DOS and QNX

There are two scenarios when sharing resources between DOS and QNX:

Remote scenario:

A machine booted in DOS sitting on the same network as a machine booted in QNX.

Local scenario:

A machine booted in QNX that must communicate with a DOS program or DOS resource on the SAME machine (while QNX is running).

For the remote scenario there are numerous third party offerings that allow a QNX machine on a network to exchange data with a DOS machine on that same network (either using NetDDE or a similar protocol). For information on this, please see your Third Party Products Guide or contact QNX customer service.

In this technical note I will discuss the local scenario only.

The Local Scenario

Dosfsys is a program that is shipped along with the standard Operating System. It allows you to adopt DOS devices in a QNX environment. e.g. Drive C under DOS would become /dos/c in QNX. This is the simplest form of file/data sharing... a QNX process can open a DOS file and write/read data. For more information on Dosfsys please see the Utilities Reference A-M (look for Dosfsys).

e.g. cp qnxfile /dos/c
ls /dos/a
etc..

Rundos 4.2 provides some more sophisticated tools for using DOS resources; in fact Rundos will allow you to run a DOS program as another process in the QNX environment. These next examples will discuss some ways of moving data between a running DOS program (running under Rundos) and other QNX processes.

There are 3 main ways of moving data between a DOS program and a QNX program:

1. Using a file.
2. Using a FIFO.
3. Using a special DOS interrupt service that QNX adopts.

1. Using a file.

Rundos allows you to assign DOS drives to QNX drives and directories. For example, suppose I had a DOS program that expected to use the file E:\OUTPUT for its output.

When I start this program (let's call it DOSTEST.EXE) I will point the E drive to be an area of the QNX filesystem that I have reserved for this purpose.

e.g. rdos DOSTEST.EXE c=. e=/dosqnx/

When I start the DOSTEST program it will output its data to /dosqnx/output. All standard DOS and BIOS services including file sharing (use the +SH option on rdos to emulate SHARE.EXE) are available to the DOS program. This can be a very effective way of moving data back and forth. If the /dosqnx directory is on a ramdisk then this will increase performance.

```
/****** DOSTEST.C ************************************************/


.. startup code ...

FILE *fp;

fp = fopen("E:\OUTPUT", "w");
fprintf(fp, "Sample data.\n");
etc...

/* this will write to /dosqnx/output *


*********************************************************************/
```

2. Using a FIFO.

A FIFO (first in, first out) created with the mkfifo command can be a useful alternative to a file.
Suppose a fifo is created called '/dosqnx/dosfifo'. The same example described above can be used with a fifo:

e.g. rdos DOSTEST.EXE c=. e=/dosqnx/

The DOSTEST.EXE program can open up E:\DOSFIFO and write/read data the same way as it can with a regular file. The advantage of using a fifo over a file is that the filesystem handles the contention for the fifo for you. For example, if the DOS program is reading from the fifo, I can have multiple QNX programs dump data to the fifo

e.g. 'sin >/dosqnx/dosfifo' on one screen while 'ls >/dosqnx/dosfifo' on another.

3. Using special DOS services.

Please see pages 68-69 in the Rundos Manual before reading this section. QNX uses an unused DOS service (int 21h, service 44h) to provide a mechanism for moving data back and forth between a DOS program and a QNX program.

Here's how it works from the DOS program:
Two new DOS functions are created: onpty() and ischars()

int onpty(int *fd, char *qnxcmd)
you pass in the pathname of a QNX process that you want to start.
e.g. onpty(&fd, qnxcmd)
where - fd is the file descriptor that will be used to communicate.
- qnxcmd is the QNX command to run (a string).
e.g. /bin/sin or qswap (see example)

int ischars(int fd)
using the file descriptor returned by onpty you can use this function to determine if
there are characters waiting to be read by the DOS program.

The QNX process started by onpty() will be spawned with its stdin pointed to the file descriptor 'fd' and its stdout also pointed to this file descriptor.

In this example, the program 'dos2qnx' is compiled as a DOS executable.The program 'qswap' is compiled as a QNX process.

Assume both 'dos2qnx.exe' and 'qswap' are in directory /dostest

cd /dostest
rdos "dos2qnx /dostest/qswap" c=.

The program uses a simple ack protocol (the letter Z) to indicate the end of a write from the QNX program. Any protocol at all may be used. Note that this example is very simple, a more sophisticated message structure may be overlayed on top for better error checking (checksums etc.). Note in the function onpty() the differences for compiling in an Microsoft Windows environment.

/ EXAMPLE **

PROGRAM: dos2qnx.c (DOS)

This program demonstrates how to move data back and forth between QNX and DOS in both MS Windows(tm) and normal DOS.

The DOS program uses _dos_read/_dos_write to move data to and fromthe QNX application. The QNX application uses stdin and stdout ... see the example code called 'qswap.c' for a simple example (listing follows this one).

Usage: run the program from rdos such as:
rdos "dos2qnx <program>" c=.
where <program> can be anything (e.g. /bin/sin or qswap)

Compilation:
To compile for DOS I used Watcom C version 9.5 with the DOS libraries. (compiled under QNX)

cc -p dos -L /usr/doslib/dos -2 -ms dos2qnx.c -o dos2qnx.exe

You could use any DOS compile environment that you wish. (Turbo C, C++, Microsoft C, Visual C++ for Windows etc.) All that may be necessary is to port the two functions onpty and ischars to the new compile environment.

****

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <io.h>
#include <dos.h>

typedef unsigned short      WORD;
typedef unsigned long       DWORD;

#define LOWORD(l)           ((WORD)(DWORD)(l))
#define HIWORD(l)           ((WORD)((((DWORD)(l)) >> 16) & 0xFFFF))

int ischars(int);
int onpty(int *, char *);

int portfd = -1;   /* default to no link created */

#define BUFSIZE 512
#define MSGSIZE 4096
char buf[BUFSIZE];     /* a buffer to store incoming data */
char qnx_cmd[80];      /* the command to execute on the QNX side */

void main(int argc, char *argv[])
{
   int fd, i;
   int status;
   int n;
   int sread;
   int numread;
   int count = 0;
   int got_last;
   int bcount = 0;

   /* strcpy the command that you want to run (with arguments) into
      qnx_cmd[]. If qnx_cmd is a null string, then /bin/login will be        run.
    */
   if ( argc > 1 ) {
      strcpy(qnx_cmd, argv[1]);
   }
   status = onpty( &portfd, qnx_cmd );
```

```
    if ( status == -1 ) {
       printf("Could not establish link to QNX process.Aborting.\n");
       exit(-1);
       }

        while(1)
        {       while( (n = ischars(portfd)) > 0) {
                if(n > sizeof(buf))
                n = sizeof(buf);
                sread = _dos_read( portfd, buf, n , &numread);

                if ( sread != 0 )
                printf("error on read: %s\n",strerror(errno));
                printf("Got a read of %d bytes. Byte count=%d\n",
                numread,bcount);

                bcount += numread;
                if ( bcount >= MSGSIZE ) {
                sendqnx("Z");
                count++;
                bcount = 0;
                if ( count > 9 ) {
                printf("Iterations complete. Hit ENTER");
                getchar();
                exit(0);
                 }
              }
           }
    }
}

sendqnx( char *dstr )
{
int swrite;
int numwrit;

swrite = _dos_write( portfd, dstr, strlen(dstr), &numwrit);
/*
printf("wrote %s: swrite=%d, numwrit=%d\n",dstr, swrite, numwrit);
*/
}

/* see pages 68-69 in the RunDos Manual for more explanation */
```

```
/************************** ISCHARS ****************************/

/* check to see if there are characters waiting to be read .. these
   characters have been written to stdout on the QNX application side.
   Arguments:     int fd      : the file descriptor obtained with onpty()
   Return value:  -1 on error   (EBADF or EACCES)
                  >= 0  indicates the number of chars waiting
 */

int ischars(int fd)
    {
    union REGS inregs;
    union REGS outregs;

    inregs.x.ax = 0x44F0; /* IOCTL type F0... dev_ischars */
    inregs.x.bx = fd;
    inregs.x.cx = 0;
    inregs.x.dx = 0;
    inregs.x.si = 0;
    inregs.x.di = 0;
    outregs.x.cflag = 0xFFFF; /* just in case */
    intdos( &inregs, &outregs );
    if(outregs.x.cflag) {
        return(-1);
        }
    return(outregs.x.ax);
    }
```

```
/************************** ONPTY ********************************/

/* create a link between the DOS program and a QNX program.
  Arguments:     int *fd   : pointer to a file descriptor
                 use this fd for all future
                 _dos_read/_dos_write calls.
                 char *cmd : pointer to a command string to run (may
                 include arguments)
                 If NULL then /bin/login will be assumed.
  Returns: 0x08 : insufficient memory (MS Windows program only)
           0x01 : unsupported function call. You must run rdos version
                  4.2
           0x00 : SUCCESS!!
       A link has been created. If 'Dev.pty' was NOT  previously running
       on the node, then Dev.pty is started.  This program will use an
       unused /dev/ttyp* to communicate to a /dev/ptyp on the QNX side.

       Special Note:  The ptyp used by the QNX application is PASSED
       down to any children it spawns.

           0x** : anything else indicates an error.
 */

int onpty( int *fd, char *cmd )
   {
   union REGS inregs;
   union REGS outregs;
   DWORD dosmem;
   WORD _far *pPointer;
   WORD selector;
   WORD segment;
   WORD offset;

   if(cmd != NULL  &&  *cmd != 0) {
#ifdef WINVER       /* an MS Windows program */
      /* Note that MS Windows requires the cmd string (only if
         defined) to be copied to a globally accessible memory
         region. This bit of code does all the magic for us.
      */
      dosmem = GlobalDosAlloc( strlen(cmd) + 1 );

      if(dosmem == 0)
         return(0x08); /* Insufficient memory */
```

```
        selector = LOWORD(dosmem);
        segment =  HIWORD(dosmem);
        offset =   0x0000;

        /*
         * Make a far pointer to the memory allocated in BASE memory
         * which is addressabale from this LDT
         */
        pPointer =
        (unsigned short _far *) ( (((unsigned long) selector) << 16 )
         |  offset);
        _fmemcpy( pPointer, cmd, strlen(cmd) + 1 );
#else
/* at this point we assume a normal DOS environment in real mode */
        selector = 0;
        segment  = FP_SEG ( cmd );
        offset   = FP_OFF ( cmd );
#endif
        }
    else {   /* no cmd[] argument was given... use default */
        /*
         * Use Rundos default command (/bin/login)
         */
        offset = 0;
        segment = 0;
        selector = 0;
        }

    inregs.x.ax = 0x44F1; /* IOCTL type F1... onpty() */
    inregs.x.bx = 0;
    inregs.x.cx = offset;
    inregs.x.dx = segment; /* 0 means default */
    inregs.x.si = 0;
    inregs.x.di = 0;

    outregs.x.cflag = 0xFFFF; /* just in case */
    intdos( &inregs, &outregs );

    if(selector)
#ifdef WINVER
        GlobalDosFree( selector );
#endif
    if(outregs.x.cflag) {
        if(outregs.x.ax == 0x44F1)
```

```
        return(0x01); /* Unsupported function */
      return(outregs.x.ax); /* Other error */
      }
   *fd = outregs.x.ax;
   return(0);
   }
```

/* qs wap.c ***

Compiled in QNX Watcom C 9.5 (compiled 32-bit).

Write a buffer of data to stdout. Then send a Z (to indicate end of transmission). Then wait for a Z to come back. Repeat 10 times.

/

```
#include <stdio.h>
#include <sys/dev.h>
#include <sys/types.h>
#include <sys/sched.h>
#include <sys/osinfo.h>
#include <string.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <i86.h>

#define BUFSIZE 4096

char buf[BUFSIZE];

main()
{
  FILE *logfile;
  int count = 0;
  int ret;
  short int c;
  unsigned old_mode_stdin;

  logfile = fopen ( "qswap.log", "w" );/* save results in a logfile */
  memset(buf,'A', sizeof(buf));
  buf[BUFSIZE] = 0;
```

```
  /* make sure stdin is in RAW mode ... */
  old_mode_stdin  = dev_mode(STDIN_FILENO,0,0);
  dev_mode(STDIN_FILENO,0,_DEV_MODES);

while (count++ < 10) {
   fprintf(logfile,"Doing iteration %d of %d bytes.\n",count,
                   BUFSIZE);
   write_buffer();
   ret = get_ack();
   }

  fclose ( logfile );

dev_mode(STDIN_FILENO, old_mode_stdin, _DEV_MODES);
}

write_buffer()
{
int swrit;

swrit = write( STDOUT_FILENO, buf, sizeof(buf));
fputchar('Z');
fflush(stdout);
}

get_ack()
{
short int i=0;
char c;

while( i < 10 ) {
    c = getchar();
    if ( c == 'Z' )
       return(1);
    }
    return(0);
}
```

/**

Some final notes ...

Using the above programs (dos2qnx.exe and qswap) throughput of 32 Kilobytes per second was observed on a 386SX/25. This was using 512 byte reads on the DOS side.

This throughput should be adequate in the most common case of passing data from a data collection process on the QNX side to a DOS program that is displaying the data (under MS Windows for example). Note that even though the dos2qnx.c program used '_dos_read()' and '_dos_write()' a regular read() or write() would also work fine (any input/output call that uses a file descriptor).

You will notice that the mechanism that is used to buffer the data going to and from the DOS program is a pseudo tty (ptyp) managed by the Dev.pty resource manager. This link is established when the onpty() function is called. If Dev.pty was NOT running before onpty() is called then Dev.pty will be started for you by rundos.
A free ptty must be available when onpty() is called otherwise an error code will be returned. This may be important if you are already using Dev.pty for other programs (such as TCP/IP).

**********/