Porting Open Source applications on MPE/iX

A case Study of Samba-3.0.22

Author: Vidya Sagar

Version 1.0

Table of Contents

Revision History

Version	Date	Description of Revision
0.1	31-May-2007	Initial Draft (By VS)
0.2	25-Jun-2007	Milestone2 (By VS)
0.3	29-Jun-2007	Milestone 3 complete paper (By VS)
0.4	05-Jul-2007	Milestone 4 partially reviewed paper(Reviewed By Jim Hawkins and Lab)
0.5	18-Jul-2007	Milestone 4(Reviewed by Jim Hawkins)
0.6	19-Jul-2007	Milestone 4(Reviewed by Jim Hawkins)
0.7	04-Aug-2007	Milestone 4(Reviewed by Jim Hawkins)
0.8	08-Aug-2007	Milestone 4(Reviewed by Jeff Bandle)
0.9	08-Aug-2007	Milestone 4(Reworked by VS)
0.10	14-Aug-2007	Milestone 4 (Reviewed by Jeff Bandle)
0.11	15-Aug-2007	Milestone 4 (Reworked by VS)
0.12	20-Aug-2007	Milestone 4 (Discussed with Jim Hawkins and Suresh Reddy and Reworked by VS)
0.13	21-Aug-2007	Milestone 4 (Reworked by VS on Jeff's Comments)
1.0	26-Sep-2007	Publish to Jazz.

Executive Summary

This paper is intended to illustrate the details of porting one of the most popular and useful open source applications, Samba version 3.0 22, from the internet and interoperability domain. The paper should provide sufficient information to
1) Refresh a new version of Samba on MPE/iX,
2) Apply future patches released by Samba organization (www.samba.org) and
3) Quickly fix defects in Samba/iX

The primary target audience of this paper is those programmers involved in any of the above mentioned activities that can be classified into two types: 1) a person familiar with MPE/iX but less familiar with open source or the GNU tools, 2) a person familiar with open source and the associated tools, but less familiar with MPE/iX and some of MPE/iX's POSIX limitations. This paper is based on the porting of Samba-3.0.22 by the vCSY lab and suitable examples are provided based on their experience.

While this paper discusses a port of Samba, it is not strictly restricted within the domain of Samba. A *few* selected porting concepts are generalized to provide some information to the programmers to enable them to port any Open Source application that conforms to POSIX standards. This motivates the secondary target audience who wants to port any Open Source or UNIX application on MPE/iX.

This paper also discusses the GNU tools in sufficient enough detail to give an idea to beginners, the POSIX interface implementation on MPE/iX and the MPE/iX limitations, with ideas about how to overcome them. Finally this paper also describes the details of Samba porting, starting from downloading the application source through the porting steps to follow until one can use the new Samba to access files on an MPE/iX system.

Introduction

The MPE/iX implementation of POSIX.1 C APIs and POSIX.2 conforming shell makes many UNIX targeted applications eligible to be ported on MPE/iX. Further, while many recent 'Open Source' applications are based upon a POSIX.4 standard, there exists enough of a POSIX framework to allow a useful implementation of such applications. Samba is one such application which provides a set of file server/sharing interoperability features between Windows and UNIX like systems. The MPE/iX version of Samba, sometime called Samba/iX, allows Windows based systems to access Files stored on MPE/iX Servers.

The paper starts with a very brief background on MPE/iX, POSIX compliance and GNU tools. In this paper we will describe how to port (refresh) a new version of Samba starting from downloading the source code through bringing up a new Samba/iX file server on a MPE/iX System. One major subject to be covered is how to evaluate which features of an application like Samba can be supported by MPE.

We also discuss how to configure the build machine for porting from the installation of GNU tools and utilities, through the creation of build accounts and setting proper capability permissions. The porting of the Samba code is the basis of this example.

The porting section discusses the process of how Samba has been successfully ported on MPE/iX, followed by some common problems, and a few performance notes applicable for Samba porting. The porting section also discusses how to apply patches released by the Samba organization. It is assumed that the reader has basic knowledge of the C programming language and UNIX environment (shell, system calls, file systems, signals etc).

The differences between UNIX systems and MPE/iX from a programming perspective are also outlined as a part of the evaluation of features in the application considered for the port.

Conventions:

The paper follows a few typographical conventions listed below:

1. "I" refers the author who shares their experience through this paper and "You" refers the audience/reader (porter) of this paper.
2. Programs text/snippets, variables, command, scripts, file names are colored set in `courier new` font.
3. Examples and demonstrations are set in `courier new` font and colored blue.
4. When we show interactions with MPE/iX CI (command interpreter) we use colon (":") as CI prompt and to show interactions with MPE/iX POSIX shell we use shell prompt as "shell/iX".

5. All the links and bookmarks in this document are blue underlined.
6. All the examples and command demonstration have used 4 way series 969-400 system installed with MPE/iX C.65.00 FOS, POSIX version A.50.02, Samba-3.0.22, and gcc version 3.2.

1 Background on MPE/iX, POSIX compliance, and GNU Tools

In this section we will review some general concepts regarding MPE/iX, its implementation of POSIX.1 and POSIX.2 standards and the GNU (or "Open Source") tools. If you are already familiar with MPE, POSIX and Open Source tools you may skip this section and start with "build machine setup."

MPE/iX and POSIX compliance

The IEEE standard for a Portable Operating System Interface, or POSIX, defines a standard set of operating system interfaces and an environment to support source level application portability. POSIX specifies the functions and services an operating system must support and the application programming interface to these services and functions. POSIX itself is made up of a number of sub-standards typically designated ".N", not all of which MPE/iX supports.

Specifically, MPE/iX provides POSIX 1003.1 (or POSIX.1), which defines a standard set of programmatic interfaces for basic operating system facilities, and POSIX 1003.2(or POSIX.2), which specifies an interactive interface that provides a shell and utilities similar to those provided by the UNIX operating system. These POSIX standards are integrated in the MPE XL operating system to form the MPE/iX operating system, which runs on the HP e3000 systems.

The programmatic interfaces and directory structure of POSIX.1 allows MPE/iX users to use POSIX functionally without any impact on existing HP 3000 applications. Moreover, MPE applications are able to access POSIX files, and POSIX applications are able to access MPE files. Thus, MPE/iX provides interoperability and integration between MPE and POSIX applications and data.

The book, MPE/iX Developer's Kit Reference Manual Volume 1 [1] and MPE/iX Developer's Kit Reference Manual Volume 2 [2], describes the MPE/iX implementation of the IEEE 1003 standards 1003.1 for C language bindings and 1003.2 for Shell commands and utilities.

The POSIX/iX library is implemented according to the standards set forth in the 1990 revision of the POSIX.1 standard. POSIX.1 standard defines over 200 C application programming interfaces (APIs) along with type definitions, header files and data interchange format. These APIs are standard programming interfaces to make system calls, I/O calls, and general library calls.

POSIX is significantly different from MPE in a number of technical areas such as directory structure, file system, security, user identification, file naming, process management, and signals.

Directory Structure: MPE has a fixed, three-level directory structure. In this model, the directory tree consists of one or more accounts. Each account contains one or more groups and each group has zero or more files in it. On the other hand, POSIX supports the notion of a hierarchical directory structure.

MPE groups and accounts are different from the POSIX directories because of the special information contained in them. To integrate the POSIX and MPE directory structures successfully, this special information from MPE groups and accounts has been removed to accommodate the new directory structure. On MPE, accounts are used to manage collections of users for file sharing. Each MPE account entry contains a pointer to a list of users that are members of that account. Accounts and groups are distinguished from POSIX files and directories based upon how they are created; using newacct/newgroup or using mkdir.

In the new directory structure all users are registered in the UID (user identifier) database required by POSIX, and the combination of the UID and GID (group identifier) databases replaces most of the information formerly held in the MPE account and user directory nodes.

File Naming: In designing the file naming rules, the main objective was that the existing MPE interfaces such as MPE intrinsic(s) and command interpreter (CI) commands should be able to refer to all objects in the same way they did before MPE/iX. The name server used by POSIX 1003.1 functions and POSIX 1003.2 commands is the POSIX name server.

However, MPE interfaces such as intrinsic(s) and CI commands escape to the POSIX name server when the file name begins with the "." or "/" escape characters.

File Access and Security: In MPE/iX, POSIX 1003.1 file permission bits have not been implemented as a separate access control mechanism. Instead, POSIX 1003.1 functions support the file permission bits via the MPE ACD mechanism. ACDs themselves have been enhanced to enable the ACD mechanism to operate as a POSIX 1003.1 additional access control mechanism and to provide directory access control.

POSIX 1003.1 applications use POSIX file permission bits to specify access permissions and are unaware that file permission bits are implemented on top of the ACD mechanism. On the other hand, MPE applications never deal with POSIX file permission bits; they deal with ACDs, the file access matrix, and lockwords. When a POSIX 1003.1 interface such as open() creates a file, an ACD is assigned to the file as part of the file creation operation. When the POSIX 1003.1 function chmod() is invoked to set access permissions for a file or directory, ACD information is manipulated. Similarly, the stat() and fstat() functions evaluate an ACD and map the access permissions granted by the ACD into file permission bits using this mapping in reverse.

Process Management: In MPE/iX, if a parent process terminates without waiting for all of its child processes to terminate, the resulting "orphaned" child processes are terminated immediately prior to termination of the parent process. The implementation does not allow orphaned child processes to be adopted by a system process. Your application should not rely upon orphaned child processes being adopted by a system process. No user process can be a controlling process. Only system processes, such as the MPE/iX Command Interpreter (CI), are allowed to be controlling processes.

Signals: In MPE/iX, signals cannot be delivered to a process while that process is executing system code. The signal remains pending until control returns to the calling process. A sending process cannot rely on timely delivery of a signal if the target process is executing system code. For example, if a signal is generated for a process when the process is executing a call to read() or write(), the signal remains pending until the function call returns either after successful transfer of data or when an error is encountered. If multiple occurrences of a signal are generated while that signal is blocked and pending, each occurrence of the signal is left pending. If the signal is later unblocked, multiple instances of that signal can be delivered to the process.

Refer Implementation details of POSIX on MPE/iX [5] for more details and design considerations of POSIX on MPE.

GNU tools/utilities

This section is intended to provide some basic idea about the GNU tools and various assorted 'Open Source' utilities which are used to compile and build any open source freeware portable application.

It is often quite cumbersome to distribute a portable software package in a way that takes care of all the dependencies. To solve this problem GNU provides various tools and utilities. A few of the most important are discussed briefly here and will be referenced also in the porting section of this document.

gcc: GNU c and c++ compiler. The gcc compiler was ported on MPE/iX by Mark Klein and can be downloaded from http://jazz.external.hp.com/src/gnu/gnuframe.html. It contains all the necessary tools required for a POSIX compliant software package to be built.

make: *make* is a powerful and very useful utility to maintain a set of program generated and interdependent files. The UNIX *make* utility was originally designed for maintaining C program files in order to prevent unnecessary recompilation. However, *make* is eminently useful for maintaining any set of files with interdependencies. For example, *make* can be used to maintain C++ or HTML source code. In fact, NIS (Network Information Service) uses *make* to maintain user information. The *make* utility provides a way of codifying the relationships between files as well as the commands that need to be generated to bring files up to date when they are changed. The *make* utility provides a powerful, nonprocedural, template-based way to maintain files. You tell *make* what needs to be done and supply some rules, and make figures out the rest.

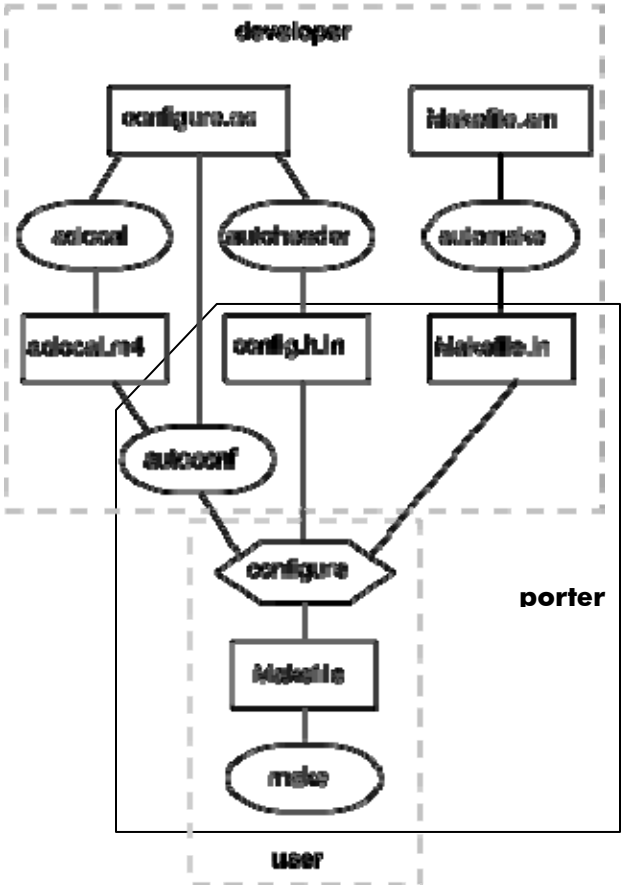
makefiles: A *makefile* is set of make commands i.e build rules governing the compile, link and installation of the software package. The *makefile* describes what set of files can be built and how to build those files. Four types of lines are allowable in a makefile: target lines, shell command lines, macro lines, and make directive lines (such as include). Comments in a *makefile* are denoted by the pound sign (#). When you invoke make, it looks for a file named *makefile* in

your current working directory. If *makefile* does not exist, then make searches for a file named *Makefile*. If you don't want to use one of the default names, other files can be used with the "-f command-line" option. The convention used to identify makefiles not named *makefile* or *Makefile* is to use the *.mk* suffix (for example, foo.mk).

autoconf: *Autoconf* is a tool for producing shell scripts that automatically configure software source code packages to adapt to many kinds of Posix-like systems. *Autoconf* takes input from a template file named `configure.ac` (or `configure.in`) and generates a configure script conventionally called `configure`. The configuration script generated, automatically checks for each feature the OS supports that is necessary for the software package. You may find an entry for the well known UNIX flavors in the template file which defines specific behavior corresponding to the feature. Changes relative to MPE/iX are required to be done in the configuration template file. The generated script, `configure`, also produces a few intermediate and log files and those are: `Makefile(s)`, `config.status`, `config.cache` (optional), `config.log`, and an optional C header file which defines some `#define` directives. One or many (in the source subdirectories) `Makefile(s)` are created by script `configure` by taking input from file `Makefile.in`. For more information on *autoconf* and to learn the syntax rules of template file it is recommended to visit the site http://www.gnu.org/software/autoconf/manual/autoconf.html.

autoheader: This tool reads the template file `configure.ac` (or `configure.in`) and generates an input file `config.h.in` which in turn is read by script `configure` to produce a C header file conventionally called `config.h`.

The diagram below shows a typical build environment of an open source portable software package.



In this paper we play the roles of "developer" & porter rather than "user". That is we will not modify the "`configure`" script or `Makefile(s)` directly as *autoconf* recreates script `configure` and the later recreates `Makefile(s)` by reading `Makefile.in`. Rather, the changes will be done in the template or input files (`configure.ac`, `Makefile.in` and `config.h.in`) so that the derived "user" steps will execute correctly and consistently.

MPE/iX header files and libraries

It is a good idea to be familiar with what header files and libraries are available on MPE/iX which will be used by the portable applications. In this section we will discuss the header and libraries which are a part of FOS as well as other headers and libraries like libbsd and POSIX wrappers required for porting applications. Please refer to the Build Machine configuration section which discusses how to install libbsd and POSIX wrappers.

1.A.1 MPE/iX Headers

Header files are the source of routine and data type declarations which require their inclusion into the application source file using `#include` directive to achieve successful compilation of the code. The standard convention requires that header files are placed into the directory named "include" and libraries are placed into the "lib" directory.

MPE/iX header files are available through three sources:

1. System and libc headers: The system and C-library header files are generally located inside /usr/include and the subdirectories within. For example, some system specific header files reside inside /usr/include/sys directory. Most of the compilers e.g. gcc, cc, c89 by default search header files in /usr/include directory. Please refer to the manpage of the specific compiler to know the default header file search directory. If you want to include some specific header files two strategies can be followed:

   - Defined in "C" Source, using #include directive: Supply the path of header file relative to /usr/include directory. In order to include header file from current directory the header file name should be enclosed with double quote.

```
#include <sys/time.h> /* Include /usr/include/sys/time.h */
#include "yourhdr.h" /* Includes yourhdr.h from current directory */
```

   - Defined at Compile time: Using –I compiler flag: You can tell compiler to search the header in the directories using –I<dir> -I<dir2> … Refer manpage of gcc for more details.

2. libbsd headers: Not all the routines required are available through system and C libraries. A fair amount of routines declarations missing in system and libc headers are available through libbsd and their corresponding header files located inside /usr/include/bsd. The header files are listed in Appendix G.

   It is necessary to add –I/usr/include/bsd into your compiler options to tell the compiler to search header files required for BSD library.

3. POSIX wrappers: Additional routines missing from system, libc and libbsd headers and required by MPE/iX are made available through POSIX wrappers. POSIX wrappers are installed in the POSIXC60 account. Hence their header files are located at /POSIXC60/include and libraries are at /POSIXC60/lib. Appendix H lists those header files.

   Add –I/POSIC60/include to compiler option to tell compiler to search the include directory of POSIX wrappers.

Many header files included into the application source code may not really required on MPE/iX. Such header files should be by passed from the compilation unit using the best portable strategy, `#ifndef#else#endif` method, shown below:

```
#ifndef mpeix
#include <sys/param.h> /* This header file is not required for MPE/iX */
#endif
```

On compiling the above code with option `-Dmpeix`, compiler preprocessor does not include the header file "sys/param.h". Conversely, `#ifdef#else#endif` is widely used approach to include system specific piece of code in portable software environment.

Mark Bixby's porting notes [3] lists a few symbols found absent in the above mentioned header files. This porting note also lists the missing header files on MPE/iX.

1.A.2 MPE/iX Libraries

Normally, an application required to be linked with many libraries. Some application requires library only at compile time and some at runtime as well. Say for example Samba requires the POSIX wrappers at runtime also; hence it becomes necessary to include the runtime library into the package.

MPE/iX system libraries are available in two types NMRL archive library and NMXL shared library. NMRL and NMXL are available in MPE namespace in as NL.PUB.SYS and XL.PUB.SYS and both contain all the routines available in various assorted libraries in /lib and /usr/lib directories. The libraries available at /lib and /usr/lib are system and C libraries. An application is linked by default with C libraries whenever required. To link with specific libraries programmer need to tell the name of library using "-l' flag and their search path using "-L" flag. Samba requires to be linked with following libraries apart from the default libraries:

```
/POSIXC60/lib/libsocket.a
/POSIXC60/lib/libunix.a
/usr/lib/socket.a
/lib/libsvipc.a
/usr/lib/libcurses.a
/SYSLOG/PUB/libsyslog.a
```

You need to specify the libraries by `-lposix60 -lunix -lsyslog -lsocket -lsvipc –lcurses` and additional search path apart from default search path by `–L/POSIXC60/lib –L/SYSLOG/PUB.` Compiler by default searches for libraries in /lib and /usr/lib. By specifying –Ldir, compiler is being told to search in the directory "dir" in addition to the default search path for the libraries specified by –l oprtion.

The compiler option –Xlinker is used to pass options to the linker. "`–Xlinker –Wl`" is used to pass an option to the system specific linker that gcc does not know. This is required when you want to link with the MPE namespace libraries like NL, XL, LIBCPXL etc. Samba is required to be linked with LIBCPXL which can be done by specifying the option `-Xlinker –Wl,xl=/$HPACCOUNT/$HPGROUP/LIBCPXL` to gcc compiler; where $HPACCOUNT is the shell variable referring to the account you have logged in, $HPGROUP is again a shell variable refers to the current working group. That is if you logged in as MGR.SAMBA, SMB3022 the $HPACCOUNT refers to SAMBA and $HPGROUP refers to SMB3022. The MPE/iX linker LINKEDIT will be invoked to link with LIBCPXL. Refer to the manpage of gcc for more details on –Xlinker. The libraries linked this way must be available during runtime at the same location from where it has been linked. Hence, such type of libraries must be packaged along with the application for distribution. For Samba we package and distribute LIBCPXL along with the Samba patch.

The discussion about header files and libraries will be helpful in configuring the compilation flags and switches to properly include header files and properly use libraries to achieve a successful and error free build of the application under port. Please refer porting section on how the compilation flags and switches are set for Samba porting.

Next, we will discuss a few major MPE/iX limitations and issues we encountered while porting Samba-3.0.22 and strategy to overcome them.

MPE/iX limitations and Major issues

This section discusses the major MPE/iX issues and limitations which a porter needs to be aware. Having cognizant about the limitations and issues saves porter's effort to debug the problem during compilation and also at runtime. We have discussed following major limitations and issues:

- **UID and GID**: MPE/iX's POSIX implementation of UIDs and GIDs is incomplete in the sense that it lacks the UID=0. In UNIX environment the superuser (root) has UID zero. The Unix root concept is roughly analogous to the MPE concept of SM capability and/or PM capability. Many (almost all) portable UNIX applications sometimes set their effective UID to zero in order to perform some privileged operations. Samba is not an exception and hence the UID=0 has been simulated in lib/replace.c. Lars Appel simulated following routines which are widely used in Samba to get and set uids and gids:

```
gid_t getegid(void);
uid_t geteuid(void);
gid_t getgid(void);
uid_t getuid(void);
```

```
int setgid(gid_t);
int setuid(uid_t);
```

The non-simulated version of setuid() must be invoked in privileged mode. The simulated version takes care of this as well as one change directory bug listed below. setgid() also required in privileged mode (Invoke GETPRIVMODE()) but does not do anything as MPE/iX does not support direct gid switching. Please see notes on supplementary groups discussed later.

setuid() side effect: There is a potential bug in MPE/iX POSIX implementation of setuid() is that the current working directory is changed to the home directory of the user. Also the group id is changed to the account. The workaround is to restore the cwd as shown below:

```
/* Save the CWD */
char saved_cwd[PATH_MAX+1]; \
getcwd(saved_cwd, sizeof(saved_cwd)); \
/* set uid */
setuid(x);
/* Restore the CWD */
chdir(saved_cwd);
```

Supplementary Groups: In UNIX environment one user can be associated with many groups and this information is kept in /etc/groups. MPE/iX lacks supplementary group and hence /etc/groups. MPE/iX user associates them with only one group and that is the account in which the user belongs to. For example user MGR.SAMBA can be the member of only one group, the MPE/iX account SAMBA. Hence, MPE/iX does not `initgroups()` and `setgroups()` routines. The routines `getgroups()` returns 0 and Samba is very well programmed to handle these deficiencies.

- **Missing telldir() and seekdir():** MPE/iX lacks the implementation of telldir() and seekdir(). The workaround is to return 0 for telldir() call and do nothing for seekdir() call. Doing this would force the application to use the key as directory name rather than using the return value of telldir(). Even if you try to implement your own version of telldir() and seekdir using MPE/iX intrinsic it would not work properly as expected for directories referring to group and account. I fake the absence of telldir() and seekdir() as follows:

```
#ifdef mpeix
long int telldir(){
      errno = 0; /* Make sure there is no error returned */
      return 0;
}

void seekdir(DIR* dp){
      errno = 0; /* Make sure there is no error returned */:w
      return;
}
#endif /*mpeix*/
```

- **Unsigned long long**: The data type unsigned long long (ULL) is supported on MPE/iX. gcc-3.3.1 and gcc-3.2 has a potential bug that truncates the ULL to unsigned long. Following is the byte wise output of an ULL when compiled with gcc-3.3.1:

```
B[0]: 0
B[1]: 0
B[2]: 0
B[3]: 0
B[4]: ff
B[5]: ff
B[6]: ff
B[7]: ff
```

If your code is using ULL then either force them not to use ULL and use unsigned long instead or upgrade to gcc 4.0.2 from http://invent3k.external.hp.com/~MANAGER.DIS/

- **Sockets:** MPE/iX sockets impose various limitations listed below:

- ✓ The maximum size of any single socket I/O request on MPE/iX is 30,000 bytes. Many applications will natively attempt do to I/O in chunks of 65,536 bytes (or larger). These I/O calls will fail on MPE, and you will need to modify the application to do socket I/O in smaller chunks
- ✓ MPE/iX bind() should be invoked in privileged mode to bind privileged (<1024) ports. Also, bind() binds only to wildcard interfaces.
- ✓ MPE/iX cannot connect to datagram sockets.
- ✓ MPE/iX socket descriptors MUST be manipulated using sfnctl() instead of using fcntl().
- ✓ Duplicating socket descriptors via dup(), dup2(), fcntl(F_DUPFD), or sfcntl(F_DUPFD) will result in a system abort the next time you call fork().
- ✓ MPE/iX setsockopt() does not support following socket options:

    - SO_KEEPALIVE
    - SO_REUSEADDR
    - SO_SNDBUF
    - TCP_NODELAY

  Refer Mark Bixby's porting notes [3] for more on socket limitations.

- **MPE/iX processes cannot become a deamon:** Samba become_daemon() does not easily apply to MPE, there is no setsid() routine available and the process management concepts are quite different in the MPE area. One might use DETACH process but using a server job is easier. You can achieve the same just by making the process visible in MPE namespace and run that process as job. The job JSMB inside samba package is used fro that purpose and looks like:

  ```
  :print JSMB
  !job jsmbstrt,manager/sagar.sys;pri=CS
  !  xeq smbd.smb3022.samba "-D -p 139"
  !eoj
  ```

- **POSIX GTI not supported on MPE/iX**: The standard POSIX.1 porting environment on HP-UX platforms supported the POSIX GTI or General Terminal Interface libraries. These library routines allow control for how terminal I/O is executed in a POSIX application beyond normal read/write activities. An example of a difference can be found with the read function. With a standard MPE/iX application, a read request for data will block the application and wait until the read data is received. In a POSIX application, there is no concept of a blocked read; as data is entered by the user it is forwarded to the system for processing. This is just one example of a major difference between MPE/iX standard terminal I/O and POSIX GTI I/O. When porting a POSIX application, if terminal I/O is a component of that applications operation, the porter will need to consider the method of structuring the terminal I/O activities. One method is to encapsulate all terminal I/O operations into a single, porter defined library routine that can be tailored to the needs of the application.

It is recommended to read the function and miscellaneous link from Mark Bixby's porting notes [3], to learn more about missing functions and their workaround. This page also contains the well known bugs which should be taken care to work your port properly.

So with this background on MPE, POSIX, GNU tools, MPE/iX header files and libraries, and a few MPE/iX major limitations, one should have a better understanding of the foundations for Porting "UNIX" applications to MPE/iX. Now we'll look at setting up an actual MPE/iX system to start our Samba port.

2 Build Machine Configuration

This is the first step of porting activity. Your build machine must have all the necessary packages installed to build an application and will likely require modifications to MPE Directories as well as some POSIX environmental changes. These steps are covered briefly below and in more detail in the following sections.

As a base for your porting efforts we strongly recommend that your build machine be running MPE/iX 6.5, 7.0 or 7.5 preferably with the POSIX shell version A.50.02. The POSIX shell and libraries are part of the MPE/iX Fundamental Operating System (FOS) and are assumed to be available on any MPE/iX machine. Installation of MPE/iX OS FOS and Subsystems is beyond the scope of this paper.

Next, configuration of a fresh build machine for any "Open Source" port typically requires the following four packages be installed:

- GNU gcc compiler and utilities
- BSD libraries and headers
- PERL
- POSIX wrappers

Once these packages are in place application specific MPE/iX Directory and User changes will need to be made. After account structure setup we discuss how you might customize your POSIX environment to make is appear more like those you might be familiar with on Unix.

NOTE: You may be able to start from the "setting up the account structure" section, if you have already the required packages installed on your build machine.

Installing essential tools and libraries

This section discusses how to install the essential four packages on your build machine.

2.A.1 GNU gcc compiler and utilities

Building any POSIX compliant application on MPE/iX requires use of the GNU gcc compiler and various other assorted utilities available from http://jazz.external.hp.com/src/gnu/gnuframe.html. Installation of gcc and GNU utilities is very simple and just a three step process:

1) Download the GNU gcc compiler and other utilities package ported by Mark Klein to your windows machine. Download the latest available version as many applications complain if the latest gcc is found absent on the host.
2) FTP the downloaded file(s) in binary mode to the /tmp directory of your build machine.
3) Install GNU gcc compiler and tools by following the installation instruction (http://jazz.external.hp.com/src/gnu/install_nmstore.html) or

   Enter into the MPE POSIX shell by issuing `sh.hpbin.sys -L` command at CI prompt and run the script `INSTALL.gcc`.

   Note: The script `INSTALL.gcc` is listed in Appendix C, which can be copy/pasted into a new file `INSTALL.gcc` on build machine in any directory and execute that by issuing command `sh INSTALL.gcc` at POSIX shell. Ensure that the downloaded files are available in /tmp directory.

To know the current version of gcc installed issue the command `gcc -v`. The output should take one of two forms:

```
shell/iX> gcc -v
gcc: not found
```

Will be returned if gcc is not installed otherwise:

```
shell/iX> gcc -v
Reading specs from /usr/local/lib/gcc/hppa1.0-hp-mpeix/3.4.2/specs
Configured with: ../configure --with-gnu-as --disable-pic
Thread model: single
gcc version 3.4.2
```

Will show the version of gcc.

NOTE: gcc versions 3.4.2 and older have a potential bug of inconsistent handling of data type long long, which is already discussed in limitations section.

2.A.2 BSD libraries and headers

As discussed in the MPE/iX header and libraries section the build machine should have the libbsd package installed. Install the BSD libraries and headers available at http://jazz.external.hp.com/src/bsd/libbsd.html?jumpid=reg_R1002_USEN. The steps are:

1) Download the libbsd package from the link specified above to your windows machine.
2) FTP that to the target machine inside the /tmp directory.
3) Finally to install the package on target MPE/iX machine, either follow the instruction on the same web page or use the script INSTALL.libbsd listed in Appendix E.

2.A.3 PERL

GNU utilities like autoheader require PERL as these utilities are nothing but PERL scripts. PERL freeware are available for MPE/iX at http://jazz.external.hp.com/src/hp_freeware/perl/?jumpid=reg_R1002_USEN . You need to register in order download the PERL package. Complete the freeware agreement, download PERL package on windows machine, ftp the package to the target machine inside /tmp and finally follow the instruction on the same web page to install PERL.

2.A.4 POSIX wrappers

The last package required to be installed on the target build machine is POSIX wrappers available at http://jazz.external.hp.com/src/px_wrappers/index.html. The steps to install POSIX wrapper are:

1) Download the complete package (compile and runtime) from the link specified above to your windows machine.
2) FTP that to the target machine inside the /tmp directory.
3) Finally to install the package on target MPE/iX machine, either follow the installation instruction (http://jazz.external.hp.com/src/px_wrappers/px_wrappers.install) on the same page or use the script INSTALL.px_wrappers listed in Appendix D.

Note: Samba requires the use of the MPE Porting Wrappers package both at compile time and run time. But Samba also requires use of the LIBCPXL runtime library which must be manually set up by running the script `patch-libcpxl` placed in Appendix F as show below:

```
shell/iX> sh patch-libcpxl
```

This script does the following:

1. Copies /POSIXC60/HPBIN/LIBCPXL to /$HPACCOUNT/$HPGROUP/LIBCPXL
2. No-ops the getpw() function in the newly copied library

The getpw() function does not behave in a manner consistent with standard MPE security concepts. We therefore make getpw() function non operational since it is not used by Samba. *Please note that this script should be run only after creating account and group on the target build machine (discussed below) and user should logged on into the same account and group configured for port.*

Setting up account structure

You should be logged on to the target MPE/iX machine as the superuser MANAGER.SYS or the user having capability to create, modify and purge account, group and user. As a check the “CAP” values for your user should be as shown below:

```
:listuser manager.sys
********************
USER: MANAGER.SYS

HOME GROUP: PUB                 PASSWORD: **
MAX PRI   : 150                 LOC ATTR: $00000000
LOGON CNT : 1
CAP: SM,AM,AL,GL,DI,OP,CV,UV,LG,PS,NA,NM,CS,ND,SF,BA,IA,PM,MR,DS,PH
:listacct sys
********************
ACCOUNT: SYS
```

```
DISC SPACE: 11327520(SECTORS)   PASSWORD: **
CPU TIME  : 725144(SECONDS)     LOC ATTR: $00000000
CONNECT TIME: 992678(MINUTES)   SECURITY--READ    : ANY
DISC LIMIT: UNLIMITED                    WRITE   : AC
CPU LIMIT : UNLIMITED                    APPEND  : AC
CONNECT LIMIT: UNLIMITED                 LOCK    : ANY
MAX PRI  : 150                           EXECUTE : ANY
GRP UFID : $05670001 $0981A6C5 $000045CE $130028B8 $1B747DCB
USER UFID: $00000000 $00000000 $00000000 $00000000 $00000000
CAP: SM,AM,AL,GL,DI,OP,CV,UV,LG,PS,NA,NM,CS,ND,SF,BA,IA,PM,MR,DS,PH
```

The account structure for Samba can then be created as follows:

```
:NEWACCT SAMBA,MGR
:ALTACCT SAMBA;CAP=AM,AL,GL,ND,SF,BA,IA,PH,PM;ACCESS=(R,L,X:ANY;W,A:AC)
:ALTGROUP PUB.SAMBA;CAP=BA,IA;ACCESS=(R,L,X:ANY;W,A,S:AL)
:NEWUSER GUEST.SAMBA
:ALTUSER GUEST.SAMBA;CAP=ND,SF,BA,IA,PH;HOME=PUB
:ALTUSER MGR.SAMBA;CAP=AM,AL,ND,SF,BA,IA,PH;HOME=PUB
```

Next we should devise an MPE Group name unique to the version of Samba we are porting What should the Group name be? For MPE/iX “group” names are restricted to 8 characters, upper case only [A-Z][0-9 | A-Z] For any new application the names of account/group should be decided by the porter but should be meaningful and must be unique. Here version is “SMB3022”, if Samba-3.0.22 is considered for port.

```
:NEWGROUP <version>.SAMBA
:ALTGROUP <version>.SAMBA;CAP=BA,IA,PH,PM;ACCESS=(R,L,X:ANY;W,A,S:AL)
:PURGELINK /SAMBA/CURRENT
:NEWLINK /SAMBA/CURRENT,<version>
:PURGELINK /usr/local/samba
:NEWLINK /usr/local/samba,/SAMBA/CURRENT
```

The reason behind adoption of such an accounting structure is to provide flexibility to switch among the various releases of same product without actually purging the other revisions.For example, assuming this system previously had the Samba-2.2.8a version installed; to revert back to Samba-2.2.8a, just we need to execute the following commands:

```
:purgelink /SAMBA/CURRENT
:purgelink /usr/local/samba
:newlink /SAMBA/CURRENT,SMB228A
:newlink /usr/local/samba,/SAMBA/CURRENT
```

At this point you’ve set-up the necessary MPE User and Directory structures for a successful Samba port.

Other application ports might need some additional packages like OpenSSL etc which requires to be installed apart from the four basic packages described above. The first place to search for any additional package is http://jazz.extenal.hp.com. The installation procedures for the packages on jazz are similar to what we have discussed for the installation of basic four packages and are readily available along with the package. Prerequisite packages for a port can be found from the application’s documentation.

2.A.5 Setting up the POSIX environment

It would be good idea to set up your POSIX environment during account setup. MPE provides a set of predefined commands via the HPPXUDC.PUB.SYS file. Issue the following command on CI prompt while logged in as MGR.SAMBA:

```
:SETCATALOG HPPXUDC,APPEND;ACCOUNT
```

Normally POSIX environment are controlled by various shell variable, which you can define in the *.profile* file in home directory or home group. On entering into the POSIX shell, this file is read and sets the variable if specified. Edit that file to include the following entries:

```
#create some aliases
alias ll="ls -l"
alias lsf="ls -F"
alias mpe="callci ci,2"
alias scan=”find . –type f | xargs grep”
alias rm=”rm –I”

#set the PATH variable to search bin and sbin of your group for any command
export PATH=$PATH:/usr/local/bin:/bin:$HOME/bin:/usr/local/samba/bin
export PATH=$PATH:/$HPACCOUNT/$HPGROUP/bin:/$HPACCOUNT/$HPGROUP/sbin
#set the MANPATH variable to search man directory of your group for manpages
export MANPATH=/usr/local/man:/usr/man:$HOME/man
export MANPATH=$MANPATH:/usr/local/samba/man:/$HPACCOUNT/$HPGROUP/man
export CC=gcc
```

In POSIX the editor we use is `vi`. The startup file for vi is *.exrc* resides in logon user's home directory/group. Edit the *.exrc* to override the default behavior of vi with following entries:

```
:map ^[B j
:map ^[D h
:map ^[C l
:map ^[V ^B
:map ^[U ^F
:set nows
:set ic
```

These settings are not mandatory but prove to be very helpful, you can use the *.exrc* file to implement your own favorites. Now we have the build machine ready for porting Samba-3.0.22. Likewise you can setup your build machine for any port.

Build machine Summary/Conclusion

At this point one should have in place the basic file sets necessary to perform an Open Source Port, have built a basic file structure on MPE and done some customizations to the command line environment to make it more “Unix-like”.

3 Porting

This section describes how to port Samba-3.0.22 on MPE/iX which has recently been done by vCSY, by making use of the various concepts discussed in earlier sections. Samba-2.2.8a was already ported on MPE/iX, hence this is more of refreshing the Samba to new version instead of doing a fresh port. Normally, porting an application involves the steps specified below:

1. Analyze the application considered for port for the features that can be supported on MPE/iX and prepare a list of the same.
2. Download the source code from the application's official site.
3. Uncompress and untar the source code (also referred as tarball).
4. Apply the “diff” of previously ported version, if exists. This indicates the refresh of already existing port.
5. Adjust, generate and run script “configure” to make the application ready for compilation.
6. Build prototypes.
7. Modify source code to accommodate known MPE/iX limitations and issues.
8. Compile, link and install, i.e., build the application and install.
9. Set proper capabilities for the binaries.

10. Determine unresolved symbols.
11. Resolve unresolved symbols and rebuild and install.
12. Test all the features you think should be supported in step 1.
13. Document the unsupported feature.
14. Finally bundle the deliverables, documentation and release.

A fresh port does not involve step 4 to apply a patch. Rather the porter modifies the necessary configuration files (`configure.in`, `Makefile.in etc`) manually wherever required which is here done by the POSIX command patch. All the remaining steps are same.

To apply a fix released by the Samba organization, if you are sure that this fix can be applied to Samba/iX you can start with step 4.

We will discuss all the steps mentioned above to successfully port Samba-3.0.22 in detail as follows.

3.1 Analysis of application to identify the features that can be supported

Due to various limitations of MPE/iX and unavailability of many libraries and dependent software, many features of the application considered for port may not be possible to support. Even though an application's `configure` script examines for the features that can be supported on the target system; it is always a good idea for the programmers to have that knowledge in advance. Samba-3.0.22 cannot support for an ADS client as well as server due to the lack of Kerberos library on MPE/iX. Other major functionalities like WINBINDD, CUPS printing remained unsupported due to lack of NSS, PAM and CUPS libraries. Also, PDC/BDC could not be supported due to the incompatible naming convention between Windows and MPE/iX. Neither did we have any workaround to overcome that incompatibility. Following strategies are applicable for the features which cannot be supported,

a) Don't do anything and by pass that piece of code from compilation and let them remain unsupported, but keep in mind that, doing this should not defeat the objective of the port.
b) Port recursively all the dependencies, or devise some mechanism to make the incompatible behavior compatible.
c) Fake the unsupported low level feature.

3.2 Downloading the vanilla source code

We can download the source code, patches, documentation related to Samba from www.samba.org which is distributed via http and ftp. Every open source portable application has one or more ftp mirror sites from where entire source code along with documentation can be downloaded. The same link can be used to download the latest patch/release. It is recommended not to download any revision without reading the release notes, because there is a high probability that new release may contain bug fixes in those part of the code which MPE/iX by passes. Also before downloading it is important to read the associated notes which may have some legal notices regarding *export outside US and Canada*. We cannot download any package directly to MPE/iX machine unless the host has a direct internet connection to the internet. Download the source code first to any middle man (for example your workstation/windows machine) and then ftp that in binary mode to the target MPE/iX build machine. GNU `wget` (http://jazz.external.hp.com/src/#wget) can also be used for download if your network supports it, which avoids the requirement of an intermediate machine.

Please read the http://us1.samba.org/samba/download/ before downloading any Samba releases. This page contains list of ftp mirror sites and much other important information. It is a good idea to opt for an ftp mirror site geographically near to the download center to allow speedy download.

Note that any downloadable with name length more than 16 cannot be directly downloaded into the MPE groups as files under MPE namespace impose the 16 character length limit. Samba source code package name also exceeds the 16 character limit, so you need to create a directory with name *src* inside /$HPACCOUNT/$HPGROUP as shown below and download and unpack the Samba source code into that directory.

```
MPE/iX: hello mgr.samba,SMB3022
:xeq sh.hpbin.sys –L
Shell/iX> mkdir src
```

Assuming you have the samba source code (e.g smb3022.tar.gz) downloaded into your Windows workstation, following example shows how to ship (PUT) them to your build machine using FTP:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\sagarvi>cd c:\

C:\>cd downloads

C:\Downloads>dir
 Volume in drive C is E-Client
 Volume Serial Number is 88B7-64DC

 Directory of C:\Downloads

05/22/2007  05:15 PM        18,234,944 smb3022.tar.gz
              28 File(s)    221,521,318 bytes
               2 Dir(s)  24,916,275,200 bytes free

C:\Downloads>ftp
ftp> o buildmc.cup.hp.com
Connected to buildmc.cup.hp.com.
220 HP ARPA FTP Server [A0010Q15] (C) Hewlett-Packard Co. 2000 [PASV SUPPORT]
User (buildmc.cup.hp.com:(none)): mgr.samba,smb3022
331 Password required for MGR.SAMBA,SMB3022.  Syntax: [,]acctpass
Password:
230 User logged on
ftp> cd src
250 CWD  file action successful.
ftp> pwd
257-"/SAMBA/SMB3022/src" is the current directory.
257 "MGR.SAMBA,SMB3022" is the current session.
ftp> binary
200 Type set to I.
ftp> put ./smb3022.tar.gz ./smb3022.tar.gz
200 PORT command ok.
150 File: ./smb3022.tar.gz opened; data connection will be opened
226 Transfer complete.
ftp: 18234944 bytes sent in 168.20Seconds 102.03Kbytes/sec.
ftp> bye
221 Server is closing command connection

C:\Downloads>
```

Now you have the tar' ed and compressed samba source code in place to your build machine.
The next step is to unpack them.

3.3 *Unpacking the tarball*

The file that we've downloaded, a tarball, is nothing but the tar 'ed and compressed source code. Unpacking the tarball comprises of two activities; First, to uncompress and then extract the files with tar. Assume that the downloaded Samba source code file name is *samba-3.0.22.tar.gz.* The file can be uncompressed as shown below:

```
Shell/iX> cd /$HPACCOUNT/$HPGROUP/src
Shell/iX> gzip –d samba-3.0.22.tar.gz
```

Now you should have a file named *samba-3.0.22.tar.* Typical uncompressed tarballs are created with relative pathnames and will eventually restore files relative to the current working directory. It is usually a good idea to know in advance what is the content of tarball and that can be achieved by issuing `tar –tvf <package.tar>` which reveals the directory structure to be created and files it contains. Next is to extract the contents of tarball.

```
Shell/iX> tar –xvf samba-3.0.22.tar
```

The directory *samba-3.0.22* just extracted as the result of executing tar command, contains the entire source tree. It is good idea to have a count of all the files inside it, so that you can track the files changed for MPE/iX. Use command `ls –ltR | wc –l` to get the total number of files inside source tree. In order to differentiate between the vanilla and mpe source directory, it is a good practice to maintain two source directories vanialla(original) and mpe(modified source tree for MPE/iX). You should rename the vanilla source directory by appending *–mpe* into that i.e. for Samba rename *samba-3.0.22* to *samba-3.0.22-mpe*. Now the source tree is ready to accommodate the changes with respect to MPE/iX. Later on completion of port you need to evaluate source diff of the vanilla and mpe source directory to ease the future refresh. More about source diff and how to apply them in a port refresh is coming up in next section.

3.4 Applying source patches

Patches are nothing but the “diff” of two source tree. Conventionally a bug fix is released by means of source code patch. Whoever wants to incorporate that fix needs to apply the patch into the already existing source code using well known GNU command `patch`. For the ease of future porters MPE/iX leveraged this strategy.

The first step in building a new release of Samba on MPE is to apply the MPE source diff from the previous version of Samba that is currently running on MPE/iX. The diff file essentially tells how to transform the files in first directory tree to match the files in the second directory tree. The previous version source diff are conventionally named diff-<prev-version>-mpe.txt and placed in /$HPACCOUNT/$HPGROUP/src/ directory.

To modify a vanilla Samba source tree with an MPE diff file, we execute the GNU patch program as shown below:

```
shell/iX> cd mpe-source-dir
shell/iX> /usr/local/bin/patch -p1 -i /path/to/diff.txt
```

That is, in order to build Samba-3.0.22, apply the diff file of last ported version of samba which is *diff-2.2.8a-mpe.txt* as shown below:

```
shell/iX> cd /SAMBA/SMB3022/src/samba-3.0.22-mpe
shell/iX> /usr/local/bin/patch -p1 -i /SAMBA/SMB228A/src/diff-2.2.8a-mpe.txt
```

The patch program will then use the diff file to modify the source files in the vanilla-source-dir directory. The -p1 option says to strip the first pathname component from each entry in the diff file before trying to find a matching file in the current source directory. The -i option gives the path to the input diff file.

The patch program will tell you whether or not each source file referenced in the diff file were successfully modified. For files that cannot be modified, i.e. the new version of Samba is too different from the previous version used to produce the diff file, reject files named **.rej* will be created in the corresponding directory containing the diff modifications that could not be applied.

For each *.rej file you will need to investigate the corresponding source code file and manually make the modifications needed to get that code to run on MPE/iX.

The same procedure can be used to apply a patch (bug fix) released by Samba organization. You should apply the patch on vanilla source tree, followed by applying the bug-fix and finally apply the source diff of previous ported version.
It is encouraged to take a source code diff once you finish the port by the GNU diff program as shown below and keep them in /$HPACCOUNT/$HPGROUP/src directory:

```
shell/iX> /usr/local/bin/diff -ruN samba-vanilla-sourcedir samba-mpe-sourcedir >/path/to/diff.txt
```

For example following steps apply diff for Samba-3.0.22:

```
shell/iX> pwd
/SAMBA/SMB3022/src
shell/iX> /usr/local/bin/diff -ruN samba-3.0.22/source samba-3.0.22-mpe/source
>./diff.txt
```

The diff -r option says to recursively compare two directory trees. The -u option says to generate output in the "unified" format. The -N option says to generate entries to create files that exist only in the second directory tree. The generated `diff.txt` file looks something like this:

```
diff -ruN samba-3.0.22/source/Makefile.in samba-3.0.22-mpe/source/Makefile.in
--- samba-3.0.22/source/Makefile.in Mon Feb 20 13:33:23 2006
+++ samba-3.0.22-mpe/source/Makefile.in      Wed Nov 8 22:49:02 2006
@@ -36,8 +36,12 @@
LDAP_LIBS=@LDAP_LIBS@
INSTALLCMD=@INSTALL@
-INSTALLLIBCMD_SH=@INSTALLLIBCMD_SH@
-INSTALLLIBCMD_A=@INSTALLLIBCMD_A@
```

Since some source files in the Samba distribution are dynamically generated, you will want to remove those entries from the diff file after you have generated it. These files are:

- `source/configure`
- `include/proto.h`
- Few more files which you can find by looking for keyword “creating” in the Samba-3.0.22 build document [8] under “make proto” and “configure” section.

Note that if these files entries are not removed there will be no harm as these files will be recreated while building the application. However it is expected that the diff entries that remain include the files used to dynamically generate the above files.

The same procedure can be used to apply a patch (bug fix) released by Samba organization. You should apply the patch on vanilla source tree, followed by applying the bug-fix and finally apply the source diff of previous ported version.

Once you are done with applying patch and manually modifying the files corresponding to *.rej files, next is to adjust and run the script `configure`.

3.5 Adjusting and Running `configure` script

As we discussed earlier, `configure` is the script responsible for checking system parameters and availability of various assorted routines and libraries. You should note here that `configure` checks for the available supported routines/headers on the host machine (MPE/iX) only by compiling tiny programs that calls/includes the specific routine/header. In MPE/iX many routines are available in header files which are not yet implemented. This script `configure` does not bother about the linker error, eventually reports many routines available and usable, which is incorrect. We will discuss how to overcome this later in this section.

`configure` is generated by GNU *autoconf* from template file `configure.in(configure.ac)`. For some POSIX applications you may find a script `autogen.sh` which invokes *autoconf* which performs checking of required version and other parameters. The Samba package has `autogen.sh` script included and you should run this script after modifying `configure.in` to generate script `configure`.

I would like to remind you that applying the source diff in last step already have modified the `configure.in`. Still a thorough check is useful. Changes in `configure.in` for Samba are as shown below:

```
.
.
.
case "$host_os" in
# MPE/iX needs several defines.
```

```
    *mpeix*)
        export MPEAUTOCONF=1
        CPPFLAGS="$CPPFLAGS -Dmpeix -D_POSIX_SOURCE -D_SOCKET_SOURCE
      -D_INCLUDE_MPEXL_SOURCE -I/SYSLOG/PUB -I/usr/contrib/include"
        LDFLAGS="$LDFLAGS -L/POSIXC60/lib -L/SYSLOG/PUB -Xlinker -WL,xl='/$HPACC
OUNT/$HPGROUP/LIBCPXL'"
        LIBS="$LIBS -lposix60 -lunix -lsyslog -lsocket -lsvipc -lcurses"
        AC_DEFINE(mpeix,1,[Target is MPE/iX])
        AC_DEFINE(_POSIX_SOURCE)
        AC_DEFINE(_SOCKET_SOURCE,1,[Socket source])
        AC_DEFINE(_INCLUDE_MPEXL_SOURCE,1,[Include MPEXL source])
        AC_DEFINE(USE_SETREUID,1)
        AC_DEFINE(_INCLUDE_MPEXL_SOURCE,1,[Include MPEXL source])
        AC_DEFINE(USE_SETREUID,1)
        AC_DEFINE(STAT_STATFS2_BSIZE,1)
        ;;
   .
   .
   .
```

You notice the CPPFLAGS, LIBS, LDFLAGS and various defines are set for MPE/iX specific. If compilation includes creation of shared libraries then one more flag LDSHFLAGS needs to be set appropriately as follows:

```
      LDSHFLAGS = -Wl, -b –nostdlib #Refer the compiler documentation for more
information
```

The Samba/iX has not been compiled to create any shared library; hence this flag has not been set. Apart from setting these parameters, `configure.in` can be hard coded not to build certain features which cannot be supported on MPE/iX. The features that cannot be supported on MPE/iX can also be disabled by passing arguments to configure script. Issue *configure –help* for more details on how not to build certain feature or plugin. I have disabled building of WINBINDD in `configure.in` as follows:

```
.
.
.
AC_MSG_CHECKING(whether to build winbind)
# Initially, the value of $host_os decides whether winbind is supported
HAVE_WINBIND=yes
# Define the winbind shared library name and any specific linker flags
# it needs to be built with.
WINBIND_NSS="nsswitch/libnss_winbind.$SHLIBEXT"
WINBIND_WINS_NSS="nsswitch/libnss_wins.$SHLIBEXT"
WINBIND_NSS_LDSHFLAGS=$LDSHFLAGS
case "$host_os" in
.
.
.
*mpeix*)
                HAVE_WINBIND=no
                winbind_no_reason=", unsupported on $host_os as C library does n
ot support NSS and PAM on MPE/iX"
                ;;
.
.
.
```

Be careful while setting parameters and defines as it should not negatively affect the search of libraries, routines etc. Changes done here governs the generation of `configure` script which in turn generates one or more makefile(s) which finally build your application.

Once `configure.in` is properly adjusted, run `autogen.sh` to generate `configure.`

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source
shell/iX> ./autogen.sh
./autogen.sh: running script/mkversion.sh
./script/mkversion.sh: 'include/version.h' created for Samba("3.0.22")
./autogen.sh: running autoheader
autoheader: `include/config.h.in' is created
./autogen.sh: running autoconf
Now run ./configure.mpe and then make proto and finally make.
```

The `configure.mpe` looks like:

```
shell/iX> cat ./configure.mpe
#!/bin/sh
./configure \
  --prefix=/$HPACCOUNT/$HPGROUP
```

In general the `--prefix` option specifies the base test install directory on build machine. It tells that all the install directories are relative to this directory. By specifying the above mentioned `--prefix` option the Samba package will be installed under /$HPACCOUNT/$HPGROUP directory. This helps managing multiple revisions of same product. Edit this file to pass arguments to script `configure`. See *./configure –-help* for more information.

Finally run the configure script.

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source
shell/iX> ./configure.mpe
configure: loading cache /dev/null
SAMBA VERSION: 3.0.22
checking for -fPIE...
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
.
.
.
checking how to build vfs_catia... not
Using libraries:
    LIBS =  -lposix60 -lunix -lsyslog -lsocket -lsvipc -lcurses
    LDAP_LIBS = -lldap -llber
    AUTH_LIBS =
checking configure summary... yes
configure: creating ./config.status
config.status: creating include/stamp-h
config.status: creating Makefile
config.status: creating script/findsmb
config.status: creating smbadduser
config.status: creating script/gen-8bit-gap.sh
config.status: creating include/config.h
```

It takes almost 5~6 hours on HP e3000 969-400 machine to complete the full configuration for Samba. Upon successful completion of configure, you will find some new Makefile or other modified files for use during build. Analyze the file `config.log` for any errors during configuration. If there is any error, modify `configure.in` to address the error and proceed again by running `autogen.sh`. Keep iteratively doing the above step till error free configuration and note that each iteration takes 5~6 hrs on HP e3000 969-400 machine.

The file `include/config.h` which has been generated by `configure` has lot of defines which controls the source code compilation. Typically it looks like this:

```
/* include/config.h.  Generated by configure.  */
/* include/config.h.in.  Generated from configure.in by autoheader.  */
.
.
.
/* Define to 1 if you have the `shmget' function. */
#define HAVE_SHMGET 1

/* Define to 1 if you have the `shm_open' function. */
/* #undef HAVE_SHM_OPEN */
.
.
.
```

The steps involve in adjusting and running configure are summarized below:

- Edit `configure.in (or configure.ac)` to adjust various flags and defines specific to MPE/iX.
- Edit `configure.in (or configure.ac)` to enable/disable the build of various features/plug-in that are supported/unsupported on MPE/iX.
- Run *autoconf* and *autoheader* either directly or through any vendor script like `autogen.sh` to generate script `configure`.
- Gather arguments to be passed to `configure` and write/edit file `configure.mpe` for maintainability.
- Run `configure` through `configure.mpe`.
- Analyze config.log, newly created Makefile(s), config.h, and other files will be used during build for correctness. Use *ls –ltr* in the source directory to list the files sorted according to modified time and analyze the last few files in the list whose modification date is newer to the time when `configure.mpe` had been invoked.

Now you are ready to attempt build the application on MPE/iX.

3.6 Changing the source code for MPE/iX specific known issues/limitations

One might tempt to start compiling and resolving errors which seems to be a very reasonable approach in the beginning. But, it is always advisable and good idea to make changes the in source code to address known issues before compilation. You may not encounter any error while compiling and definitely will get link or run time error for these limitations. Hence modifying the source code for known limitations in advance would definitely save your time a lot. We already have discussed a few MPE/iX limitations and major issues previously. One should attempt to make changes wherever these limitations apply.

For example below we address the known limit on the size of socket I/O requests by Use of `#ifdef#else#endif` :

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source
```

Code snippet from samba source code:

```
/* File: ./include/local.h */
/* maximum length in bytes of a socket I/O request */
#ifdef mpeix /* MPE limited to 30000 bytes per request */
#define MAX_SOCKET_IO_LEN 30000
```

```
#else
#define MAX_SOCKET_IO_LEN SSIZE_MAX
#endif


/* File: ./lib/system.c */
/****************************************************************
A sendto wrapper that will deal with EINTR.
****************************************************************/

ssize_t sys_sendto(int s,  const void *msg, size_t len, int flags, const struct
sockaddr *to, socklen_t tolen)
{
        ssize_t ret;

        do {
#ifdef mpeix /* Vidya Sagar: Socket Buffer size on MPE/iX is 30000 */
                ret = sendto(s, msg, len > MAX_SOCKET_IO_LEN ? MAX_SOCKET_IO_LEN
:len , flags, to, tolen);
#else
                ret = sendto(s, msg, len, flags, to, tolen);
#endif /* mpeix */
        } while (ret == -1 && errno == EINTR);
        return ret;
}
```

The other known issues/limitations should also be searched and changed with the above mentioned strategy.

3.7 Build Prototypes

In order to incorporate better portability, some portable applications do not include the prototypes of their internal routines into the downloadable source tree. The prototypes (some times called headers) are generated on target machine before final build of the application. Such applications tells programmer through file README or INSTALL how to build prototypes. Samba is one such application. Prototypes are built in Samba by invoking `make proto`in the POSIX shell.

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source
shell/iX> make proto
Making Prototypes...
make delheaders; \
make smbd/build_options.c; \
make include/proto.h; \
make include/build_env.h; \
make include/wrepld_proto.h; \
make nsswitch/winbindd_proto.h; \
.
.
.
creating /SAMBA/SMB3022/src/samba-3.0.22-mpe/source/client/client_proto.h
make[1]: Leaving directory `/SAMBA/SMB3022/src/samba-3.0.22-mpe/source'
make[1]: Entering directory `/SAMBA/SMB3022/src/samba-3.0.22-mpe/source'
creating /SAMBA/SMB3022/src/samba-3.0.22-mpe/source/utils/ntlm_auth_proto.h
make[1]: Leaving directory `/SAMBA/SMB3022/src/samba-3.0.22-mpe/source'
make[1]: Entering directory `/SAMBA/SMB3022/src/samba-3.0.22-mpe/source'
creating /SAMBA/SMB3022/src/samba-3.0.22-mpe/source/utils/net_proto.h
make[1]: Leaving directory `/SAMBA/SMB3022/src/samba-3.0.22-mpe/source'
make[1]: Entering directory `/SAMBA/SMB3022/src/samba-3.0.22-mpe/source'
creating /SAMBA/SMB3022/src/samba-3.0.22-mpe/source/utils/passwd_proto.h
make[1]: Leaving directory `/SAMBA/SMB3022/src/samba-3.0.22-mpe/source'
```

Please note that many header files have been created. After running `configure` and building prototypes its time now to build your application.

3.8 Build (make and make install)

Some portable applications have only one Makefile at the root of source tree and some have many Makefiles each in a subdirectory which contain rules to build code in that particular subdirectory. In the latter case, Makefile(s) call each other in nested fashion. Samba has a single Makefile located inside root source directory (/SAMBA/SMB3022/src/samba-3.0.22-mpe/source). A full build of Samba on MPE/iX takes significant amount of time (We observed ~5-6 hrs on an HP e3000 969-400 system in our HP Lab environment).

Starting a Samba build can be started by just issuing command `make` at POSIX shell as follows:

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source
shell/iX> make
Using FLAGS =  -O -D_SAMBA_BUILD_ -I./popt -Iinclude -I/SAMBA/SMB3022/src/samba-
3.0.22-mpe/source/include -I/SAMBA/SMB3022/src/samba-3.0.22-mpe/source/ubiqx -I/
SAMBA/SMB3022/src/samba-3.0.22-mpe/source/tdb  -I. -Dmpeix -D_POSIX_SOURCE -D_SO
CKET_SOURCE -D_INCLUDE_MPEXL_SOURCE -I/SYSLOG/PUB -I/usr/contrib/include -DLDAP_
DEPRECATED -I/SAMBA/SMB3022/src/samba-3.0.22-mpe/source -D_SAMBA_BUILD_
      LIBS = -lposix60 -lunix -lsyslog -lsocket -lsvipc -lcurses
      LDSHFLAGS = -shared  -L/POSIXC60/lib -L/SYSLOG/PUB -Xlinker -WL,xl='/SAMBA
/SMB3022/LIBCPXL'
      LDFLAGS = -L/POSIXC60/lib -L/SYSLOG/PUB -Xlinker -WL,xl='/SAMBA/SMB3022/LI
BCPXL'
      PIE_CFLAGS =
      PIE_LDFLAGS =
Compiling dynconfig.c
Compiling param/loadparm.c
Compiling param/params.c
Compiling smbd/files.c
Compiling smbd/chgpasswd.c
Compiling smbd/connection.c
Compiling smbd/utmp.c
Compiling smbd/session.c
.
.
.
Compiling lib/smbldap_util.c with
Compiling groupdb/mapping.c with
Linking libsmbclient non-shared library bin/libsmbclient.a

175 OBJECT FILES HAVE BEEN ADDED.

Compiling libsmb/smb_share_modes.c with
Linking libsmbsharemodes non-shared library bin/libsmbsharemodes.a

3 OBJECT FILES HAVE BEEN ADDED.

Compiling libmsrpc/libmsrpc.c with
Compiling libmsrpc/libmsrpc_internal.c with
Compiling libmsrpc/cac_lsarpc.c with
Compiling libmsrpc/cac_winreg.c with
Compiling libmsrpc/cac_samr.c with
Compiling libmsrpc/cac_svcctl.c with
Linking libmsrpc shared library bin/libmsrpc.so
Linking libmsrpc non-shared library bin/libmsrpc.a

181 OBJECT FILES HAVE BEEN ADDED.
```

If you see both the “linking libmsrpc” and “nnn OBJECT FILES HAVE BEEN ADDED” then the Samba application has been built successfully and is ready to be installed. The moment `make` encounters any error it stops there with error message on the STDOUT. If you see any error while building, just look into the corresponding file and do the necessary modification. Refer the section common problems and their resolutions for more information. After doing the appropriate changes make starts building from the last successfully built point.

It is always a good idea to maintain a log file of `make` output for future references or to resolve compilation errors. I recommend you use the shell command `tee` to redirect display into a file as well as to the standard output. For example:

```
shell/iX> make | tee ./make.log
```

Now that you have successfully built the application, it’s time to install them into the test directory. Hopefully you remember that the `--prefix` option of script `configure` sets the install directory as discussed in the adjusting and running configure section. So in order to complete the test install on build machine all we need to do is to issue command `make install` at POSIX shell:

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source
shell/iX> make install
Using FLAGS =  -O -D_SAMBA_BUILD_ -I./popt -Iinclude -I/SAMBA/SMB3022/src/samba-
3.0.22-mpe/source/include -I/SAMBA/SMB3022/src/samba-3.0.22-mpe/source/ubiqx -I/
SAMBA/SMB3022/src/samba-3.0.22-mpe/source/tdb  -I. -Dmpeix -D_POSIX_SOURCE -D_SO
CKET_SOURCE -D_INCLUDE_MPEXL_SOURCE -I/SYSLOG/PUB -I/usr/contrib/include -DLDAP_
DEPRECATED -I/SAMBA/SMB3022/src/samba-3.0.22-mpe/source -D_SAMBA_BUILD_
      LIBS = -lposix60 -lunix -lsyslog -lsocket -lsvipc -lcurses
      LDSHFLAGS = -shared  -L/POSIXC60/lib -L/SYSLOG/PUB -Xlinker -WL,xl='/SAMBA
/SMB3022/LIBCPXL'
      LDFLAGS = -L/POSIXC60/lib -L/SYSLOG/PUB -Xlinker -WL,xl='/SAMBA/SMB3022/LI
BCPXL'
      PIE_CFLAGS =
      PIE_LDFLAGS =
Installing bin/smbd as /SAMBA/SMB3022/sbin/smbd
Installing bin/nmbd as /SAMBA/SMB3022/sbin/nmbd
Installing bin/swat as /SAMBA/SMB3022/sbin/swat
======================================================================
The binaries are installed. You may restore the old binaries (if there
were any) using the command "make revert". You may uninstall the binaries
using the command "make uninstallbin" or "make uninstall" to uninstall
binaries, man pages and shell scripts.
======================================================================
Installing bin/smbclient as /SAMBA/SMB3022/bin/smbclient
Installing bin/net as /SAMBA/SMB3022/bin/net
Installing bin/smbspool as /SAMBA/SMB3022/bin/smbspool
Installing bin/testparm as /SAMBA/SMB3022/bin/testparm
Installing bin/smbstatus as /SAMBA/SMB3022/bin/smbstatus
.
.
.
/usr/local/bin/install -c bin/libmsrpc.so /SAMBA/SMB3022/lib
/usr/local/bin/install -c bin/libmsrpc.a /SAMBA/SMB3022/lib
/usr/local/bin/install -c /SAMBA/SMB3022/src/samba-3.0.22-
mpe/source/include/libmsrpc.h /SAMBA/SMB3022/include
```

The binaries, libraries and documentation including manpages are installed in the install directory. Some binaries require special capabilities to run properly on MPE/iX. The next subsection describes how to assign the essential capabilities to a binary.

3.9 Setting proper Capabilities and ownership

As we discussed earlier some binaries (smbd, nmbd & swat installed in /$HPACCOUNT/$HPGROUP/sbin) require that they be run as super user that is UID should be set to 0. In MPE/iX that is achieved by allowing the code to run into privileged mode by invoking MPE/iX intrinsic GETPRIVMODE(). In order to GETPRIVEMODE, the binary must be linked with the capability PM and must be moved into the MPE file system name space (that is /ACCOUNT/GROUP/PROGRAM or PROGRAM.GROUP.ACCOUNT). Also, the ownership of these files should be changed to MANAGER.SYS. The following script performs the job of putting the essential binaries into MPE/iX name space and assigns proper capability for Samba.

```
shell/iX> cat ~/mpebin/progs
#!/bin/sh

cd /$HPACCOUNT/$HPGROUP/sbin

for FILE in *; do
  UPPER=$(echo $FILE | tr [a-z] [A-Z])
  if [ ! -L $FILE ]; then
    mv -f $FILE ../$UPPER
    ln -s ../$UPPER $FILE
  fi
  callci "xeq linkedit.pub.sys 'altprog ../$UPPER;cap=ia,ba,ph,pm'"
  chown MANAGER.SYS ../$UPPER
done
```

Invoke the above script to perform the same as follows:

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source
shell/iX> sh ~/mpebin/progs
shell/iX> sh ~/mpebin/progs
HP Link Editor/iX (HP30315A.06.24) Copyright Hewlett-Packard Co 1986

LinkEd> altprog ../NMBD;cap=ia,ba,ph,pm

LinkEd> altprog ../SMBD;cap=ia,ba,ph,pm

LinkEd> altprog ../SWAT;cap=ia,ba,ph,pm
```

You can verify the changed ownership and modified capability and as follows:

```
shell/iX> ls -l NMBD SMBD SWAT
-rwxr-xr-x   1 MANAGER.SYS         SAMBA    2537216 Jul  4 05:30 NMBD
-rwxr-xr-x   1 MANAGER.SYS         SAMBA    5799168 Jul  4 05:30 SMBD
-rwxr-xr-x   1 MANAGER.SYS         SAMBA    3835904 Jul  4 05:30 SWAT

shell/iX> exit

MPE/iX: linkedit "listprog nmbd.smb3022.samba"
HP Link Editor/iX (HP30315A.06.17) Copyright Hewlett-Packard Co 1986

LinkEd> listprog nmbd.smb3022.samba

PROGRAM        : NMBD.SMB3022.SAMBA
XL LIST        : /SAMBA/SMB3022/LIBCPXL
CAPABILITIES   : BA, IA, PM, PH
NMHEAP SIZE    :
NMSTACK SIZE   :
ENTRY NAME     :
UNSAT NAME     :
PRIORITY       :
MAX PRIORITY   :
```

```
POSIX          : YES
SHARED DATA    : YES
TEXT SIZE      : 000F8C10
DATA SIZE      : 0003B000
VERSION        : 85082112

MPE/iX: :linkedit "listprog smbd.smb3022.samba"
HP Link Editor/iX (HP30315A.06.17) Copyright Hewlett-Packard Co 1986

LinkEd> listprog smbd.smb3022.samba

PROGRAM        : SMBD.SMB3022.SAMBA
XL LIST        : /SAMBA/SMB3022/LIBCPXL
CAPABILITIES   : BA, IA, PM, PH
NMHEAP SIZE    :
NMSTACK SIZE   :
ENTRY NAME     :
UNSAT NAME     :
PRIORITY       :
MAX PRIORITY   :
POSIX          : YES
SHARED DATA    : YES
TEXT SIZE      : 002BB170
DATA SIZE      : 00095000
VERSION        : 85082112

MPE/iX: linkedit "listprog swat.smb3022.samba"
HP Link Editor/iX (HP30315A.06.17) Copyright Hewlett-Packard Co 1986

LinkEd> listprog swat.smb3022.samba

PROGRAM        : SWAT.SMB3022.SAMBA
XL LIST        : /SAMBA/SMB3022/LIBCPXL
CAPABILITIES   : BA, IA, PM, PH
NMHEAP SIZE    :
NMSTACK SIZE   :
ENTRY NAME     :
UNSAT NAME     :
PRIORITY       :
MAX PRIORITY   :
POSIX          : YES
SHARED DATA    : YES
TEXT SIZE      : 001A7390
DATA SIZE      : 00054000
```

Once the capabilities are set and ownership is changed we need to check for unresolved routines as discussed in the next subsection before start testing.

3.10 To check for the unresolved symbols

Unresolved externals are routines whose definition is not found by the system loader while loading the program into memory for execution. Normally the GNU ld linker returns a linker error if any unresolved routines are encountered during linking unlike the MPE/iX LINKEDIT linker which does not report any error while linking (because MPE/iX allows all system calls and those to user supplied libraries (NL, XLs and SLs) to be resolved at run time). On MPE/iX, the loader will not load the program into memory if any unresolved routines are encountered and reports error. Hence, it is a good idea to know and resolve those routines before actually running the program. Run the following command to list the unresolved externals into the MPE/iX CI:

```
:RUN theprogram;STDIN=*notfound > tempfile
```

Where “theprogram” is the newly linked program, “notfound” is any :FILE equation name that has not been defined, and “tempfile” is a temporary file where any :RUN error messages will be written.

If the program contains unresolved external references, the :RUN command will fail, and the list of unresolved symbols will be written to the temporary file “tempfile”. But if the program does not contain any unresolved symbols, the :RUN will still fail because the STDIN :FILE equation does not exist. It is important to note that under no circumstances will the program actually execute as a result of this special :RUN command.

One question that may also be asked is if an unresolved routine is called by functions that are part of the “core” functions which you care about or perhaps (hopefully) by functions you care less about. A methodology is to :RUN program;UNSAT=DEBUG – this will run the program and the invoke the MPE system debugger when an routine that cannot be found in the SL, XL, or NL files is actually called. This may allow to more quickly focus on those routines which are likely to cause a problem. There are some complexities with this method, especially on applications which fork() many child processes; also typically this may not work with exec() children. See [1] for more information.

3.11 Strategies to resolve unresolved symbols

The following strategies can be adopted to resolve the problem of unresolved symbols:

- Undef the routine switch: Some applications are coded with the knowledge that certain system functions may or may not be present. In this case the source will have conditional compiler directives which can be set to indicate whether or not this function is available and the application will then be compiled in the manner necessary to account for this the lack of this functionality. For example Samba defines HAVE_USLEEP to use usleep() if it exists on the system. However MPE/iX header have the declaration of usleep() but left unimplemented. It can be removed from compilation as shown below:

```
/* MPE/iX lacks usleep */
#ifdef mpeix
#undef HAVE_USLEEP
#endif
```

Some applications are not set-up to use the “undef” method when accessing system functions and therefore these functions may be viewed as mandatory for the application Several methods may be used.

- Leverage routine definition: Attempt to copy the function from another source. For example in this Samba port, MPE/iX lacks definition of strptime(). I leveraged the code (strptime.c) form latest libc source code can be downloaded from http://www.gnu.org/software/libc/libc.html and include the source for the strptime() into the samba source tree into source/lib directory.

- Fake the unresolved routine: telldir() and seekdir() are two routines unimplemented on MPE/iX and used by Samba. These two routines allow offset based directory traversal by program if present. Otherwise, the calling program uses “name” based directory traversal, i.e., directory access key can be file/directory name instead of their offset. The lib/replace.c in Samba source tree contains definition for them as described in MPE/iX limitations and major issues.

  You can acquire similar approach for these types of routines.

- Use some other routines instead of the unresolved routines: Due to continue evolution in POSIX interfaces many POSIX.1 compliant routines are deprecated, hence a more “modern” application typically avoids using them. Samba uses random() and srandom() which are unimplemented on MPE/iX. In this case use rand() instead as shown below.

```
#ifdef mpeix
#define random rand
#define srandom rand
#endif
```

  Another similar instance is where Samba uses ldap_unbind_ext() because ldap_unbind_s() is deprecated. Hence we replaced the call of ldap_unbind_ext() with ldap_unbind_s() as shown below.

```
#ifdef mpeix
                ldap_unbind_s(ldap_state->ldap_struct);
#else
                ldap_unbind_ext(ldap_state->ldap_struct, NULL, NULL);
#endif
```

If you encounter unresolved symbols which do not seem to be covered by any of the above mentioned class, either refer common runtime problems for other suggestions.

Once all the unresolved routines are resolved rebuild the application as mentioned in step 8 and 9. You now should have an application that is built, installed on the build machine and ready to test.

3.12 Testing the features to be supported (make test)

In section 3.1 you would have created a list of features your application supports. Unit test all those features and document the results, especially if you found any limitation during testing. One must also decide if this is an expected outcome to be documented, or a bug which needs to be fixed before you proceed. Many applications contain standard tests to be done after successfully building the code on target machine. Identify the tests applicable to your port and run `make test` to carry out those tests.

```
shell/iX> pwd
/SAMBA/SMB3022/src/samba-3.0.22-mpe/source #Just for example

shell/iX> make test #not applicable to Samba
```

After running those tests analyze the test results against expected outcomes. Standard tests available in Samba source tree do not apply to MPE/iX Samba as test scripts are written for Unix environments, hence you can skip running standard test i.e. no need to run `make test`. For HP Lab engineers a list of standard tests should be performed for Samba/iX which is available at http://jazz.external.hp.com/src/samba/samba-3.0.22-test.htm.

3.13 Packaging

Packaging the binaries and essential source for distribution is solely the porter's responsibility. While creating a distribution package you must keep in mind that it should not include any extraneous file, any experimental file, any temporary file etc which increases the deliverable size. Also, it should not miss any binary, library, header file, manpages. You should also include any script can be useful in running the application. Typically only a diff of vanilla and mpe source should be distributed, it is not recommended to include the entire source as that can be available from the application's official site. The package must have a script to install it on target machine. The installation script must check and create the proper account structure if absent, sets the proper capability to the user, fixes the binary and directory permission and finally cleanup, i.e., purge the temporary created files/directory during installation.

Normally HP distributes ported applications for MPE/iX as patches which can be installed using PATCHIX or AUTOPAT. Their distribution requires minimally following three files

- Program file P00xxxx which is the package for distribution.
- IHFxxxx file is the installation job file which is streamed by patchix or autopat to install the patch.
- README file REAxxxx contains installation notes.

where xxxx is the last four characters of patchid.

If you are packaging a patch for use with HP installation tools you should ensure that the job IHFxxxx invoke the installation script. Appendix A lists the script to package Samba and few important notes on patch creation.

3.14 Documentation notes

Given the limitations of the MPE/iX POSIX implementation it is possible to successfully port a useful percentage of an application's feature set. However it is quite normal that features a user may be looking for remains unsupported or supported with changed behavior. One must document those and advertise them in such a way that user should not miss them. Please refer Samba-3.0.22 communicator article [7] on Jazz which lists significant limitations of Samba-3.0.22 on MPE/iX. There are similar articles on Jazz for other ported products.

3.15 Performance Notes

You may encounter noticeable performance degradation on running some ported application on MPE/iX and some may give good performance also. One reason behind performance degradation I believe might be due to some extra overhead introduced as a result of implementation/simulation of POSIX APIs on MPE/iX. Another is that MPE/iX Process creation is relatively more expensive than on UNIX systems.

Hence it is always a good idea to tune your application to adapt more into the MPE/iX environment. Please remember that MPE/iX just simulates the POSIX environment and internally uses native calls for handling resources. A few performance tuning tips for Samba can be found in communicator article of Samba-3.0.22 [7] on Jazz.

This finishes the porting of Samba-3.0.22 on MPE/iX. Your package is ready for installation on MPE/iX systems!

It is our hope that other open source freeware POSIX.1 compliant application can be successfully ported on MPE/iX by leveraging the concepts used for Samba porting. The next section discusses few common problems and some idea how to resolve them which may be helpful during some other port. You can also refer few other successful ports on MPE/iX listed in Appendix J as source of inspiration to do a fresh port or to refresh an existing port.

4 Common problems and their resolutions

This section is a general discussion of problems you may face during various phases of porting any POSIX based application to MPE. We focus on a few selected areas which have caused problems for others in previous porting attempts. This section complements the section "MPE/iX limitations and major issues."

4.1 Common Include File Problems

If a compile attempt dies because MPE is missing an include file expected by the application, first try to correct the problem by modifying the application to NOT include the file on MPE. If the application does not use a configure script, use the following technique to omit the include file on MPE:

```
#ifndef MPE
#  include <foobar.h>
#endif
```

If the application uses GNU autoconf, examine configure.in to see if the header file in question is being checked for existence, and if is it not, add the appropriate macro language to perform such checking. For example if CPPFLAG does not include "-I/SYSLOG/PUB", the script `configure` reports unavailability of file "syslog.h". Refer section adjusting and running configure [3.5] on how to set CPPFLAG. Then modify the failing source code like this:

```
#ifdef HAVE_FOOBAR_H
#  include <foobar.h>
#endif
```

These two techniques are the most typical ways of solving most compile time problems, not just include file problems. If the application uses GNU autoconf, modify the autoconf scripts to check for all of the functionality the application needs. Otherwise, use the compiler's implicit OS-based #define symbols in conjuction with #ifdef/#ifndef to generate the correct source code.

Note that sometimes a hybrid approach is required, i.e.:

```
#ifdef HAVE_FOOBAR_H
#  include <foobar.h>
#else
#  ifdef MPE
/* do something MPE-specific */
#  endif
#endif
```

But what if it's not possible to compile with the missing include file? Check to see if the file is provided via the Porting Wrappers (/POSIXC60/include or Appendix H) or in /usr/contrib/include, and if it is, specify either -I/POSIXC60/include or -I/usr/contrib/include (but preferably not both) in the CPPFLAGS list of precompiler options.

If the include file does not exist in the Porting Wrappers or in /usr/contrib/include, and you are getting undefined symbol compile errors due to your inability to include this file, examine this include file on an HP-UX or Linux system to see what the correct symbol definitions might be. Then copy just what you need into the source file you are porting, i.e.:

```
#ifdef HAVE_FOOBAR_H
#  include <foobar.h>
#else
#  ifdef MPE
#    define symbol1 value1
#    define symbol2 value2
#    define symbol3 value3
#  endif
#endif
```

Alternatively, you could copy an entire include file or set of include files from HP-UX or Linux, and store them in your own directory structure that mirrors the regular system include root directory, i.e.:

/MYACCT/MYGROUP/src/myapp/mpeinclude/arpa
/MYACCT/MYGROUP/src/myapp/mpeinclude/net
/MYACCT/MYGROUP/src/myapp/mpeinclude/sys

Then modify the compiler flags in CPPFLAGS to also include:

-I/MYACCT/MYGROUP/src/myapp/mpeinclude

Then when the source code does something like:

#include <foobar.h>

The compiler will look for /MYACCT/MYGROUP/src/myapp/mpeinclude/foobar.h before checking the default system include directories. One important drawback to this approach is that very few open source applications are structured this way, so you may have trouble submitting your porting changes back to the original developers.

Some time-related symbols that a real Unix system would define in <sys/time.h> are instead defined in MPE's <time.h>. Don't ask me why. Since <sys/time.h> does not exist on MPE, replace references to it with <time.h>.

4.2 Common Compile-Time Problems

Make sure your source files (and *.h header files) are bytestream files. Files that are MPE Fixed-length ASCII will cause gcc to return strange errors about premature EOFs. This is a well-known side-effect of MPE's bytestream emulator, which also impacts other POSIX apps like Apache and Samba.

The gcc compiler can generate a mind-numbing amount of warnings and yet still complete the compile. The compiler will make certain assumptions for certain warnings, and sometimes it can guess wrong in a way that will prevent your application from running properly. For example, due to available telldir() and seekdir() declaration compiler assumes they are available on MPE/iX however the same is not.

Nearly all POSIX applications on MPE require the use of the compiler options -D_POSIX_SOURCE and -D_SOCKET_SOURCE in CPPFLAGS as we did in section adjusting and running configure [3.5]. If you are getting lots of compiler errors about undefined types and macros, try defining these compile symbols first before doing any other investigation.

If the compile error is related to something that an autoconfigure script checked for, verify that the script came up with the correct answer and that the source code actually relies on this answer to determine what to do. We did not encounter this error, hopefully you won't either.

Many socket applications use PF_INET and other PF_xxx macros instead of AF_INET and corresponding macros on MPE. The simplest way to deal with this is to modify your application's source code to do:

#define PF_INET AF_INET

4.3 Common Run-Time Problems

Bytestream vs. MPE Fixed ASCII
Just like you should always use bytestream files to avoid compiler problems, you should also use bytestream files to avoid run-time problems with your own application. An application that calls stat() to determine the EOF of an MPE fixed ASCII file will get a value greater than the number of bytes the application would sequentially read from the file due to the way the MPE bytestream emulator does not present trailing spaces to POSIX applications read from such files.

fcntl() vs sfcntl()
MPE has two different versions of the POSIX fcntl() function -- one named sfcntl() that you must use on sockets, and the traditional one named fcntl() that you use on all other file types. Sometimes it may be easiest to examine an application and look for fcntl() calls that apply only to sockets and then replace those calls with sfcntl(). But at other times, the same fcntl() call may be in a general purpose function that could operate both on sockets and regular files. Therefore a run-time method is required to determine whether to call fcntl() or sfcntl(). The Porting Wrappers has a solution for this problem -- see /POSIXC60/posix.1_defects/fcntl/fcntl.c for details. Section [2.1.4] discusses how to install porting wrappers and Appendix H contains the list of files.

Unresolved Externals
Nearly every porting attempt will experience multiple unresolved external references the first time the application is run after a clean compile. Your first step in troubleshooting these errors is to verify that you are linking with the correct *.a archive libraries (NMRLs) and/or *.sl (or *.so) shared libraries (NMXLs). The standard POSIX-related libraries on MPE reside in /lib and /usr/lib. If you need to link with one of these system libraries, specify it via the -l option of ld (note that gcc will pass-through linker options to ld if you are using gcc to link).

If the unresolved symbols are not part of the standard POSIX or sockets APIs, then you may need to link with a non-system library. Specify a library path via the ld -L option. Subsequent -l options will then search the preceding -L directories looking for the library. Set LDFLAGS and LIBS switches as discussed in section adjusting and running configure [3.5].

Socket Binding
Note that when doing socket programming, you must call GETPRIVMODE() before calling bind() to ports less than 1024. Something else to be aware of with bind() is that you can only bind to the wildcard address (0.0.0.0, meaning all network interfaces) or the loopback address (127.0.0.1) on MPE. Binding to a specific non-wildcard, non-loopback address will fail. The general workaround here is to bind to the 0.0.0.0 wildcard address instead. But note that this will listen on ALL network interfaces, so if an e3000 has multiple network interfaces, this behavior could in theory be problematic if you did not really want the application to listen on certain network interfaces.

Uninitialized struct passwd Fields
The POSIX getpwxxx() functions do not initialize the /usr/include/pwd.h fields labeled as "the following fields are uninitialized until further notice". That means the char * fields will contain garbage for the pointer addresses, so if you attempt to dereference these fields, your program will either abort or retrieve random data.

Streams/iX (pipes) Hangs
POSIX pipes are implemented via Streams/iX. Over the years Streams/iX has had a number of bugs that can result in system crashes or application hangs for programs that use POSIX pipes. Streams/iX hangs will be

readily apparent from the stack traces of the afflicted process(es). Installing the latest Streams/iX patch for your MPE release is always a good idea before you start your porting activities.

Hard Links vs. Symbolic Links
The MPE HFS file system does not support hard links. It is frequently acceptable to modify applications to create symbolic links instead.

SVIPC Resources
Note that MPE does not release SVIPC semaphore or shared memory resources when processes normally or abnormally terminate. Usually a well-behaved application will free these resources prior to termination. But while you are debugging your port for the first time, you may experience process aborts which don't free your SVIPC resources. There is a limited pool of these resources, and over time, repeated aborts can exhaust the pool. When pool exhaustion happens, you can either reboot the system, or use the IPCS.HPBIN.SYS, IPCRM.HPBIN.SYS, and SVIPC.HPBIN.SYS utilities to free up resources. Note that these utilities in HPBIN.SYS are actually CI command files.

UID 0 Superusers
One very key difference between Unix and MPE is that MPE doesn't implement the Unix concept of a superuser as defined when the POSIX userid (UID) is equal to zero. The Unix root concept is roughly analogous to the MPE concept of SM capability and/or PM capability. Some Unix applications check the current userid of the process and reserve certain functionality for root users only. Other Unix applications may want to change their UID to zero in order to gain read/write access to files that would be protected from normal users, or to be able to invoke certain Unix functionality reserved only for root users, like bind()-ing to ports less than 1024. If your application depends heavily on UID zero (i.e. superusers), you may have quite a porting challenge ahead of you. Both Samba and Sendmail are heavily dependent on UID zero, and large and complicated porting solutions had to be developed to simulate much of the UID zero functionality that would exist on a real Unix system.

UID and GID Issues
On MPE, a UID is an integer number corresponding to an MPE user object, and a GID (Group ID) is an integer number corresponding to an MPE account object. The MPE implementation of POSIX UIDs and GIDs requires that your UID and GID both refer to the same MPE account. Unix allows a user to belong to multiple groups, but on MPE a user can only belong to one group (that corresponds to the MPE account).

Terminal I/O
Support for POSIX terminal I/O (sometimes called the General Terminal Interface or GTI) was never fully implemented in MPE. However, some terminal interfaces are provided in <termios.h>. Note that you will find those functions in the -lcurses library /usr/lib/libcurses.a.

Child Processes Killed When Parent Terminates
As mentioned in section 1.4 a typical Unix server application architecture may call fork() to spawn child processes (daemons) that will continue running even if the parent process terminates. However when a process terminates on MPE, all child processes of the terminating process will also be terminated. So, when porting applications with such parent/child architectures to MPE, one make sure the parent process does not terminate if child processes are required by the application to function.

5 References

[1] MPE/iX Developer's Kit Reference Manual Volume 1 (36430-90001) (http:/docs.hp.com/en/36430-90007/index.html)

[2] MPE/iX Developer's Kit Reference Manual Volume 2 (36430-90002) (http://docs.hp.com/en/36430-90008/index.html)

[3] Mark Bixby's Porting guide (http://www.bixby.org/mark/porting.html)

[4] Pretty Good porting for MPE/iX (http://www.editcorp.com/Personal/Lars_Appel/pgp-ix.html)

[5] Implementation details of POSIX on MPE/iX (http://www.docs.hp.com/cgi-bin/doc3k/B3021690178.13563/42)

[6] How to make POSIX work on MPE (http://www.3000newswire.com/FNPlug-N-Play.html)

[7] Samba-3.0.22 communicator article (http://jazz.external.hp.com/papers/Communicator/7.5/Samba_3.0.22.html)

[8] Samba-3.0.22 build document (http://jazz.external.hp.com/src/samba/BuildingSamba-3.0.22.htm)

6 Appendix

A. Packaging and Patch Notes for Samba

The following information references systems inside HP for use by MPE Lab persons when building a patch for distribution to HP customers; it is included here for completeness.

The script to package Samba is available at http://knowmpe.cup.hp.com/krt-docs/Network/ARPAInternetServices/Samba/makedist.txt. This script bundle samba and creates a file for distribution with name HFSFILES. Rename this file to P00xxxx while copying it to LPATCH for patch activity. The files included in Samba package is listed at http://knowmpe.cup.hp.com/krt-docs/Network/ARPAInternetServices/Samba/File-list-package.txt. Refer the steps listed at http://knowmpe.cup.hp.com/krt-docs/Network/ARPAInternetServices/Samba/Patch_notes.txt to create a patch for delivering it to customer.

B. Installation Notes (Autopat and Patch/iX)

Autopat fails to install with the custom install file (IHF files) that does not contain ";outclass=,1" in their job (!job abcd, manager.sys) line however patch/ix succeeds without any error.

C. Script to install GNU gcc compiler and utility (INSTALL.gcc)

```
#!/bin/sh
################################################################
# File: INSTALL.gcc
# Author: Vidya Sagar (vidya.sagar2@hp.com)
# Purpose: To install the GNU gcc 3.4.2 compilers and assorted tools
# Assumption: The file all.bin or file[1-9].bin has been downloaded from JAZZ and
resides in/tmp
# Note: Rerun this script every time you install or update from a FOS or CSLT tape
################################################################
function exit_on_error
{
        if [ $? != 0 ]; then
        echo Exiting due to error
        exit 1
        fi
}

exit_on_error
cd /tmp
if [ -f all.bin ]; then
        echo Renaming all.bin to GNU.Z...
        mv all.bin GNU.Z
else
        NO_OF_FILE_BIN=`ls file[1-9].bin | wc -l`
        echo Total number of files downloaded are $NO_OF_FILE_BIN
        if [ $NO_OF_FILE_BIN -le 9 ]; then
                yesno=N
                echo Files are `ls file[1-9].bin` "[correct? (Y/N)]: "
                read yesno
                if [ "$yesno" = Y ]; then
                        echo Okay..Continuing....
                        cat file[1-9].bin > GNU.Z
                else
                        echo Download all the files "file[1-9].bin" or all.bin and
try again
                        exit 0
```

```
                fi
        else
                echo Download all the files "file[1-9].bin" or all.bin and try
again
                exit 0
        fi
fi
exit_on_error

echo Installing...Please wait...
echo Installing...Please wait...
compress -d GNU.Z
exit_on_error

frombyte -b GNU GNU1
exit_on_error

callci 'file GNU1=/tmp/GNU1;dev=disc'
callci 'restore *GNU1;/;tree;create;show'
exit_on_error

if [ -f /usr/local/INSTALL.hp3000 ]; then
        /usr/local/INSTALL.hp3000
        exit_on_error
        echo "***********Installation complete "":)*************"
else
        echo /usr/local/INSTALL.hp3000 does not exist
        exit_on_error
fi
```

D. Script to install POSIX wrappers (INSTALL.px_wrappers)

```
#!/bin/sh
#################################################################
# File: INSTALL.px_wrappers
# Author: Vidya Sagar (vidya.sagar2@hp.com)
# Purpose: To install the POSIX wrapper into /POSIXC60 directory
# Assumption: The file px_wrappers.tar.tar or px_wrappers_run.tar.tar has been
downloaded
# from JAZZ and resides in/tmp
#################################################################
# Rename the downloaded file to posixc60.tar.Z
cd /tmp
if [ -f px_wrappers.tar.tar ]; then
        mv px_wrappers.tar.tar posixc60.tar.Z
elif [ ! -f posixc60.tar.Z ]; then
        echo Download posix wrapper package and try again
        exit 1
fi
#Create account to put POSIX wrappers:
#We have already logged on as system manager and into the POSIX shell
callci newacct POSIXC60,MGR
cd /POSIXC60
tar -xvozf /tmp/posixc60.tar.Z PUB/INSTALL
callci install.pub.POSIXC60
tar -xvozf /tmp/posixc60.tar.Z
#Hope everything went fine
if [ $? != 0 ]; then
        echo Error: try again
else
        echo Done ":)"
```

```
fi
```

E. Script to install BSD library (INSTALL.libbsd)

```
#!/bin/sh
################################################################
# File: INSTALL.libbsd
# Author: Vidya Sagar (vidya.sagar2@hp.com)
# Purpose: To install the BSD lib (libbsd.a) into /usr/lib
# Assumption: The file libbsd.tar.tar has been downloaded from JAZZ and resides
in/tmp
################################################################

#Proceed only if system has /usr/lib dir
if [ -d /usr/lib ]; then
        cd /tmp
        if [ -f libbsd.tar.tar ]; then
                mv libbsd.tar.tar libbsd.tar.Z
        fi
        if [ ! -f libbsd.tar.Z ]; then
                echo Error: Download the libbsd package and try again
                exit 1
        fi
        if [ ! -d /usr/include/bsd ]; then
                mkdir /usr/include/bsd
        fi
        if [ $? != 0 ] ; then
                echo Error: try again
                exit 1
        else
                cd /usr/include/bsd
                tar -xvofz /tmp/libbsd.tar.Z
                echo Done ":)"
        fi
else
        echo "Error: /usr/lib does not exist"
```

F. Script to patch LIBCPXL (patch-libcpxl)

```
#!/bin/sh
# no-op the getpw() function in /POSIXC60/HPBIN/LIBCPXL
cd /$HPACCOUNT/$HPGROUP
rm –f  LIBCPXL
cp /POSIXC60/HPBIN/LIBCPXL LIBCPXL
cat <<EOF | /SYS/PUB/SOMPATCH
open LIBCPXL.$HPGROUP.$HPACCOUNT
modify getpw+8,2 6bda3db9| e8000940 6bd93db1| 0800025c
exit
EOF
if [ $? != 0]; then
   echo Done “:)”
fi
```

G. libbsd header files

```
/usr/include/bsd/a.out.h

/usr/include/bsd/ar.h

/usr/include/bsd/db.h
```

```
/usr/include/bsd/dirent.h
/usr/include/bsd/errno.h
/usr/include/bsd/fcntl.h
/usr/include/bsd/ftw.h
/usr/include/bsd/glob.h
/usr/include/bsd/mnttab.h
/usr/include/bsd/mpe.h
/usr/include/bsd/mpool.h
/usr/include/bsd/ndbm.h
/usr/include/bsd/nl_types.h
/usr/include/bsd/nlist.h
/usr/include/bsd/paths.h
/usr/include/bsd/pwd.h
/usr/include/bsd/resolv.h
/usr/include/bsd/sgtty.h
/usr/include/bsd/signal.h
/usr/include/bsd/stdio.h
/usr/include/bsd/stdlib.h
/usr/include/bsd/string.h
/usr/include/bsd/strings.h
/usr/include/bsd/sysexits.h
/usr/include/bsd/syslog.h
/usr/include/bsd/termios.h
/usr/include/bsd/time.h
/usr/include/bsd/tzfile.h
/usr/include/bsd/unistd.h
/usr/include/bsd/utmp.h
/usr/include/bsd/arpa/ftp.h
/usr/include/bsd/arpa/nameser.h
/usr/include/bsd/arpa/telnet.h
/usr/include/bsd/arpa/tftp.h
/usr/include/bsd/machine/endian.h
/usr/include/bsd/net/if.h
/usr/include/bsd/net/if_arp.h
/usr/include/bsd/netinet/in.h
/usr/include/bsd/netinet/in_systm.h
/usr/include/bsd/netinet/ip.h
/usr/include/bsd/netinet/ip_icmp.h
/usr/include/bsd/netinet/ip_var.h
/usr/include/bsd/protocols/rwhod.h
```

/usr/include/bsd/protocols/talkd.h
/usr/include/bsd/rpc/rpc.h
/usr/include/bsd/sys/cdefs.h
/usr/include/bsd/sys/dir.h
/usr/include/bsd/sys/errno.h
/usr/include/bsd/sys/file.h
/usr/include/bsd/sys/ioctl.h
/usr/include/bsd/sys/mman.h
/usr/include/bsd/sys/param.h
/usr/include/bsd/sys/resource.h
/usr/include/bsd/sys/signal.h
/usr/include/bsd/sys/socket.h
/usr/include/bsd/sys/stat.h
/usr/include/bsd/sys/statfs.h
/usr/include/bsd/sys/stdsyms.h
/usr/include/bsd/sys/syslog.h
/usr/include/bsd/sys/time.h
/usr/include/bsd/sys/ttychars.h
/usr/include/bsd/sys/wait.h

H. POSIX wrapper header files

/POSIXC60/include/dlfcn.h
/POSIXC60/include/errno.h
/POSIXC60/include/errnongd.h
/POSIXC60/include/fcntl.h
/POSIXC60/include/general.h
/POSIXC60/include/grp.h
/POSIXC60/include/langinfo.h
/POSIXC60/include/libgen.h
/POSIXC60/include/libposix.h
/POSIXC60/include/mpe2.h
/POSIXC60/include/mpeaif.h
/POSIXC60/include/mpeerrno.h
/POSIXC60/include/mpestub.h
/POSIXC60/include/mpesubsys.h
/POSIXC60/include/netdb.h
/POSIXC60/include/ourhdr.h
/POSIXC60/include/setsigint.h
/POSIXC60/include/signal.h
/POSIXC60/include/string.h

```
/POSIXC60/include/stringpac.h
/POSIXC60/include/syslog.h
/POSIXC60/include/tcmpe.h
/POSIXC60/include/termtype.h
/POSIXC60/include/turboimage.h
/POSIXC60/include/ulimit.h
/POSIXC60/include/unistd.h
/POSIXC60/include/util.h
/POSIXC60/include/sys/mman.h
/POSIXC60/include/sys/param.h
/POSIXC60/include/sys/socket.h
/POSIXC60/include/sys/stat.h
/POSIXC60/include/sys/stdsyms.h
/POSIXC60/include/sys/time.h
/POSIXC60/include/sys/uio.h
/POSIXC60/include/sys/utsname.h
/POSIXC60/include/sys/wait.h
```

I. Using Other Ported Applications for Inspiration

It's unlikely that very many unique new porting problems will arise at this stage in MPE's lifecycle. Therefore when faced with a porting problem in some new application, it's quite likely that the solution you are seeking has already been implemented in earlier ported applications. What follows is a high-level overview of some of the significant porting solutions in various applications (note that this is not a complete list of applications or solutions).

Apache

- an implementation of the Unix dlopen()/dlsym()/dlclose()/dlerror() functions used to load modules from shared libraries (NMXLs)
- use of Mark Klein's longpointer support for passing 64-bit long pointers to MPE intrinsics

Network Time Protocol

- an implementation of the Unix adjtime() clock adjustment function
- a partial implementation of the Unix setitimer() interval timer function
- priv-mode implementation of the Unix gettimeofday() function that returns microsecond timestamps

Perl (PERL account on invent3k.external.hp.com)

- More examples of using Mark Klein's longpointer support to pass 64-bit pointers to MPE intrinsics
- Support for dynamically loading Perl modules from NMXLs

Samba

- elaborate UID/GID emulation to support the illusion of UID 0 superusers on MPE
- strptime() leveraged from libc

Sendmail (SENDMAIL account on mpesourc.cup.hp.com)

- another elaborate UID/GID emulation to support the illusion of UID 0 superusers on MPE