Port-a-kit p-System

Knowledge Software Ltd
Farnborough, Hants, England

Chapter 1

Introduction

The basic aim is to implement a p-System Port-a-kit. This kit provides all of the tools needed to port the p-System to a new host processor (provided a reasonable ‘C’ compiler is available).

This portability has been achieved by writing a P-code interpreter in ‘C’. Also provided is a bios, written in ‘C’, containing all the hooks needed to get the p-System up and running on a new host processor.

The p-System Port-a-kit will run existing software unchanged, provided that it does not use assembly language.

The Port-a-kit is aimed at mini and mainframe computers. It is expected that users will require multi-user facilities. It is assumed that the host operating systems will be commercially oriented.

1.1 This document

This document is a guide for bringing the Port-a-kit up on a new processor or operating system.

1.2 Other documents

Port-a-kit source code. This is heavily commented and obviously provides the detailed documentation.

Internal Architecture Reference Manual (IARM). The definitive guide to P-code definitions. Also provides a definition of the basic functions of the bios.

Adaptable p-System guide.

Stride 400 series owners manual Vol 1 & 2. A good source for ideas on how to extend the standard p-System.

Chapter 2

The distribution disc

2.1 P-code interpreter

- P-code interpreter in ‘C’. All features found in a standard IV.2.1 interpreter plus the extensions documented here.
- Bios in ‘C’ with well defined hooks to allow rapid porting to new hosts.
- LONGOPS unit in ‘C’

2.2 Support Tools

a) PDIR. Manipulate p-System directory from host OS. Export/Import files, create p-System volumes, etc.

b) Symbolic debugger. See page 11

c) MC. A utility to deduce the runtime properties of C compilers.

2.3 Directory listing

```
       Volume in drive C has no label
 Directory of  C:\PORTKIT

MC       C      10996   2-13-89   9:46a
PDIR     C      12963   1-05-89  10:20a
PMEDEF   H      18971   2-20-89   3:20p
PMEOPS   H       9725  12-14-88   4:06p
PMEMAC   H       7741  12-14-88   4:06p
PMEVAR   H       7107   2-14-89  11:42a
PMEDBG   C      23084   2-14-89  12:00p
PMEFEC   C      15120  12-14-88   3:56p
PMEIO    C      22012   1-04-89   5:35p
PMEMCD   H        951   2-20-89  11:40a
PMERW    C      24180   2-14-89   2:50p
PMESTD   C      14713  12-14-88   3:53p
PMENAT   C       6468  12-14-88   3:54p
PCODE    DAT     2824   3-26-87   2:33p
PMEMC    H      14752   2-20-89  11:41a
ERRNO    H        326  12-14-88   4:08p
PMECAL   C      10631  12-14-88   3:49p
MCH          <DIR>      9-24-88   5:58p
PMELNG   C      21486  12-14-88   3:51p
```

```
PMELUT   C       8396  12-14-88   3:51p
PMETSK   C       5597  12-14-88   3:52p
PMECOD   C      19739  12-14-88   3:52p
PSCVT    C       2340   1-04-89   5:36p
PMEUTL   C      22860   2-20-89  10:55a
PSYS     C      19813   2-14-89  11:42a
AMOSMCD  H        980  12-14-88   4:06p
TERMST   C        281  12-14-88   3:49p
PSAMOS   C       2578  12-14-88   3:56p
PSEMA    C      18672   1-05-89   4:55p
PSUTIL   C       6710   2-20-89   1:21p
PSEMA    H        873   2-14-89  10:18a
       31 File(s)    708608 bytes free
```

Chapter 3

Bringing up the Port-a-kit

Before attempting to install the Port-a-kit it is recommended that you obtain copies of the appropriate ‘C’ language users guide and ‘C’ runtime library reference manual.

3.1 MC

This is a utility that deduces the runtime implementation dependent features of your ‘C’ compiler. When executed MC produces a file containing macros specific to a given C compiler on a particular host cpu.

MC generates a file called PMEMCDF.H which is *#include*d into PMEMC.H and contains the following fields:

- *Host* - name of host system as supplied to MC
- *HostSex* - 0 or 1 (byte index of msb in *WORD*)
- *WORD* - 16 bit signed type
- *UWORD* - 16 bit unsigned type
- *LSx()* - logical left shift macros
- *RSx()* - logical right shift macros
- *BYTE* - 8 bit type
- *UBW()* - convert *BYTE* to *UWORD*
- *SBW()* - convert *BYTE* to *WORD*
- *no_real/RealSize/REAL* - define characteristics of floating-point type to be used
- *StreamIO* - select file I/O method. See page 28

The following are a list of non-runtime specific *#define*s . These are all in PMEMC.H:

- *has_ioctl, has_signal, has_profile* - these control the inclusion of certain system header files and the code associated with them. See page 26 for details of what these mean in Unix-like environments.
- *M_PROC* - taken from the field in KERNEL, this should indicate the host processor type if machine code is to be supported. See page 29 for more details.

- *NAT68000* - example of support for native code on 68000 based hosts.
- *REG1-9* - register variables in order of priority. Define as many to *register* as your compiler allows, define the rest as white space. See page 12 for further discussion.
- *void* - define this as *int* if the host compiler does not have the *void* type.
- *HostFNameLen* - length of maximum host file name. A reasonable value is *128.*
- *DefaultSysDisk* - host file name to use when no boot volume is given. e.g., "psystem.vol"
- *baby_compiler* - define this if the P-code fetch loop in PMEFEC.C is too large for the compiler to handle. This selects whether the file PMECAL.C is *#include*d into PMEFEC.C or is treated as a separately compiled file. Also certain macros become function calls to reduce code size.
- *PTReq(), PTRlt()* - pointer comparisons

  PTRsub() - difference between two pointers. The result should be a *WORD* which is usually interpreted as an unsigned displacement, however there are a few places where this result is treated as signed.

  PTRdisp() - add an unsigned displacement to a pointer. e.g., *PTRdisp(p,-2)* must be equal to *p+0xfffeL*.
- *GETCH* - read a character from the keyboard **without** echoing it.

  PUTCH() - write a character to the terminal.

  CharWaiting - should return *FALSE* if no characters are waiting to be read from the keyboard, *TRUE* if one or more characters are waiting.

  CBufSize - size of internal type-ahead buffer.

  CEOFChar - if *GETCH* can return *-1* on any input character, this defines what p-System sees.

  IsEOF - code to test for end-of-file condition on keyboard after error on *GETCH*. Define this as *FALSE* if not applicable.

  ReadString - usually just *gets*, this macro is used by PMEDBG to read from the console (with echo). On hosts that use *ioctl* something more elaborate needs to be done to save and restore the state of the terminal across *gets*.

  See page 7 for more details of these macros and how they affect the function of the Bios.
- *BlockOffset()* - convert a p-System block number into an offset for *fseek/lseek*. The result must be a correct *long* value.
- Various macros under *StreamIO* - check that these definitions correspond to the host's way of doing file I/O.
- *LowClock/HighClock* - return the low and high *WORD*s of a 32-bit, 60Hz system clock. If the host does not support a clock, return *0*. It may be necessary to convert the

host clock to 60Hz to simulate this.

- *GetPDate* - return a 16 bit value representing the date as follows:
  RSn(year 0..99, 9) | RS4(day of month 1..31) | month 1..12
  If the date is not available, return a suitable base date such as 1st Jan 1980 i.e., *0xA011*.

3.2 Interpreter

To bring up the interpreter compile the following:

- Main interpreter files:

  PME.C - main, bootstrap

  PMEFEC.C - fetch-execute loop

  PMECOD.C - complex P-codes

  PMESTD.C - standard procedures (UNITREAD etc)

  PMETSK.C - task switching, signal, events

  PMEUTL.C - various routines used by the Port-a-kit

  PMEIO.C - the input/output routines (RSP/IO and BIOS)

  PMERW.C - the UnitRW functions

  PMELNG.C - main LONGOPS routine

  PMELUT.C - LONGOPS utilities

  PMENAT.C - native code and relocation stubs: must be present even if native code is not supported.

- If *DEBUG* is *#define*d in PMEDEF.H

  PMEDBG.C - interactive debugger

- If *baby_compiler* is *#define*d in PMEMC.H

  PMECAL.C - the rest of PMEFEC

- If *stride_semaphores* is defined in PSEMA.H

  PSEMA.C - support for global semaphores

Link them together as PSYS (for example). See page 23 for details of running PSYS and options available.

3.3 RSP/IO and Bios

The Port-a-kit has combined the various functions of the RSP/IO, bios and SBios into one file called PMEIO.C.

3.3.1 Bios

There are four macros that are bios related and must be supplied for the new host:

GETCH — This macro should expand to 'C' code that reads one character from the keyboard **without** echoing it.

If it is necessary to switch the terminal into half duplex this should be added to *pme_exit()* in PMEUTL.C.

PUTCH(c) — This macro should expand to 'C' code that outputs the character *c* to the terminal. This routine is used to echo input as well as normal p-System terminal output.

While this could be done more portably by using *putchar(c),* it was felt that performance was an issue here and that if a faster system call could be used, it would be better.

PUTCH can be *#define*d to be *putchar* without affecting the functionality of the Port-a-kit bios.

Some output buffering can be handled by the Port-a-kit to improve performance. See the definition of BPUTCH in PMERW.C for an example of how this can be done.

CharWaiting — This macro should expand to 'C' code that returns *TRUE* if a character has been typed on the keyboard, otherwise *FALSE*.

If it is not possible to ascertain whether or not unread characters have been typed at the keyboard *FALSE* should be returned.

This macro controls the ability to buffer input and handle special p-System keys (such as Stop/Start, Flush and Break). If CharWaiting cannot be implemented the Port-a-kit will not be able to correctly process any special characters.

CBufSize — This is the number of characters that can be buffered internally by the Port-a-kit. If CharWaiting is not available this macro may as well be set to *0*.

Characters received after the buffer has become full are thrown away (the bell is rung for every character typed when this has happened). A reasonable size for this buffer is 32 characters.

3.3.2 Special Key Processing

The p-System has its own interpretation of certain keys. This section looks at keys that may cause problems and discusses possible solutions.

- ETX or ^C is the 'accept' key in the p-System editor. It may be the host interrupt key.

  The best solution is to *#define* GETCH in such a way that ^C is passed to the p-System without causing an interrupt, or to disable ^C altogether (in *TertiaryBootStrap()* and enable it in *pme_exit()*)

  If disabling ^C causes it to disappear, rather than be read in, it may also be necessary to change ETX under p-System using the SETUP utility.

- End-Of-File, typically ^D or ^Z. This key should be passed intact to the p-System.

  If the host end-of-file character causes *GETCH* to return *-1*, *CEOFChar* should be the value of that character. The Port-a-kit bios will automatically translate this.

  Ideally, GETCH should return all characters untouched, including end-of-file.

- BREAK or ^@ as the p-System Break key. Because it is ASCII NUL some hosts do not pass it back through a read.

  Ideally, GETCH should pass ^@ back untouched.

  If the host does not return ^@, SETUP should be used to change the Break key.

  On some hosts, this is the interrupt key. Here, either GETCH must be able to return it (as for ETX above) or else the Break key must be changed.

- p-System Break. The Break condition is tested by the routine that buffers the console input. The buffering routine is called from various points in the PME: at CONSOLE/SYSTERM I/O, at jumps and at calls/returns. This checking slows the interpreter down and can be switched off by *#undef*'ing PSBREAK in PMEDEF.H (see 'Tuning the Port-a-kit').

- Stop/Start or ^S/^Q. This is usually the host stop-output key.

  If the host handles ^S/^Q correctly, there is no reason why this function should not be left to the host OS. However, if it is required that the PME handles it, the following options are suggested:

  If GETCH can return ^S untouched, the PME will handle it.

  If ^S is handled by the host, some combinations of ^S with Break and Flush may not behave identically to p-System. i.e., it may be necessary to press ^Q before p-System responds again, whereas Break should cancel ^S and Break immediately.

- Flush output ^F. If GETCH cannot return this character untouched, use SETUP to change the Flush key to something else.

3.3.3 Standard Bios Entry Points

The following list gives the all the bios entry points assumed to exist under native p-System implementations, and indicates where in the Port-a-kit the equivalent code, if any, can be found.

Because the RSP/IO and bios have been more or less combined, the equivalent of one bios call may be handled in part by one routine and in part by another, with the objective being to make the source more readable, portable and still flexible.

SYSINIT — The first routine called when the p-System is bootstrapped. In a native environment it is used to initialize the hardware.

SYSHALT — Called when the p-System terminates through a Ⓗ(alt. Achieved by issuing unitread(0), in the Port-a-kit this simply calls *pme_exit()*.

CONINIT — *UnitClear(1, ...);*

CONSTAT — *UnitStatus(1, ...);*

Can only return useful information if *CharWaiting* is implemented and internal character buffering is used.

CONREAD — *UnitRead(1, ...);*

Also see the routines *ubrc(), ubrcn(), ubrs(), ubrsn()* in PMERW.C and the macros *GETCH, CharWaiting* in PMEMC.H, *ReadCh* in PMEMAC.H and the routine *ChGet* in PMEIO.C.

CONWRIT — *UnitWrite(1, ...);*

Also see the routines *ubwc(), ubwcn()* in PMERW.C and the macro *PUTCH* in PMEMC.H.

SETDISK — Used to hold the current disk number in a native p-System environment.

Not applicable in the Port-a-kit, since the level of i/o is much higher.

SETTRAK — Used to hold the current track number in a native p-System environment.

Not applicable in the Port-a-kit, since the level of i/o is much higher.

SETSECT — Used to hold the current sector number in a native p-System environment.

Not applicable in the Port-a-kit, since the level of i/o is much higher.

SETBUFR — Used to hold the buffer used to hold data read/written to/from disk in a native p-System environment.

Not applicable in the Port-a-kit, since the level of i/o is much higher.

DSKREAD — *UnitRead();*

DSKWRIT — *UnitWrite();*

DSKINIT	In a native p-System environment this resets the current disk.
	Not applicable in the Port-a-kit, since the level of i/o is much higher.
DSKSTRT	In a native p-System environment this starts the current disk. Used in the days before more sophisticated floppy controllers were available.
	Not applicable in the Port-a-kit, since the level of i/o is much higher.
DSKSTOP	In a native p-System environment this stops the current disk. Used in the days before more sophisticated floppy controllers were available.
	Not applicable in the Port-a-kit, since the level of i/o is much higher.
PRNINIT	*UnitClear(6, ...);*
PRNSTAT	*UnitStatus(6, ...);*
	Some mechanism must be available on the host to establish the status of the printer device being used for this to provide any useful information.
PRNREAD	*UnitRead(6, ...);*
	Calls *ubr(), ubrn()* to read the printer as a serial device. The behaviour is undefined if the attached printer does not allow a read operation.
PRNWRIT	*UnitWrite(6, ...);*
	Calls *ubw(), ubwn()* to write to the printer as a serial device.
REMINIT	*UnitClear(7, ...);*
REMSTAT	*UnitStatus(7, ...);*
	As for PRNSTAT above, if the host does not allow you to establish such status information, the Port-a-kit cannot return any useful information.
REMREAD	*UnitRead(7, ...);*
	Calls *ubr(), ubrn().*
REMWRIT	*UnitWrite(8, ...);*
	Calls *ubw(), ubwn().*
SERINIT	*UnitClear();*
SERSTAT	*UnitStatus();*
	The same applies as for REMSTAT and PRNSTAT above.
SERREAD	*UnitRead();*
	Calls *ubr(), ubrn()* on the required serial device.
SERWRIT	*UnitWrite();*
	Calls *ubw(), ubwn()* on the required serial device.

CLKREAD	Called from *UnitStatus(0, ...);*, this is simulated in the Port-a-kit with the macros *LowClock, HighClock* in PMEMC.H.
QUIET	See *Quiet* in PMEMAC.H. An interrupt mask is incremented, this mask should be tested by any interrupt routine and the interrupt ignored if the mask is non-zero.
ENABLE	See *Enable* in PMEMAC.H. This is the inverse of *Quiet*.

3.4 Debugging tools

To enable the PME interactive debugger, *#define* DEBUG in PMEDEF.H. This tool allows P-codes to be single stepped (and disassembled), breakpoints to be set on various events (such as encountering a particular P-code, entering a particular segment or procedure within a segment or reaching a given offset within a procedure, when a location changes or changes to a given value), inspection (and modification) of main memory and so on. (Typing **?** in the debugger lists the commands available. More documentation on the debugger will be available shortly)

3.5 LONGOPS

The Port-a-kit includes ‘C’ code to handle those arithmetic operations performed by the p-System LONGOPS unit that are written in assembly code. In order for the Port-a-kit to intercept calls to LONGOPS procedure 2 (the only one written in assembler and emulated by the Port-a-kit) the LONGOPS code segment must be placed in SYSTEM.PASCAL.

During the p-System boot, Port-a-kit searches SYSTEM.PASCAL for the LONGOPS unit and remembers its position in the environment. Calls to that unit may then be trapped with the minimum of overhead.

3.6 REALOPS

The representation of reals in the Port-a-kit is determined by the host ‘C’ compiler. This format may not necessarily be IEEE format.

In those cases where the native ‘C’ compiler uses more than 32 bits, but less than 64 bits for representing reals the following solution is suggested: Configure the p-System for double precision reals but only use the single precision host format.

The p-System unit REALOPS assumes IEEE format with a specific word (16 bit) ordering.

3.7 SYSTEM.MISCINFO

The format of the SYSTEM.MISCINFO file is byte sex dependent.

Thus before booting p-System on a new machine make sure that the SYSTEM.MISCINFO you are using has the correct byte sex.

Chapter 4

Tuning the Port-a-kit

4.1 Registers

'C' provides a mechanism to allow the programmer to indicate which variables are frequently used. The reserved word *register* is prefixed to a declaration. It is compiler specific as to how many, if any, register variables are actually kept in registers. The Port-a-kit defines the macros REG1, REG2..REG9. Consult your 'C' user guide to find out how many register variables are effective. Edit the file PMEMC.H to define the appropriate number of macros as register, the remainder are defined as null.

4.2 Assembler Code

It is unlikely that your 'C' compiler generates the best possible machine code from the Port-a-kit source. Most compilers can produce assembler output. This assembler in turn can be assembled. There are a few critical areas where hand tuning this assembler could improve performance.

a) A surprising amount of time is spent fetching and decoding the next instruction.

   The largest saving to be made in hand tuning the Port-a-kit is in this area. Typically your 'C' compiler will generate code to check the ranges on the *switch* expression. These can be removed. Also the *switch* jump table can be moved, if it is not already there, to the head of the *switch*.

b) The *OPCODE* macro (in PMEFEC.C) allows for the situation where the p-code can potentially be held in a *register* variable.

   If the fetch/exec loop has been hand tuned with the *#define tweaked* flag, the p-code should be in some fixed *register*. Thus it is possible to further enhance the fetch/exec loop code (as above) to speed up all the short load/store p-codes that use *OPCODE*.

   Note that if the assembler is not being hand tuned, the extra code created by the assignment to *count* in the *tweaked* version of *OPCODE* could actually slow the interpreter down, depending on how well your compiler handles *register* variables.

4.3 p-System break key

If the name PSBREAK is *#define*d in PMEDEF.H, all jumps, calls and returns check for console input. This allows the p-System break to be detected. If the Break facility is not required, PSBREAK should be *#undef*'d. There is something like a 10% performance penalty in enabling break.

4.4 Profiling and debugging

The names DEBUG and STATS are used to switch internal debugging and profiling on/off.

Some 'C' compilers insert code at the head of each function to check that stack overflow is not about to occur. This overhead should be avoided if possible, by compiling the interpreter without stack checking.

Profiling is possible with some 'C' compilers. This can be useful in determining which areas of the Port-a-kit would benefit from tuning. See page 12 for comments on the likely areas and how to improve them.

Chapter 5

Advanced Bios

5.1 Events

The routine *Event()* can be called by any event handler to signal any associated p-System semaphore.

An example is the timer event in PMETSK.C (which only works if the library routines *signal* and *alarm* are supported).

To be expanded…

5.2 Special calls to Unitread/Unitwrite

The Port-a-kit supports the following special calls:

UnitWrite(133, filename, 0, unit, access);
Open host file on given p-System **unit**. **access** is made up of the values 1 for reading, 2 for writing and 4 for re-use existing file (when write bit set).

UnitWrite(134, dummy, 0, unit, 0);
Close host file attached to given p-System **unit**.

UnitRead(135, buffer, len, unit, 0);
Read **len** bytes from host file attached to **unit**. If **IOResult** is 20 (read past end of file), call **UnitStatus(135, stat, 0);** to find out how many bytes were actually read (in **stat[0]**).

UnitWrite(135, buffer, len, unit, 0);
Write **len** bytes from **buffer** to host file on given **unit**.

UnitRead(136, dummy, blk, unit, offset);
Perform a *seek* to given **blk** and **offset** within host file attached to **unit**. The actual argument to *seek* is *pBlkSize* ∗ *blk* + *offset* where *pBlkSize* is 512 (i.e., a p-System file block).

UnitWrite(137, buffer, len, 0, 0);
Set up the I/O buffer to be used on subsequent calls to Import/Export (see below). The **buffer** must be at least 3k long as it is used to hold the p-System directory (2k) and buffer the file during transfer. The larger the better.

UnitRead(138, options, 0, unit, 0);
Import host file to p-System volume open on **unit**. The **options** field is identical to the '-i' option on the Pdir utility: hostname,psysname,type (see page 30 for more details of Pdir).

UnitWrite(139, options, 0, unit, 0);
Export p-System file from volume open on **unit** to given host file. The **options** field is

identical to the '-e' option on the Pdir utility: psysname,hostname (see page 30 for more details of Pdir).

The **filename** and **options** fields must be **STRING** type.

5.3 Multi-User facilties

In a multi-user environment there are two main concerns:

1. There must be a safe method of sharing resources. A resource might be a database, line printer, etc.
2. As more users gain access to a particular machine it is likely that the response time will increase. To reduce this loading the application developer should have access to tools that allow programs to be configured or configure themselves.

5.4 Global/Stride semaphores

These are not connected with p-System semaphores in any way. Rather they are a method of testing and setting flags accessible to two independent p-Systems under the hosted Operating System.

As the name suggests the implementation is compatible with the calling sequence defined for Stride computers.

The basic primitives are:

- Get control of one or more semaphores.
- Release control of one or more semaphores
- Clear one or more semaphores
- Check semaphore for existence and/or controlling user.
- Obtain statistics on all global semaphores.
- Clear all semaphores.

In order to implement this extension to p-System some form of shared memory must be supported by the host operating system.

5.4.1 Using the semaphores

All semaphore operations are performed using unitread/write p-System unit 132 (the same as Stride):

```
Unitwrite(132, sema, num_sems, sem_op, 1)
Unitread(132, sema, num_sems, sem_op, 1)
```

Where the parameter are:

132	The unit number used to identify semaphore operations.
sema	A data buffer containing one or more semaphores.
num_sems	The number of semaphores contained in *sema*.
sem_op	The operation to be performed.
1	This value must be given.

The semaphore data structure is an 18 byte area. The first 2 bytes are used to hold the users task number and are filled in by the system. The remaining 16 bytes are available for programmer use.

```
Sem_Type = Record
           Task_Num  :Integer;    (* Filled in by the system *)
           Sem_Info  :Packed Array[1..16] of Char
           End;
```

Release control

A write request with a a *sem_ops* of 0 (zero) releases control of one or more semaphores.

It is possible to release control of a semaphore by a user for which control was not originally obtained.

IORESULT error codes:

0	Control of all requested semaphores was released.
2	Feature not supported.

Get control

A write request with a *sem_ops* of 1 gives control of one or more semaphores. If control cannot be obtained for all of the semaphores, control will not be granted for any of them. In this case the request must be repeated.

IORESULT error codes:

0	Control of all requested semaphores was obtained.
1	Control of the semaphores was rejected.
2	Feature not supported.
3	Not enough room in the semaphore table.

Check sempahores

A write request with a *sem_ops* of 2 checks on the existence of one or more semaphores.

IORESULT error codes:

0 The semaphores exist and are all under the control of the calling user.

1 At least one of the semaphores does not exist in the semaphore table.

2 Feature not supported.

3 The semaphores exist but not all are are under the control of the calling user.

Clear users semaphores

A write request with a *sem_ops* of 3 clears all the semaphores associated with the calling user.

Read back semaphores

A read request with a a *sem_ops* of 4 reads all of the active semaphores in the system's semaphore table. The *num_sems* field must specify the maximum number of semaphores that can be held in the callers buffer.

The list will be terminated with a semaphore who's task number is zero.

IORESULT error codes:

0 Information was returned.

2 Feature not implemented.

Get semaphore statistics

A read request with a a *sem_ops* of 5 reads the following information into the data buffer.

Data buffer offset:

0 Maximum number of system semaphores.

2 Number of active semaphores.

4 Calling user's task number.

IORESULT error codes:

0 Statistics were returned.

2 Feature not implemented.

Clear all semaphores

A write request with a block number of 6 clears the system semaphore table. All entries are cleared for all users.

IORESULT error codes:

0 Clearing was performed.

2 Feature not implemented.

5.5 Releasing a users timeslice

The call:

```
unitwrite(132, dummy, 0, 0, 2)
```

causes the users current timeslice to be released.

Programs attempting to gain access to a group of semaphores should release their timeslice if a semaphore operation fails to grant access; and then try again.

5.6 PSUTIL

This utility read and manipulates the system data file used by the global/Stride semaphores.

-q Display information about the current configuration.

-r Reset the semaphore handling for all users. Warning: this is a dangerous option and should only be used if, for any reason, the semaphores have become locked up.

-s Recreates the system semaphore data file to handle a different number of semaphores. The file should be created in a system directory to which all users have access.

```
usage: psutil <options>
 where <options> are:
        -q       query p-System semaphore configuration
        -r       reset p-System semaphores
        -s<num>  set number of p-System semaphores
```

5.7 Inter-User communication

The ability for different users being able to send messages to each other is not usually provided by operating systems.

It would be possible to implement this feature via shared files.

Chapter 6

Optional tools

The following tools are not supplied as a standard part of the Port-a-kit. For further details of implementation on your host cpu please contact Knowledge Software.

6.1 POPTYSER(Optimiser)

This works at the P-code level and can be supplied on a MS-DOS or Unix disc for porting to the new host.

6.2 Native code generator

The POP-NCG system provides a toolbox and framework for providing optimizing native code generators.

A typical estimate for producing a version of POP-NCG targetted to a new cpu is 4-5 months.

6.3 System Assembler

It would be possible to produce a version of the p-System assembler.

6.4 Turtlegraphics

It is not anticipated that this option will be required. Given demand Turtlegraphics could be implemented.

Chapter 7

Mini/Mainframe considerations

Most large computers have evolved over a period of many years. They are invariably oriented towards handling large numbers of users and batch processing. There are thus a number of areas where they differ in performance/structure from micro-computers.

- Usually have a low relative cpu performance.
- I/O handled by specialized hardware units.
- Single character I/O usually very inefficient.
- Block serial I/O usually well supported.
- 64K restrictions. Some Operating Systems/cpus have code/data size restrictions of this magnitude.

These considerations have been taken into account in the design and implementation of the interpreter.

7.1 Shared code

Most modern Operating Systems allow the total size of running programs to be larger than the available memory. This is achieved by swapping code and data to/from memory and disc. To reduce the space taken up by programs some Operating Systems allow code to be shared between users. Thus if more than one user is running a particular program there only need be one copy of that program in memory. This sharing cuts down swapping and thus improves overall system performance.

The ability to share Port-a-kit program code will be available on many machines. This sharing usually requires the intervention of a system administrator to configure the sharing. Check with your ‘C’ compiler guide and Systems Administrator for the availability of Port-a-kit code sharing.

Chapter 8

Portability considerations

All computers are different. This is true across manufacturers machines and ranges of machines from the same manufacturer. An efficient P-code interpreter providing multi-user facilities is not going to be 100% portable.

Experience with porting other software between different environments has shown that it can be done. The price paid is a slower running program with few interfaces to its host.

There are two conflicts of interest:

1. The P-code interpreter must be as fast as possible. It must also be portable to a wide range of machines.
2. Application developers, under user pressure, want to provide a good interface to the host environment. All hosts are different.

There are three basic obstacles to portability:

1. 'C' compilers. Contrary to popular belief 'C' is not a very portable language. Its philosophy of allowing programmers to get close to the target machine coupled with the lack of a definitive standard has lead to a wide variety of compilers. Compilers differ in the facilities offered. Also there are some quite fundamental decisions that are specified as being implementation dependent, in the language definition.
2. Underlying hardware. This affects the way the 'C' compiler handles features and could impact the P-machine's view of the world.
3. Facilities provided by the host operating system. The facilities required to run a basic p-System under a given operating system are virtually assured. However, the more sophisticated features will rely on the host operating system providing the necessary features.

8.1 'C' compilers

Previous experience with 'C' compilers has already suggested a number of areas where caution is required. As the Port-a-kit is compiled with new 'C' compilers other portability problems will be uncovered.

The 'C' preprocessor provides a method of configuring the source to be handled by a variety of compilers. Those areas containing constructs with known portability problems

are defined as macros. Conditional compilation then selects the appropriate definition.

8.2 The underlying hardware

There are five basic issues:

1. The method of representing integers. The P-machine definition specifies two's complement. There is no intrinsic reason why one's complement should not work provided no 'dirty tricks' are used.
2. Size of Integers. The P-machine assumes an integer size of 16 bits. Host processors may have more than one integer size provided that one of them is 16 bits long. Processors having a larger minimum integer size may present severe portability problems. For instance a machine having a minimum integer size of 24 bits could be handled provided it used word addressing, but not if it used byte addressing.
3. Size of reals. The P-machine assumes that single precision reals will fit in 32 bits and double precision in 64 bits. If single precision reals occupied more than 32 bits but less than 64 bits it would be possible to treat single precision reals as double. This would rule out the use of double precision.
4. Byte sex. This issue has been thoroughly handled by the P-machine. Care has been be exercised in the design and implementation of the P-code interpreter to ensure no byte sex dependencies creep in.
5. Byte/Word addressing. With one exception all processors currently running the p-System are byte addressed. It is understood that with a few modifications the p-System can be made to run on a word addressed processor. Very little experience has been gained running the p-System on word addressed processors and so any potential problems are unknown.

8.3 Implementing Portability

The only reliable way of producing portable software is to take into account non-portable issues and design the software accordingly.

During development the Port-a-kit was regularly processed by a variety of 'C' compilers. Two of these compilers were Intel 8086 based and two Motorola 68000 based. This ensured that no compiler or byte sex dependencies appeared in the design or implementation.

Chapter 9

Configuration Options

The p-System itself has various internal options that can be configured and the interface to the host Operating System will need to be easily modifiable.

To start the interpreter, execute

```
PSYS volname
```

where *volname* is the host name of a p-System bootable volume. If *volname* is omitted, PSYS will try to boot from the file specified by *DefaultSysDisk* in PMEMC.H

Various configuration options may be specified after *filename*. This will be dependent on the host Operating System having the ability of passing parameters to a program when it is executed. The following section detail these optional parameters (if available).

9.1 Code pool size

When booting the Port-a-kit uses the following rules to determine the code pool size.

1. If a **-c** option is specified that value is used (see below).
2. SYSTEM.MISCINFO is examined and its code pool field definitions (internal or external and size) are used.

```
PSYS volname -c<size>
```

allocates <*size*>k bytes for the external code pool. If <*size*> is zero, p-System will boot with an internal code pool. If <*size*> is omitted, a default of 85 is used - this is a good general size for the external code pool.

If PSYS is unable to allocate enough memory to satisfy the request, it successively reduces the code pool size until it succeeds or the size becomes less than 36, in which case an

internal code pool is used.

9.2 Data space

The p-System has an upper limit of 64K and the IARM quotes a minimum limit of 36K. By default the Port-a-kit creates a data space of 64K. This value can be lowered.

```
PSYS volname -d<size>
```

allocates <*size*>k bytes for main p-System memory (heap/stack space). <*size*> must be less than or equal to 64. If PSYS is unable to allocate the requested amount, it successively reduces <*size*> by 5-10% until it succeeds or <*size*> becomes less than 36, in which case the Port-a-kit will not attempt to boot the p-System. If <*size*> is specified as less than 36, the Port-a-kit will attempt to boot the p-System, assuming the allocation succeeds first time, however its behaviour is undefined.

9.3 Mounting external files and serial devices

Assuming that the host Operating System provides a set of primitives for accessing serial channels a method of configuring these to map onto p-System serial volumes would be desirable.

```
PSYS volname -m<unitno><device>
```

mounts <*device*> as p-System unit <*unitno*>. <*unitno*> must be a valid unit to connect either a 'disk' (4, 5, 10-13) or a serial device (6-8, 39-127) *. If <*unitno*> is omitted, PSYS will prompt for it.

9.4 Printing

```
PSYS volname -p<device>
```

connects the p-System unit PRINTER: to the specified <*device*>. -p<*device*> is identical to -m*6*<*device*>. By default, PRINTER: is connected to *stdprn* under *StreamIO.* <*device*> may be a host file that can be printed at a later date.

All large computer systems operate a print spooler for outputting to the line printer. A method of connecting the p-System printer volume to this spooler will be needed.

---

* Note: these numbers vary according to how the p-System is set up. 4 & 5 are usually reserved for left and right 'floppy'. 6 is PRINTER:, 7 is REMIN:, 8 is REMOUT: 10 thru' 12 are the 'externally-mounted disks' (9 is the boot device). 14 thru' 38 are usually subsidiary volumes. Which devices are available for the '-m' option is determined by the units reserved for subvols.

For rapid printing of text files it may prove more effective to use a print program from within the p-System rather than the F(iler T(ransfer command.

Print spoolers vary in how they handle output sent to them. Some cause the spooled text to be sent to a file which is only printed when that file is closed, i.e.,the p-System is H(alted.

9.5 RAMDISK

```
PSYS volname -r<size>
```

allocates <*size*>K bytes for the p-System RAMDISK: which is mounted as #11: If <*size*> is omitted, the default is 256. As with the data and code pool options, if the allocation fails, PSYS will successively reduce <*size*> until it succeeds or <*size*> drops below 25, when RAMDISK: will not be created - 25K is considered the minimum, useable RAMDISK.

9.6 Timer Interrupt

Assuming that some method of both causing and catching a cyclic timer interrupt is available on the host, this option specifies the frequency (assumed to be in seconds).

```
PSYS volname -t<sec>
```

Every <*sec*> seconds, a p-System event associated with the timer is signaled. See page 26 for a discussion of this facility under Unix, and PMEDEF.H for the event number associated with the timer *#define TICK_EVENT*.

Chapter 10

Unix

The Unix Operating System, in all its variations, is becoming available on an increasing number of computers. Most new computers support Unix alongside any manufacturers proprietary Operating System. Also many manufacturers are porting Unix to many existing machines.

As with ‘C’ the belief of ’standard’ Unix is not borne out by reality.

10.1 Using ioctl

To provide the required functionality for p-System terminal i/o under Unix, it is necessary to use the library routine *ioctl* to alter the attributes of the terminal.

During its development, the Port-a-kit was ported to a Unix environment so the prototype for the use of *ioctl* is already in the ‘C’ source code. It is conditionally selected on the *#define has_ioctl* flag in PMEMC.H (see page 4).

ioctl allows the use of *getchar()* to find out whether any characters are waiting to be read by issuing a ’no-wait’ read and testing the result. If any characters are waiting, the first of them is read in by *getchar()* and is stored in a single character buffer (in the *ConBuf* structure - see PMEVAR.H)

Further complications arise from the fact that the Port-a-kit needs to issue reads that ’wait’ for input if no characters are waiting. The Port-a-kit also needs to read strings from the keyboard in PMEDBG (see page 4 for details of *ReadString*, see PMEUMC.H for a prototype of Unix machine definitions).

10.2 Using signal

The library routine *signal* is the simplest way for the Port-a-kit handle interrupts in a graceful manner.

The code to deal with *signal* is conditionally selected by the *#define has_signal* flag in PMEMC.H. The relevant code is in *InitEvents()* and *StopEvents()* in PMETSK.C

All ’kill’ interrupts (Quit, Hangup, Interrupt) should be routed to *pme_exit()* to close down the Port-a-kit gently (closing files and restoring the keyboard state if necessary).

Other interrupts, such as a clock interrupt (Alarm under Unix) can be routed to handlers that call *Event()* on a p-System event number. e.g., code is included in PMETSK.C to show how a clock interrupt can trigger a p-System event (see *InitEvents(), StopEvents()* and *ring_alarm()* in PMETSK.C) An option is provided on the Port-a-kit command line (**-t***N*)

that activates this.

10.3 Using prof

prof is a Unix profiling tool that provides a good indication of how much time is spent in each routine.

Code has been included into the Port-a-kit, conditionally selected by the *#define has_profile* flag, that places profiling marks into key places in PMEFEC.C This gives a better breakdown of the performance of groups of P-codes.

Any profiling information added to the Port-a-kit should always be conditionally selected by a *#ifdef*.

Chapter 11

Input / Output Considerations

11.1 Input / Output in ‘C’

Two forms of i/o are generally provided in ‘C’:

1. buffered, or ’stream’, input/output
2. unbuffered input/output

These two forms have different functions for file input/output and the appropriate definitions are selected by the *#define StreamIO* flag in PMEMCD.H (the associated definitions are in PMEMC.H)

In general, the ’stream’ file i/o will be more efficient than the unbuffered method. Unfortunately, some ‘C’ environments do not provide the full functionality required by the Port-a-kit with ’stream’ i/o. The best approach is to select ’stream’ i/o first, and if the p-System behaves in an unexpected manner, assume that full functionality is not present and alter the flag. Then recompile the Port-a-kit.

11.2 Terminal Input / Output

Since terminal i/o is handled explicitly by user-supplied macros, it is possible to make the terminal buffered or unbuffered in the true ‘C’ sense, i.e., using the same i/o primitives as for file i/o, or using pure system calls.

Terminal output benefits from being buffered and the routine *ubwc()* in PMERW.C uses the file i/o primitive WriteFile to achieve this if possible.

Instead of PUTCH, this routine uses BPUTCH (Buffered PUTCH) which may be coded to save characters in a buffer for outputting with writefile.

Terminal input is complicated by the fact that the Port-a-kit requires that the function ’key–pressed’ to be available for full p-System functionality. This is explicitly coded as the macro *CharWaiting* and the routine *QConsole* (in PMEIO.C). See page 7 for further discussion of terminal input considerations.

Chapter 12

Native Code Support

When the Port-a-kit is ported to a new processor, native code will not need to be supported until either:

a) POP-NCG becomes available for that processor

b) SYSTEM.ASSEMBLER becomes available for that processor

An example of native code support for the Motorola 68000 is given in the Port-a-kit, conditionally selected by the *#define NAT68000* flag in PMEMC.H

When native code is not supported, PMENAT.C will contain stubs that cause execution of p-System to halt when either:

a) Inline native code (the NAT P-code) is encountered.

b) A call is made to a wholly native code procedure (as produced by the System Assembler).

PMENAT.C also contains a stub for segment relocation. This is performed on every segment containing assembly code procedures.

12.1 SYSTEM.PASCAL containing native code

Some existing SYSTEM.PASCALs contain segments that have native code procedures that are only used by such facilities as networking. The Port-a-kit can only boot these p-Systems if the *M–PROC* flag in PMEMC.H is set to an appropriate value, even if the native code is never used. This is because the p-System itself checks all operating system segments for compatibility with the interpreter. Owing to a bug in the p-System bootstrap code, if incompatible native code is found, an attempt is made to print a message using the KERNEL segment before it is resident and p-System loops or crashes! So always make sure that *M–PROC* is correctly set - at worst the Port-a-kit will halt with the 'Native code unsupported' message.

Chapter 13

Using Pdir

13.1 Creating Pdir

Compile PDIR.C and link it with PMEUTL, PMEIO, PMERW and PMETSK. Pdir uses many of the internal PME I/O routines, so it is completely compatible with the UnitRead/Write calls listed on page 14.

When running, Pdir allocates about 20K of working storage for file buffers. If this is not acceptable, alter the *#define TransBufSize* in PDIR.C. The actual amount allocated is *SYSCOMSize + MemRecSize + 2K for a directory buffer + TransBufSize*. (*SYSCOMSize + MemRecSize* is 110 bytes). See comment on I/O buffer size for **UnitWrite(137,...)** on page 14.

13.2 Pdir Options

```
PDIR volname -l
```

list p-System volume in host file *volname.* This is virtually the same as the Ⓔ(xt-dir option in Ⓕ(iler.

```
PDIR volname -x<nblks>
```

extend p-System volume to <*nblks*> blocks. This can be used to dynamically increase the size of a volume. Note that the host file *volname* will not actually change size until more files are imported into it. It is also possible to shrink a directory. Again, the host file will not change size. However, when importing a subvolume, Pdir will extend the target directory as necessary, as if the subvolume were physically full. On those hosts that do not support extension of contiguous files the p-System volume will have been created with its full size.

```
PDIR volname -c<nblks><pvolname>
```

create host file *volname* as an empty p-System volume, logical <*nblks*> long, with p-System name <*pvolname*>. If <*nblks*> is omitted, the default is 2560. If <*pvolname*> is

omitted, it is derived from *volname.* e.g.,

```
PDIR fred.svol -cfred ; create fred.vol, 2560 blocks long, called FRED:
PDIR boot -c260 ; create boot, 260 blocks long, called BOOT:
```

PDIR does not create duplicate directories (although it will handle p-System volumes with duplicate directories).

```
PDIR volname -i<hostname>,<psysname>,<type>
```

import host file <*hostname*> to p-System volume in *volname*, as p-System file <*psysname*> with a file type of <*type*>. If <*psysname*> is omitted, it is derived from <*hostname*>. If <*psysname*> ends in .TEXT, .DATA, .SVOL or .CODE the correct type will be deduced, else DataFile is assumed. If <*type*> is present, it overrides the deduced filetype. <*type*> may be T (for TextFile), D (for DataFile), S (for SVolFile) or C (for CodeFile) e.g.,

```
PDIR main.vol -ifred.pas,fred.text ; import fred.pas as TextFile FRED.TEXT
PDIR main.vol -isyspas,system.pascal,c ; import syspas as CodeFile SYSTEM.PASCAL
```

```
PDIR volname -e<psysname>,<hostname>
```

export p-System file <*psysname*> from p-System volume in *volname* to host file <*hostname*>. If <*hostname*> is omitted, it is assumed to be the same as <*psysname*>.

Options may be combined (in any logical order) e.g., when creating a new boot volume

```
PDIR newboot -c300boot -ipsysos,system.pascal,c -imiscinfo,system.miscinfo …
       … -ipsysulib,userlib.text -isystem.library,,c -l
```

Chapter 14

Existing Implementations

Host cpu	OS	Compiler
80286	MS-DOS	MicroSoft version 4.0 and 5.1, all memory models
80286	MS-DOS	Zorland version 1.10 and 2.0
80386	OS/2	MicroSoft version 5.1
68000	CP/M 68K	Digital Research
68010, 68020	Unix System V.2	AT&T V.2
68000	AMOS	Alpha Micro C
68020	Mirage	Lattice C
ARM	Arthur	Northcroft C
SPARC	Unix	Sun C
6150	Unix	IBM AIX C
MIPS	Unix	MIPS C

TradeMarks

Port-a-kit is a trademark of Knowledge Software Ltd.

POP-NCG is a trademark of Knowledge Software Ltd.

POPTYSER and EDIP are registered trademarks of Knowledge Software Ltd.

UCSD, UCSD Pascal and UCSD p-System are trademarks of the Regents of the University of California.

MS-DOS is a registered trademark of Microsoft Corporation.

Unix is a trademark of AT&T.

Knowledge Software Ltd, 62 Fernhill Road, Farnborough, Hants GU14 9RZ.

Tel: (44) 0252-520667

Telex: 858893 FLETEL G.

Contents