Tops-20 Kermit-20 Batch Test Battery

Table of Contents

Introduction

Before any major release is contemplated, Kermit-20 must pass a battery of twenty four (24) separate regression tests which exercise all available functionally that can be tested under via pseudo-terminal (also known as loopback testing) as well as DECnet NRT transport.

In addition, Kermit-20 must communicate successfully with C-Kermit using TCP/IP based virtual terminals (also known as TVT's or Telnet) and properly handle the SPACE, PWD, CWD, CDUP and DIRECTORY commands as well as a GET and PUT of both 7 text and 8 bit binary files. Remarkably, while many macros exist for C-Kermit, none of them appear to test such basic functionality and these tests are run manually in the course of normal packaging.

Unfortunately, not all functionality can be tested. Currently, no actual physical serial hardware is available to test Kermit-20 from. It is believed that Kermit-20 will still function as a great deal of effort was expended to not break serial support by simulating a physical terminal as much as possible with pseudo-terminals. The protocol itself, of course, has no idea of what kind of connection it might be running on.

The following batch tests and their resulting log files are provided as a courtesy and it is hoped that Kermit-20 enthusiasts will run them on their own systems. Timings should be taken with a gigantic grain of salt as a large number of factors can skew them. They are at best suggestive, and are only used for comparison with previous implementations on the same platform.

Building Kermit-20

All Kermit-20 releases are accompanied by a `K20BLD.CTL` control file which will completely build Kermit-20 entirely from scratch. The accompanying log file, `K20BLD.LOG`, which will explicitly show the files being opened is also included. This produces a cross referenced listing and takes about 23 seconds of processor time and perhaps ½ a minute of wall time.

`K20MAK.MIC` is also provided for assembly during a timesharing session. Typing `DO /L COMPIL K20MAK` will perform an incremental assembly of only changed modules and not produce a cross-referenced listing. It is thus far faster.

Triggered Sequential Testing

The tests are entirely separate and there is no requirement to perform them in the order shown. That being said, three `.MIC` files are provided to queue related batteries for execution, the differences being whether regression testing is being done against the release candidate with itself, with older Kermit-20's, or with Kermit on other platforms and which kind of regressions being run.

The first, **`K20ALL.MIC`**, submits a battery of sixteen separate tests: K20NUL, K20NUP, K20PTY, K20NRV, K20NRT, K20NUR, K20B8T, K20B8P, K20B8A, K20DPD, K20TCP, K20TCN, K2036P, K2036C, K20GPT, and K20TPO for sequential execution. These exercise new and reimplemented functionality.

The second, **`K20174.MIC`**, submits a battery of six separate tests: K20NOL, K20POL, K20PNW, K20POR, K20P8R, and K20RDC. These are regression tests against Kermit-20 4.2(174) from

May 2nd 1985. The lesser total execution time of Kermit-20 4.2(174) of 22.7 seconds should be ignored as it is being asked to do less (such as not logging).

Altogether, the tests take less than eighteen (18) minutes of wall time to complete, much of being used for non-local communications latency.

The third, **`K10ALL.MIC`**, submits a battery of three separate tests: `K10NRT`, `K10NRP` and `K1036C`. These are regression tests against Kermit-10 V3(136) running on Tops-10 7.04 NET. Timings are 2:47 processor and 2:14:57 wall time. These figures are suspect, possibly due to timing issues with TOPS-10 and (less likely) the `simh` KS10 micro-engine. Note, due to availability of Tops-10 systems, these tests—which deal only with regression—are not always run.

Speed Measurement Considerations

Differences in speed are seldom worthy of comment unless some situation yields such truly abysmal throughput that the subject is unavoidable, although this is typically due to simulator quirks. Otherwise, Kermit-20 is about as efficient and fast as it can be, years' worth of development effort having been expended in this area, some of it coupled with reading (and sometimes modifying) Tops-20 sources.

For example, as of the time of writing (late 2024Q3), Kermit-20 was tested sending a Tops-20 local file (`TOMMYT:<DOCUMENTATION>JSYS_REFERENCE.MEM`) of 1,794,319 seven bit ASCII bytes to `/dev/null` on Mac OS X 13.5.1 using C-Kermit 10.0 pre-Beta.11, 26 Mar 2024 (Numeric: 1000411) with a packet size of 9,000 bytes (the maximum size possible).

C-Kermit reports a data rate of 234.33 KC/s (or 2.289 MBd), the entire transfer taking 7.297 seconds. In contrast, Kermit-20 measured a total characters per second figure of 180.1997 KC/s and an effective data rate of 2.1569 MBd, taking 7.93345 seconds to complete.

These numbers are roughly the same, but why are they not exactly the same? It's not a completely answered question, but one factor in the difference is probably that C-Kermit and Kermit-20 appear to be considering differing points in time as the beginning and end of the transfer. Therefore, the comparison of the figures may not be valid.

Also, the reason Kermit-20 is reporting 2.1569 MBd as the baud rate is because this is the *effective* baud rate, taking expansion of compression sequences into account. Otherwise, the corresponding baud rate to 180.1997 KC/s would be 1.7597626953125 MBd. The point is that making sure the measurements are analogous is not an immediately obvious or even intuitive decision.

More importantly, C-Kermit is essentially seeing coaxial cable based Ethernet transfer rates of the early 1980's. If you really need speed, then the thing to do is consider implementing sliding windows. Also, remember that Kermit's original purpose was not to provide the maximum possible speed, but to not crash the DECSYSTEM-20's front end while providing communications integrity in an openly and freely distributed protocol as well as a rather large number of source code and executable files.

`K20RDC` does do such an apples-to-apples comparison, although its purpose is to validate character counts. That being said, the same set of 11 files totaling 85,840 characters was sent both the old and new versions of Kermit. These were sections of out of copywrite book, The Ghost of Canterville by Oscar Wilder and are as follows:

File	Pages	Characters
GUTENBERG-POSTFIX-AND-LICENSE.TXT.1	8	19,066
GUTENBERG-PREFIX.TXT.1	1	1,048
I.TXT.3	4	8,637
II.TXT.2	3	6,849
III.TXT.1	6	13,243
IV.TXT.1	4	9,783
LIST-OF-ILLUSTRATIONS.TXT.1	1	834
TITLE-PAGE.TXT.1	1	302
V.TXT.2	4	9,179
VI.TXT.1	4	8,469
VII.TXT.1	4	8,430
Subtotal:	40	85,840

With Kermit-20 4.2(174) doing the sending, a throughput of 66.6436 KC/s (580.274 KBd, effective) was achieved for a total duration of 1.44463 seconds. The newer Kermit-20 took 1.16638 seconds to yield a throughput of 82.5429 KC/s (718.7034 KBd effective), an increase of 23% (15.8993 KC/s)

Of more interest, perhaps, are the processor time differences, Kermit-20 4.2(174) taking 600 milliseconds, the newer Kermit-20 taking 400 millisecond, a reduction of a ⅓.

That being said, this is anything but a fair comparison as Kermit-20 4.2(174) dates to May 2nd, 1985, nearly four decades ago and altogether a very different time. Processor utilization was simply not a factor as 2400 baud dial up was just being deployed, 1200 baud being the most common high speed dial up. However, even at the typical Columbia campus connection speed of 4800 baud, Kermit-20 use did not present a significant load.

More importantly, is the fact that these numbers can vary wildly, with the 1985 Kermit sometimes outperforming the 2024 Kermit! Again, the testing results have to be taken with a grain of salt. A more thorough analysis would certainly be of interest, but the 2024 Kermit is being artificially constrained as the typical use case is to use 9,000 character buffers for upload to a TCP/IP Kermit client.

Loading Considerations

While it may be possible to run these jobs in parallel, it is very strongly and most emphatically recommended that you *not* do this as a load test.

- First, and of critical importance, the temporary file names that are generated are known to be all unique, making it not impossible for one test to overwrite the working results of another.
- Second, because you will then be executing on a shared system, you will distort internal timing measurements, making comparisons with previous code execution speeds dubious at best.
- Third, putting certain simulators under extreme load has triggered false hardware errors and could bring out other timing bugs, yielding false negatives to the tests or even crashing the system.

The User Environment Testing Package (UETP) should be used for load testing and system checkout.

K20NUL: Basic Kermit-20 Server functional checkout

1) Pseudo-terminal login
2) INPUT statement test with C escape sequences
3) OUTPUT statement test with C escape sequences
4) Basic text file transfer checking
5) Wildcard file transfer checking to NUL:
6) Wildcard file transfer checking with different checksums
7) Session log testing
8) 94 character packets (largest for basic protocol)
9) Basic SERVER command repertoire
10) Basic NUL: tests

K20NUP: Basic Kermit-20 Server functional checking using PUT

1) Like K20NUL, except uses push to remote NUL: instead of GET to local NUL:

K20PTY: Kermit-20 Server Transfer additional and post transfer checks

1) Prompt defaults (DECnet node if remote) and parsing
2) Time-out parsing
3) 120 character packets (larger then basic minimum, somewhat faster)
4) Internal transport timing checkout
5) INPUT/OUTPUT statement tests with C escape sequences
6) Pseudo-terminal login with no secondary fork
7) Basic macro definition check
8) Basic SERVER command repertoire
9) Basic NUL: tests
10) Short duration communications check: 1.62 milliseconds!
11) Basic text file transfers
12) Comparison of statistics between both sides (they're 'close'...)
13) Large text file test:

Quantity	Type
1,794,319	ASCII (7 bit) bytes
701	Tops-20 disk pages
9,000	Packet size (largest possible)
101.8023	KC/s (Highest total characters/second)
1.2186	MBd (Highest effective data rate)

14) Post transfer file comparisons (all OK)

K20NRV: Kermit-20 Server functions via Tops-20 DECnet NRT to local host

1) Same as K20PTY except 94 byte packet size only
2) DECnet NRT connection to the local Tops-20 host
3) Checks files, post transfer (all OK)

4) Does not include large file case
5) Fastest possible NRT (all communications are memory only)
6) Additional program information (memory layout)

K20NRT: Kermit-20 Server functions via Tops-20 DECnet NRT to remote host

1) Similar to K20NRV except going to a different Tops-20 host
2) Shows Flow-Control being cleared
3) Shows smaller packet sizes (which may be faster under certain simulators)
4) Slower transfer speeds (suspected to be an emulator artifact)
5) Example of BYE command
6) Example of /TIMEOUT

K20NOL: Basic regression test against Kermit-20 4.2(174) [2-May-85]

1) Similar to K20NUL with reduced functionality because 4.2(174) does not have:
   a) No remote NUL: testing
   b) No PWD testing
   c) No remote statistics
   d) No large file/buffer
   e) No pseudo-efficiency simulations
2) 4.2(174) reports disk quota of 70,000 as "+Inf"
3) Compression appears to be somewhat better (could be incorrect, however)
4) Session logging test
5) N.B., 174 decimal is 256 octal

K20POL: Regression test against Kermit-20 4.2(174) [2-May-85] using GET

1) Similar to K20PTY with reduced functionality
2) Otherwise, similar to K20NOL, except for 3)
3) Post transfer file comparisons (all OK)

K20PNW: Regression test against Kermit-20 4.2(174) [2-May-85] using PUT

1) Same as K20POL, but uses PUT instead of GET

K20NUR: Basic Kermit-20 Server functional checkout with parity

1) Like K20NUL (Basic Test), but using parity on packets
2) Somewhat slower transfer, perhaps 5% impact (may be false negative)

K20POR: Regression test against Kermit-20 4.2(174) [2-May-85] with parity

1) Like K20POL with parity checking
2) ASCII 7 bit files

K20B8T: Kermit-20 Server 8-bit file transfer and post transfer checks

1) Like K20PTY, except 8 bit files

K20B8P: Kermit-20 Server 8-bit file transfer with parity and post transfer checks

1) Like K20B8T, except does 8 bit files with even parity

K20B8A: Kermit-20 Server terminal 8-bit file transfer with parity and transfer checks

1) Like K20B8E, except also checks parity from Tops-20 terminal driver

K20P8R: Regression test against Kermit-20 4.2(174) [2-May-85] 8 bit files with parity

1) Like K20B8P and K20B8T, except does 8 bit files with even parity

K20RDC: Communications comparison Kermit-20 4.2(174) [2-May-85] and 5.3(255)

1) Regression test to send the same file and check character counts
2) Shows use of ECHO /DEBUG to put messages into log files

K20DPD: Kermit-20 Packet Decoding Example

1) Enhancement to decode a packet instead of dumping raw data
2) PTY transfer checks for fineness of time logging
3) Will use high precision Time of Day functionality (HPTIM% .HPTOD), if available
4) Otherwise, unmodified Tops-20 design limits time of day resolution to a little less than a 1/3 of a second (0.32958984375 seconds). In this case, Kermit-20 will attempt to construct millisecond resolution.
5) Shows timing resolution and number of bytes written to log file.

K20TCP: Kermit-20 Transmit/Capture Testing via pseudo-terminal

1) Basic transmit, no parity processing, captured to NUL:
2) Transmit, space parity processing, captured to NUL:
3) Transmit, mark parity processing, captured to NUL:
4) Transmit, odd parity processing, captured to NUL:
5) Transmit, odd parity processing, forcing a parity error
6) Transmit, full even parity, including Tops-20 terminal line generated
7) Post transmit comparison checking of large and small files

K20TCN: Kermit-20 Transmit/Capture Testing via DECnet NRT

1) Same as K20TCP, except tests are performed over a Tops-10/20 type DECnet Network Remote Terminal
2) Since DECnet NRT's do not generate parity, test 6 is performed without terminal line parity checking

K2036P: Kermit-20 36 Bit Mode via pseudo-terminal

1) Same as K20PTY, except no large files, all files being executables
2) Demonstrate correct and enhanced transaction logging
3) FILCOM/E demonstrates transfers correct to bit level
4) Double checked against EXEC directory, CHECKSUM BY-PAGES
5) Pre and post transfer file byte sizes and counts visually compared to be correct

K2036C: Kermit-20 36 Bit Mode via pseudo-terminal with parity

Note, be aware that while BATCON will properly strip parity for .LOG files, performing a DO of the same control (.CTL) file as if it were a .MIC file can result in strange output on the local terminal as Kermit-20's ECHO command will force parity. This is a design feature which is used for checkout and debugging.

Further information may be gotten from Kermit-20's extensive and up to date built-in help by typing HELP ECHO and also HELP SET PARITY.

1) Same as K2036C, but with parity sending on terminal and packets and checking being performed
2) Demonstrate correct and enhanced transaction logging which is transparent to parity
3) Demonstrate proper error handling when a file cannot be opened by the Kermit-20 server;
   a. That server state is updated,
   b. That an 'E' packet is sent,
   c. That the 'E' packet is properly interpreted by the Kermit-20 client, and
   d. That no-remnants of the file are left in the testing directory.

K20GPT: Kermit-20 CDUP and remote Relative Directory Connection

1) Tests Local CDUP gets to superior (or upper) directory
2) Verifies that "Local CWD .." functions exactly like CDUP
3) Tests Remote CDUP
4) Verifies that "remote CWD .." functions exactly like a remote CDUP
5) Tests that remote relative directory specifications are interpreted properly
6) Produces decoded packet streams at microsecond resolution (if the monitor supports this, milliseconds, otherwise)

K20TPO: Kermit-20 Test Parity Override

1) Performs several stress tests to force parity errors
2) Demonstrates Parity Error counting and substitution
3) Demonstrates Enhanced Transmit Functionality
   a. Maximum length of sent text
   b. Pause after sending each line

   c. Use of SMAP% file large files to save processor time
   d. Override of global settings
4) Shows files with over-printing have bare carriages replaced with carriage return line feed.
5) `FILCOM` does not notice such differences.
6) Produces transmit speed calculations, approximately 69 KC/s (about .5 Mbps)

`K10NRT`: Kermit-**10** Regression Tests via DECnet NRT

1) Remote terminal tests (`LOGIN`, `SYSTAT`, Etc.)
2) Basic server tests, `REMOTE SPACE`, `REMOTE HELP`
3) `REMOTE DIRECTORY`, `REMOTE DELETE` validation
4) Kermit-10 does not implement CWD/PWD (as Tops-10 does not have the idea of a connected directory)
5) Transfer and comparison of various files, some quite large (all OK)
6) Session logging

`K10NRP`: Kermit-**10** Parity Regression Tests via DECnet NRT

1) As `K10NRT`, except `REMOTE DELETE` not validated and remote terminal assumed to work
2) Transfer and comparison of various files, none particularly large
3) Errors
   a. Correct parity **not** sent in response to FINISH
   b. Correct parity **not** sent in wildcarded directory listing

`K1036N`: Kermit-**10** 36 bit Regression Tests via DECnet NRT

1) Test 36 bit mode against only other 36 bit client,
   a. Kermit-**10** version 3(136)
   b. Tops-10 7.04 NET (KS10)
2) Comparison check against three executables on SYS: (`DSKA: [1,4]`)

```
Name   Extension Blocks Protection    Date      Version
MACRO    EXE       108     <055>     1-Sep-88   53B(1247)
CREF     EXE        28     <055>     1-Sep-88   53C(101)
FILCOM   EXE        28     <055>     1-Sep-88   22A(121)
```

3) Example of local session log
4) Example of local transaction log