogue boxes, etc. in its windows on the screen and the user controls it mainly by moving a pointer on the screen

generally provide much better documentation through a GUI and therefore usually provide help though there own interface or a help system, which is usually a part of the GUI used. This is frequently implemented as a HTML[28] based help system.

Internet-based Help

Apart from these general forms of help there are numerous other sources. There are obviously a who host of books covering various subjects (many good examples are published by O'Reilly & Associates[9]). There are many other sources of information, which can be more specific, in the form of mailing lists, newsgroups, "FAQ's" and "howtos".

Due to the feeling of common gain that is present in the open source community it is generally easy to find information in the archives of mailing lists[29] or newsgroups[30] specific to any problem with any piece of software. If the archives fail there is always the option to join the mailing list and submit a question, which will usually be answered by multiple people within a day or so[31]!

Many projects maintain a list of FAQ's (Frequently Asked Questions) on there web-sites, it is considered polite to at least try to establish whether one exists and to have a quick look through it if it does prior to submitting a query, as asking a question that has been answered numerous times can lead to being "flamed"[32].

There are a large number of howtos available on the web, they are "detailed "how to" documents on specific subjects."[11] They cover varied topics from "How do I set up X piece of software to do Y" to interfacing a Linux machine to a coffee maker[12].

Programming

There are numerous utilities available on Linux to aid programmers, including a number of fairly fully featured IDE's[33], some of which are proprietary others which are open source. Linux

---

(typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse while the pointer is pointing at them." (from the "Free On-Line Dictionary Of Computing"[3]).

28 HyperText Markup Language - "A hypertext document format used on the World-Wide Web. HTML is built on top of SGML. "Tags" are embedded in the text. A tag consists of a "<", a "directive" (case insensitive), zero or more parameters and a ">". Matched pairs of directives, like "<TITLE>" and "</TITLE>" are used to delimit text which is to appear in a special place or style." (from the "Free On-Line Dictionary Of Computing"[3]).

29 Information concerning the whereabouts of a projects mailing list[s] and archive can generally be found on the project web-site.

30 A good web-front end to many news groups is google groups[10].

31 Note: Of course it is considered polite to try and answer any queries which may have been posed by other people, that you feel in a position to answer.

32 Flame - "To rant, to speak or write incessantly and/or rabidly on some relatively uninteresting subject or with a patently ridiculous attitude or with hostility toward a particular person or group of people". (from the "Free On-Line Dictionary Of Computing"[3]).

33 Integrated Development Environment - "A system for supporting the process of writing software. Such a system may include a syntax-directed editor, graphical tools for program entry, and integrated support for compiling and

distributions generally come with the gcc[13] compiler, which is very capable of compiling many languages, from Fortran to C++, and to a whole host of different platforms.

In this section of the report I will be detailing the code required, in order to utilise certain features, mainly the control of certain items of hardware, under Linux, with the C programming language.

Networking with TCP

Networking under Linux is achieved though the sockets layer as described above. There are a number of different protocols that can be used under Linux to actually send information, this report will deal with TCP/IP. TCP/IP is a very versatile networking standard, it is the protocol which is used on the Internet and has a very strong implementation on Linux.

It is notable that when programming network communications there is no mention to what hardware is used. This is because the sockets act as a layer of abstraction, which allows the programmer to concentrate on writing the application rather than sorting out the hardware. It is the job of the kernel and the supporting daemons to take care of the actual hardware used to transmit [34].

There are a fair number of steps required to setup networking. This differs for the client and server portions. There are numerous examples that can be found both on the Internet and in books[14].

Text 4 and Text 5 which follow show how to setup a basic client and server program respectively.

---

running the program and relating compilation errors back to the source." (from the "Free On-Line Dictionary Of Computing"[3]).

34 Note: It is entirely possible for the computer to have multiple network cards installed to different networks of computers and the operating system will work out which card is the correct one to send the information out on.

```
// Include standard library header files
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <unistd.h>


// Main Function
int main (int argc, char *argv[]){

        int sock_desc;          // Setup variables for socket descriptor
        struct sockaddr_in Server_Sock_Addr;  // Create Server Address Structure

        sock_desc = socket(AF_INET, SOCK_STREAM, 0); // Create Socket

        // Setup Server Address Structure
        Server_Sock_Addr.sin_family = AF_INET;
        Server_Sock_Addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        Server_Sock_Addr.sin_port = 9000;

        // Connect to address set in Server Address Structure
        if(connect(sock_desc,  (struct  sockaddr  *)  &Server_Sock_Addr,  sizeof
(Server_Sock_Addr)) < 0) exit(-1);

        write(sock_desc, "A", 1);         // Write Character "A" to network
        close(sock_desc);                 // Close Socket
        return 0;                         // Return success
}
```
Text 4 : Network Client Example

```
// Include standard library header files
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <unistd.h>


// Main Function
int main (int argc, char *argv[]){

        // Setup variables for server and client socket descriptor and
        // length of client structure
        int server_sock, client_sock, client_len;
        // Create Client and Server Address Structure
        struct sockaddr_in Client_Sock_Addr, Server_Sock_Addr;

        server_sock = socket(AF_INET, SOCK_STREAM, 0);  // Create Socket

        // Setup Server Address Structure
        Server_Sock_Addr.sin_family = AF_INET;
        Server_Sock_Addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        Server_Sock_Addr.sin_port = 9000;

        // Bind server socket – exit on failure
        if(bind(server_sock,  (struct   sockaddr  *)&Server_Sock_Addr,  sizeof
(Server_Sock_Addr)) < 0) exit(-1);

        // Listen to server socket
        listen(server_sock, 5);

        while(1){  //Forever
                char ch_data;

                // Accept communication, returning client soket descriptor
                // Else exit on failure
                client_sock  =  accept(server_sock,  (struct   sockaddr   *)
&Client_Sock_Addr, &client_len);
                read(client_sock, &ch_data, 1);     // Read Data
                printf("Data Sent:%c\n", ch_data);  // Print Data to screen
        }
}
```
Text 5 : Network Server Example

When compiled and run in different consoles, on the same machine, with the server started first, upon running the client the server will print "`Data Sent:A`" to the screen.

Low Level Parallel Port control with Parapin

It would be very hard to control any device attached to a few pins of a Linux box's parallel port using the standard Parallel port driver. The driver has been designed to automatically do handshaking when connecting[35], as would be required by a printer or any other normal parallel port device. This is indeed the case with most operating systems (if the parallel port is indeed directly accessible). In order to easily control the pins of the parallel port a C library called Parapin[16] was used. This package simplifies the process required to control individual pins by automatically correcting for inverted pins and helping when dealing with the parallel ports control registers. Text 6 shows the basic code required to raise all the parallel port's data pins.

```
// Include standard library header files required for parallel port access.

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// Include Parapin header file

#include "parapin.h"

// Main Function

int main(int argc, char *argv[]){

        // Initialise Parapin - Gain access to parallel port,
        // exit with failure if refused.
        if (pin_init_user(LPT1) < 0)
                exit(-1);

        // Set data pins as outputs
        pin_output_mode(LP_DATA_PINS);

        // Raise data pins
        set_pin(LP_DATA_PINS);

        // Return success if completed
        return 0;

}
```
Text 6 : Parapin Example

Low Level Hardware Control

This is relatively simple under Linux and the first step is to gain the permissions required to write to the memory address range used by the hardware. In the PC architecture hardware registers for devices are accessible via the first 65536 bytes of the memory map. There are 2 functions that can

---

35 In the case of printers, this handshaking usually uses the centronics protocol[15].

be used in order to free up different amounts of this memory, `ioperm` and `iopl`.

The first of these functions, `ioperm`, can only be used on the first 978 bytes or the memory map. This function does however have the advantage that it allows a portion of this range to be made accessible from a specific point for a number of bytes, from 1 byte to the full range. The second, `iopl`, doesn't have this range selection and can be used to change the privilege level for the entire 65536 I/O ports. In is preferable to use the first function if possible as this lowers the chance of the program being used maliciously if the computer the program was on an exploited system, as the required range can be finely selected.

Once the required permissions have been gained `outb` and `inb` can be used to read and write bytes of data to a selected ports. Text 7 shows the code required to output binary of 170 (which is "10101010") to the data pins of a PCI parallel port, which has its base address at `0xdc00` (hexadecimal for 56320, hence out of the range for `ioperm`).

```
// Include standard library header files required for access.

#include <stdio.h>
#include <stdlib.h>
#include <term.h>
#include <unistd.h>
#include <sys/io.h>

// Main Function

int main(int argc, char *argv[]){

        // Obtain required permission to access hardware register.
        if(iopl(3) < 0){
                perror("can't get required permissions!");
                exit(-1);
        }

        // Output data to register
        outb(170, 0xdc00);

        // Return success
        return 0;

}
```
Text 7 : Low Level Register Access Example

Utilising the mouse with libgpm

The are now 3 ways in which a mouse maybe connected to a modern PC, serial, PS/2 and USB. It would be possible to write a driver for a specific mouse, using the correct device file from "`/dev`" for low level access to the device, however this would be a large amount of work and could still potentially yield buggy code.

In the open source community it is seen as silly to "re-invent the wheel" just for the sake of it, if the code already exists it is thought to be fair more efficient to utilise that. It is even better it this can be

used as an external component as any improvements to the component are then seen in all the programs which use it, without further changes being made.

It so happens that this philosophy has been used in the design of the console mouse driver that Linux uses, GPM[17]. This project has had years of development and is thus considered very stable. There is an interface that allows the user to integrate mouse control into console applications called libgpm. The use of this interface is fairly well documented[18]. A basic example is shown in Text 8.

```
// Include required libraries
#include <stdio.h>
#include <gpm.h>


// Create function to be run on mouse event
int my_handler(Gpm_Event *event, void *data){


        // Print event type, x & y mouse values
        printf("Event Type : %d at x=%d y=%d", event->type, event->x, event->y);
        // Return success
        return 0;
}


// Main Function
int main(int argc, char *argv[]){


        Gpm_Connect conn;      // Create instance of Gpm connection structure
        int c;


        // Populate structure appropriately (want everything for this example)
        conn.eventMask  = ~0;  /* Want to know about all the events */
        conn.defaultMask = 0;  /* don't handle anything by default  */
        conn.minMod     = 0;   /* want everything                   */
        conn.maxMod     = ~0;  /* all modifiers included            */


        // Open connection with Gpm Daemon – Exit on failure
        if(Gpm_Open(&conn, 0) == -1){
                // Write exit reason to standard error stream
                perror("Cannot connect to mouse server");
                exit(-1);
        }


        gpm_handler = my_handler;          // Initialise Event Handler
        while((c = Gpm_Getc(stdin)) != EOF)         // Wait for event
                printf("%c", c); // Print any other output to screen
        Gpm_Close();           // Close communications with Gpm Daemon on exit
        return 0;              // Return successful exit
}
```
Text 8 : LibGPM Example

Posix threading under Linux

The C programming language is a procedural language and as such is programmed to run the

commands in series. Threading allows the flow of the program to be split into multiple streams. This effectively allows two or more different operations to be carried out at the same time. This differs from having 2 or more different programs in that all of the threads have access to the same program variables.

It is fairly simple to create a thread in C under Linux as is shown in the simple example in Text 9.

```
// Include required libraries
#include <stdio.h>
#include <pthread.h>


void thread_function( void *ptr);         // Function to be cast off as
thread.

// Main Function
main(int argc, char *argv[]) {

        // Create identifier variable for thread
        pthread_t thread;

        // Start thread using thread_function
        pthread_create( &thread, NULL, (void*)& thread_function , NULL);

        // Output string from main function to standard output
        printf("Hello From Main Function\n");

        // Exit thread – not application in case other thread has not finished
        pthread_exit(0);
}


// Thread Function
void  thread_function ( void *ptr) {

        // Output string from thread function to standard output
        printf("Hello From Thread\n");

        // Exit thread – not application in case other thread has not finished
        pthread_exit(0);
}
```
Text 9 : pthread Example

Even this simple example highlights some of the problems that maybe experienced whilst using

threads. It cannot be guaranteed which of these threads will return first, since both run effectively independently of each other, therefore the output could be either:

```
Hello From Main Function
Hello From Thread
```

or:

```
Hello From Thread
Hello From Main Function
```

A global variable, that both threads could see, could be used to sync the activities of the two threads[36], however this presents its own problems. An error would occur if both threads were to try to use the global variable at the same time, however luckily help is available to allow the programmer to stop this from happening. Special flags, called "mutexes" can be use to ensure that only one thread can access a variable at a time. Once a mutex is created (a variable of type `pthread_mutex_t`) and initialised with the `pthread_mutex_init()` function, the programmer can use it to try and "lock" access to the variable as shown in Text 10.

```
#include <pthread.h>

pthread_mutex_t mutex;              // Create a standard mutex variable
int variable;                       // Variable to be controlled with mutex

// Function to first use mutexed variable
function() {

        // Initialise mutex
        pthread_mutex_init(&mutex, NULL);

        // Loop indefinitely
        while(1){
                // Lock mutex - or suspend until possible
                pthread_mutex_lock( &mutex );

                // Use Variable
                variable = 1;

                // Unlock mutex so variable can be used by other threads
                pthread_mutex_unlock( &mutex );
        }
}
```
Text 10 : Mutex Example Code

---

36 I.e. One thread only outputting if the variable is set, then clearing it; the other doing the reverse.

Should the mutex already be locked in another thread, execution of the thread trying to lock the mutex will be suspended until the mutex becomes available and will then be allowed to continue. It is important to remember to free up the mutex as soon as possible after the variable has been used to allow other threads to use the variable.

Conclusion

This report has described the various features of Linux, where appropriate drawing attention to similarities between in and other operating systems. A comprehensive section has been included, detailing where extra help can be found. Lastly a number of programming challenges under Linux have been explained and example code given to aid understanding.

Reference

1: Denis Howe, The Free On-Line Dictionary Of Computing, 1993, http://www.foldoc.org/

2: Various, Portable Application Standards Committee, 2000, http://www.pasc.org/

3: Various, The Jargon File, version 4.4.2, 2003, http://www.catb.org/~esr/jargon

4: Google, Google Groups, 2003, http://groups.google.co.uk

5: Free Software Foundation, Categories of Free and Non-Free Software, 2001,
http://www.gnu.org/philosophy/categories.html#ProprietarySoftware

6: Craig Peacock, Interfacing the Standard Parallel Port, 2001,
http://www.beyondlogic.org/spp/parallel.htm#3

7: Free Software Foundation, GNU Project Website, 2003, http://www.gnu.org/

8: Free Software Foundation, GNU GENERAL PUBLIC LICENSE, 1991,
http://www.gnu.org/licenses/gpl.html

9: Various, Filesystem Hierarchy Standard, 2003, http://www.pathname.com/fhs/

10: Welsh, M & Kaufman, L, Running Linux, 1996, pp.163-164

11: O'Reilly & Associates, O'Reilly & Associates Web site, 2003, http://www.oreilly.com/

12: Various, The Linux Documentation Project Web site - Howto Section, 2003,
http://www.tldp.org/docs.html#howto

13: Georgatos, F & Pinder, A, COFFEE-HOWTO, 2000,
http://www.tldp.org/HOWTO/mini/Coffee.html

14: Free Software Foundation, GCC home page, 2003, http://gcc.gnu.org/

15: Frederic Pont, Socket Programming in C, 2002, http://pont.net/socket/

16: Matthew, N & Stones, R, Beginning Linux Programming - Sockets Example, 1996, pp.453-485

17: Jeremy Elson, parapin -- a Parallel Port Pin Programming Library for Linux, 2003,

http://www.circlemud.org/~jelson/software/parapin/

18: Nico Schottelius, freshmeat.net: Project details for GPM, 1999,

http://freshmeat.net/projects/gpm/?topic_id=136%2C861

19: Pradeep Padala, Mouse Programming with libgpm, 2002,

http://www.linuxjournal.com/article.php?sid=4600

Bibliography

- Welsh, M & Kaufman, L, Running Linux, O'Reilly & Associates, 1996
- Siever, E, Spainhour ,S, Figgins, S & Hekman, J, Linux in a nutshell, O'Reilly & Associates, 2000
- Matthew, N & Stones, R, Beginning Linux Programming, Wrox Press, 1996
- Various, The Linux Documentation Project, http://www.tldp.org