Issue with RTCS running out of memory during data transfer

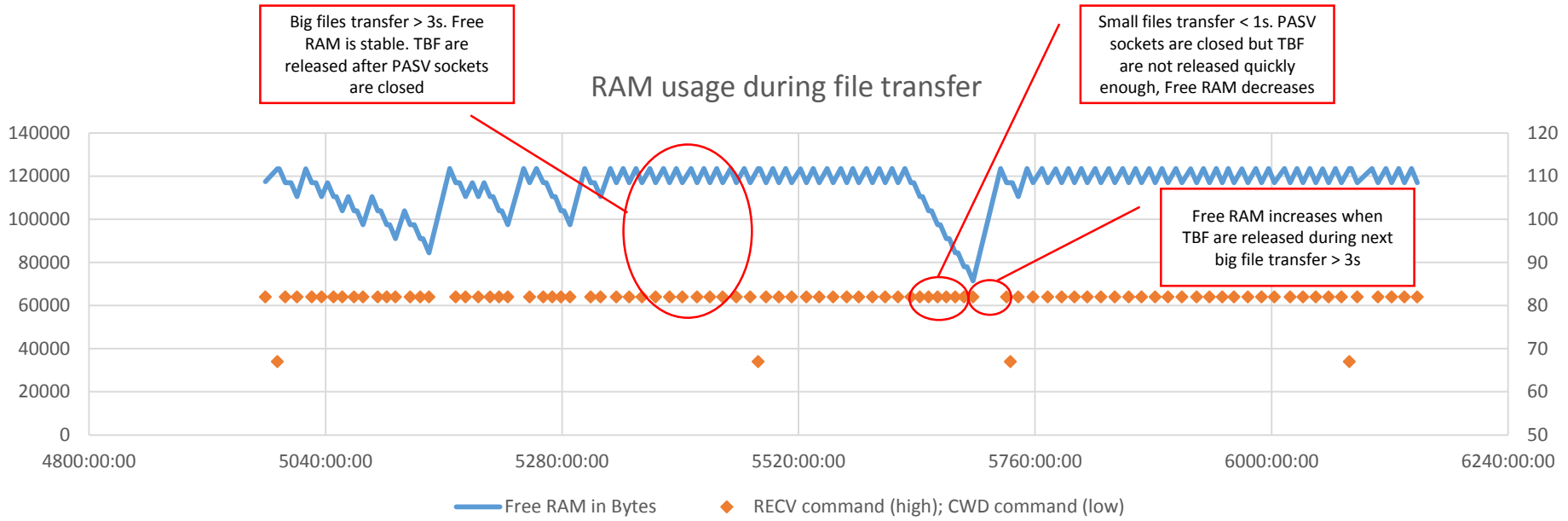
March 2015

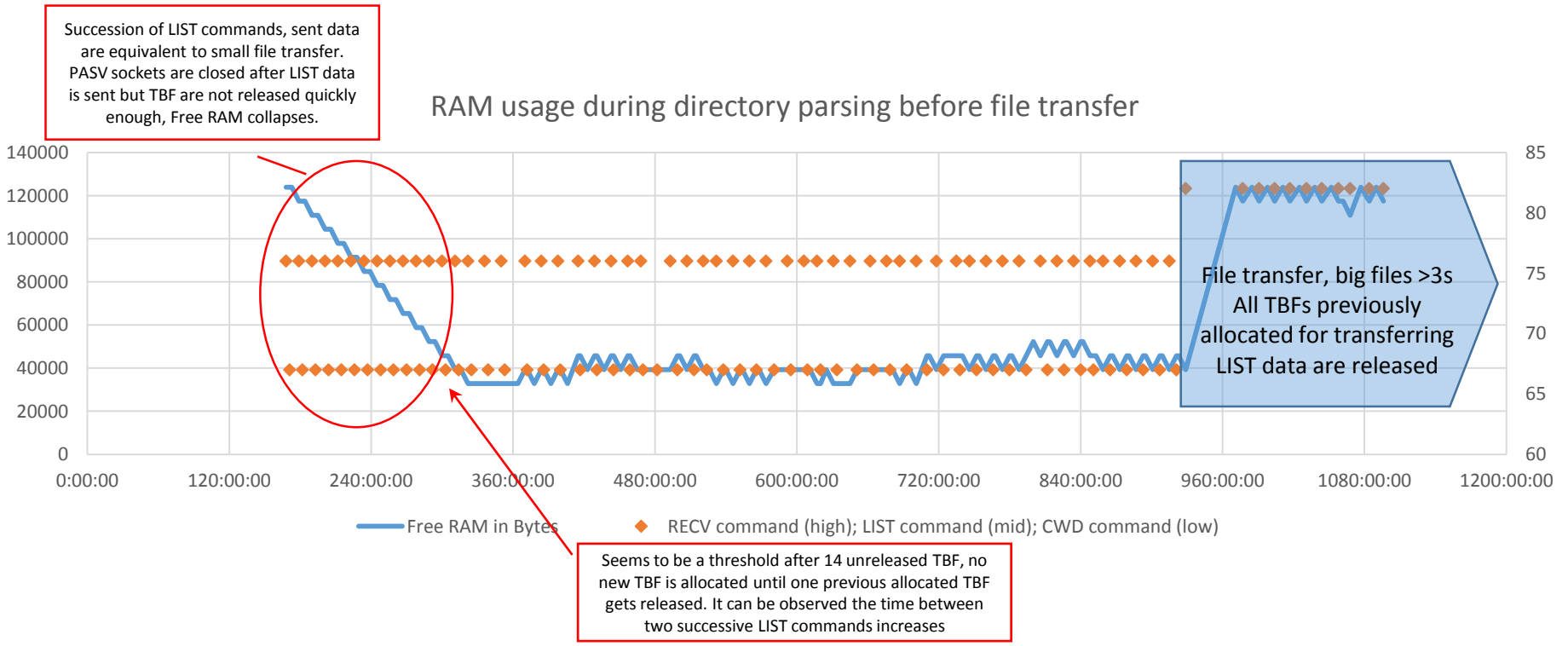
Issue description (high level)

- Issue is as follow: Internal RAM memory quickly turns full during FTP file transfers or HTTP transfers (file, CGI, …). In the TAD with MQX 4.1.1, RTCS task reports Out of Memory error. At first sight the issue seems to be random (actually we will see it is not).
- We have been able to isolate the issue to successive TCP transfers of small files. In other words the issue happens when the time from connecting the socket for data transmission + time to transfer the data + time to release the socket is below the TCP retransmission timeout value (3 seconds).
- Issue is observed on our boards with MQX 4.0.2 and MQX 4.1.1 and with Freescale K64 and K60 devices. Reading various posts on the Freescale MQX community forum, similar issue has been reported since MQX 3.8 by several users on the Tower boards.
- According to the posts on the forum, the typical trick to hide the issue consists in tweaking the various RTCS parameters for TCP and IP protocols such as the buffer size or numbers of PCBs or number of sockets, etc till reaching a compromise that works. However this trick suffers negative consequences to the end users: first it decreases the TCP/IP performances, second it is not sufficient for small memory devices such as the K60, third it penalizes the other user tasks that cannot be used in parallel of FTP/HTTP file transfers due to lack of available memory in the system.
- No external RAM is mounted on the boards, MQX is used with the standard configuration for small memory devices, with default clock configuration and with no low power mode enabled. RTCS task is assigned with the default priority (6) and no other user task in the system has a higher priority (all other tasks have priority above 8). RTCS task is not given a dedicated pool, by default it uses the system memory pool.

Issue description (technical)

- The root cause of the issue is in the way the resources allocated to cloned TCB are managed in the RTCS code. It can be observed that when issue happens, the memory is full of TCBs structures. It happens when there is a continuous flow of FTP data transfer requests, with less than 3 seconds between each. If the time between two requests is longer than 3s, the problem disappears. Which can give the feeling that issue is somehow random when size of data to transfer fluctuates.
- After shutting down a TCP stream socket, resources allocated to the TBF for data transfer (cloned TBF) are released only after the TCP retransmit timeout expiry (3 seconds), even if parent socket (*) are aborted. During that time, if new incoming requests for data transmission happens, new TBFs are allocated. It could happen situations where two many TBFs are pending the timeout expiry, causing a lot of memory being used. If not enough memory available in the device, RTCS task or other user tasks will run out of memory.
- (*) by parent socket, it should be understood a hierarchy structure between socket that TCB is cloned from (parent), in our case the socket used for listen(), and the socket that is returned after accept() (child)


Big files transfer > 3s. Free RAM is stable. TBF are released after PASV sockets are closed
RAM usage during file transfer
Small files transfer < 1s. PASV sockets are closed but TBF are not released quickly enough, Free RAM decreases
Free RAM increases when TBF are released during next big file transfer > 3s
140000
120000
100000
80000
60000
40000
20000
0
120
110
100
90
80
70
60
50
4800:00:00
5040:00:00
5280:00:00
5520:00:00
5760:00:00
6000:00:00
6240:00:00
Free RAM in Bytes
RECV command (high); CWD command (low)


Succession of LIST commands, sent data are equivalent to small file transfer. PASV sockets are closed after LIST data is sent but TBF are not released quickly enough, Free RAM collapses.
RAM usage during directory parsing before file transfer
140000
120000
100000
80000
60000
40000
20000
0
85
80
75
70
65
60
0:00:00
120:00:00
240:00:00
360:00:00
480:00:00
600:00:00
720:00:00
840:00:00
960:00:00
1080:00:00
1200:00:00
File transfer, big files >3s
All TBFs previously allocated for transferring LIST data are released
Free RAM in Bytes
RECV command (high); LIST command (mid); CWD command (low)
Seems to be a threshold after 14 unreleased TBF, no new TBF is allocated until one previous allocated TBF gets released. It can be observed the time between two successive LIST commands increases

Conditions to reproduce and debug (setup)

- For debug and explanations that follow, we have been concentrating on the case of FTP file transfer, with MQX RTCS implementing the FTP server, IPv4 only. Use case limited to a single connection at a time, passive mode, binary type of file transfer. The same issue happens for other types of TCP file transfer.
- Creates a directory structure of several directories into the MFS file structure. Into some of these directories create non empty files of several thousand bytes. Into the other dicrectory create files of more than 10 mega bytes so that the transfer of those files will last more than 3 seconds.
- Use a FTP client such as Mozilla FileZilla, or any other FTP client that has directory parsing and FTP file retrieve enqueing capabilities.
- Connect to the server. The server root directory will be displayed. Select all directories and initiate a transfer to the client. In the case of FileZilla, the client will initiate a sequence of FTP LIST commands to parse the entire directory structure and build the list of files to get. When completed, it will then send a sequence of FTP RETR commands to transfer the files to the client. Each LIST and RETR commands will be preceded by a PASV command to force the passive mode according to the FTP standard.
  It is worth noticing that upon reception of a LIST command the directory content is sent back to the client as a data connection; in other words, from a TCP point of view, LIST command is equivalent to a RETR commands of a file of few hundred bytes.

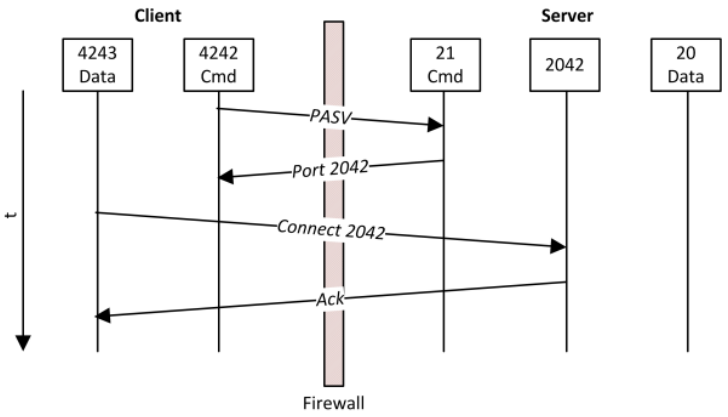
Conditions to reproduce and debug (code instrumentation, MQX 4.1.1)

- Probe free RAM when receiving a new FTP command: in file ftpsrv_task.c, function ftpsrv_read_cmd, around line 363 (after *session->cmd_arg = strtok(NULL, " ");*), capture and log free RAM with _mem_get_free()).
- Probe TCP socket shutdown: in file sstream.c, at the begining of function SOCK_STREAM_shutdown, log function parameters.
- Probe allocated TCB send and receive buffer pointers: in file tcp.c, function TCP_Clone_tcb, around line 1379 (after */* Clone the TCB */*)
- Probe released TCBs:
  - in file tcp_clos.c, function TCP_Close_TCB, probe tcb->sndringbuf and tcb->rcvringbuf around line 288 (after line *_mem_free(tcb->sndringbuf);*). NOTE: tcb->rcvringbuf is freed later in the function TCP_Free_TCB; probing it here is just convenient to remember the allocated rcv buffer address.
  - In file tcp_clos.c, function TCP_Free_TCB, probe tcb->rcvringbuf around line 67 (after line _mem_free(tcb->rcvringbuf);)
- Optional as previous is sufficient to understand the problem: Probe all memory allocation and free functions in lwmem.c (for small memory devices), then reconcile the allocated and the freed addresses to identify unreleased resources in logs. It demonstrates that memory leakage is related only to the cloned TCB resources.

Note: ensure to log time (in seconds).

FTP server behavior

- As illustrated, when receiving a FTP PASV command, the server is expected to respond to the client with the IP address and port on which to connect for the data connection. Then the client connects to this port and the server sends the DATA requested in the FTP command that follows the PASV command (in our case this command is either a LIST or a RETR command).



The way it is implemented in the RTCS stack (MQX 4.1.1):

- When receiving a PASV command, ftp server executes ftpsrv_pasv() function, which basic purpose is first to abort any previously opened data connection, second to open a listening stream socket on a free TCP port; it is done by the ftpsrv_open_passive_conn() function. This stream socket has option set so that receive and transmit buffer have FTPSRVCFG_RX_BUFFER_SIZE and FTPSRVCFG_TX_BUFFER_SIZE
- Then client sends a FTP request for transfering data (LIST or RETR) and connects to the socket for data connection. Server process the request and open the data connection on the listening socket by calling accept(). Immediate consequence of the “accept()” function is that TCP_clone_TCB function is executed which allocate a new TCB instance in memory for the data connection (size is FTPSRVCFG_RX_BUFFER_SIZE +FTPSRVCFG_TX_BUFFER_SIZE+a few bytes for TCB instance parameters).
- Then server sends the requested data. When completed, it acknowledges the FTP request, and then shutdowns the data socket to free the resources allocated to the data transmission (by calling shutdown (sock, FLAG_CLOSE_TX); where sock is the socket assigned for the data connection).

Log analysis (1/4)

```
receive new FTP session request on 0x20000d30, allocating 0x200010f0
---> FTPCMD IN: AUTH : TLS :::  RAM Free 131132 B - 50 p.c.
---> FTPCMD IN: AUTH : SSL :::  RAM Free 131132 B - 50 p.c.
---> FTPCMD IN: USER : Qualisteo  ::: RAM Free 131132 B - 50 p.c.
---> FTPCMD IN: PASS : Qualisteo :::  RAM Free 131100 B - 50 p.c.
---> FTPCMD IN: CWD : \DATA\POWER :::  RAM Free 131100 B - 50 p.c.
---> FTPCMD IN: TYPE : I :::  RAM Free 131084 B - 50 p.c.
---> FTPCMD IN: PASV :::  RAM Free 131084 B - 50 p.c.
1 ---> FTPCMD IN: LIST :::  RAM Free 123948 B - 47 p.c.
1 <1427120197> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
2 <1427120197> TCP_Clone_TCB S20010490 R20011010 cloned from socket 0x200011b0
2 <1427120197> TCP_Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\13_12_12 :::  RAM Free 123948 B - 47 p.c.
---> FTPCMD IN: PASV :::  RAM Free 123932 B - 47 p.c.
3 <1427120198> TCP_Socket shutdown sock: 0x200011b0 ABORT
3 <1427120198> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 117420 B - 44 p.c.
1 <1427120198> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
2 <1427120198> TCP_Clone_TCB S20011e60 R200129e0 cloned from socket 0x200011b0
2 <1427120198> TCP_Socket shutdown sock: 0x20001270 CLOSE
4 <1427120199> TCP_Close_TCB S20010490 R20011010 sock:0
4 <1427120199> TCP_Free_TCB R0x20011010
---> FTPCMD IN: CWD : \DATA\POWER\15_02_26 :::  RAM Free 123932 B - 47 p.c.
---> FTPCMD IN: PASV :::  RAM Free 123932 B - 47 p.c.
3 <1427120199> TCP_Socket shutdown sock: 0x200011b0 ABORT
3 <1427120199> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 117420 B - 44 p.c.
1 <1427120199> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
1 <1427120199> TCP_Clone_TCB S200104b0 R20011030 cloned from socket 0x200011b0
1 <1427120199> TCP Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\15_02_27 :::  RAM Free 117420 B - 44 p.c.
---> FTPCMD IN: PASV :::  RAM Free 117420 B - 44 p.c.
4 <1427120200> TCP Socket shutdown sock: 0x200011b0 ABORT
4 <1427120200> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 110908 B - 42 p.c.
1 <1427120200> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
1 <1427120200> TCP_Clone_TCB S200137d0 R20014350 cloned from socket 0x200011b0
1 <1427120200> TCP Socket shutdown sock: 0x20001270 CLOSE
4 <1427120200> TCP_Close_TCB S20011e60 R200129e0 sock: 0
4 <1427120200> TCP_Free_TCB R0x200129e0
---> FTPCMD IN: CWD : \DATA\POWER\15_02_28 :::  RAM Free 117420 B - 44 p.c.
---> FTPCMD IN: PASV :::  RAM Free 117420 B - 44 p.c.
<1427120201> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120201> TCP_Close_TCB S0 R0 sock: 200011b0
```

Colors indicates lines that are related to the same passive data connection. Intend is to easily catch reentries and delays

1. Client connects the data link socket. A cloned TCB is allocated for the data path

2. LIST data has been sent to the client, ftpsrv_list() releases the data link socket

3. New PASV command. The previous data link socket is killed. Immediate release of the TCB allocated to the socket for control path. TCB allocated to the data path is not rleased. A new listening socket is allocated (same address)

4. TCB allocated for the first data path (1) is released.

1. New list request, client connects to the new listening socket. A new cloned TCB is allocated for the data path

…

After reviewing the RTCS TCP code, the delay between (2) and (4) is explained by the need to wait for the TCP retransmit timeout (3 seconds) before releasing the TCB resource.
Shouldn't the TCB release be forced when the « grand parent » socket is killed (3)?

Log analysis (2/4)

```
---> FTPCMD IN: LIST :::  RAM Free 110908 B - 42 p.c.
<1427120201> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120201> TCP_Clone_TCB S20011e20 R200129a0 cloned from socket 0x200011b0
<1427120201> TCP Socket shutdown sock: 0x20001270 CLOSE
<1427120201> TCP_Close_TCB S200104b0 R20011030 sock: 0
<1427120201> TCP_Free_TCB R0x20011030
---> FTPCMD IN: CWD : \DATA\POWER\15_03_01 :::  RAM Free 117420 B - 44 p.c.
---> FTPCMD IN: PASV :::  RAM Free 117420 B - 44 p.c.
<1427120201> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120201> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 110908 B - 42 p.c.
<1427120201> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120201> TCP_Clone_TCB S200104e0 R20011060 cloned from socket 0x200011b0
<1427120202> TCP Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\15_03_02 :::  RAM Free 110908 B - 42 p.c.
---> FTPCMD IN: PASV :::  RAM Free 110908 B - 42 p.c.
<1427120202> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120202> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 104396 B - 39 p.c.
<1427120202> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120202> TCP_Clone_TCB S200151b0 R20015d30 cloned from socket 0x200011b0
<1427120202> TCP_Close_TCB S200137d0 R20014350 sock: 0
<1427120202> TCP_Free_TCB R0x20014350
<1427120202> TCP Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\15_03_03 :::  RAM Free 110908 B - 42 p.c.
---> FTPCMD IN: PASV :::  RAM Free 110908 B - 42 p.c.
<1427120203> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120203> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 104396 B - 39 p.c.
<1427120203> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120203> TCP_Clone_TCB S20013790 R20014310 cloned from socket 0x200011b0
<1427120203> TCP Socket shutdown sock: 0x20001270 CLOSE
<1427120203> TCP_Close_TCB S20011e20 R200129a0 sock: 0
<1427120203> TCP_Free_TCB R0x200129a0
---> FTPCMD IN: CWD : \DATA\POWER\15_03_04 :::  RAM Free 110908 B - 42 p.c.
---> FTPCMD IN: PASV :::  RAM Free 110908 B - 42 p.c.
<1427120203> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120203> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 104396 B - 39 p.c.
<1427120203> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120203> TCP_Clone_TCB S20011de0 R20012960 cloned from socket 0x200011b0
<1427120203> TCP Socket shutdown sock: 0x20001270 CLOSE
```

Log analysis (3/4)

```
<1427120204> TCP_Close_TCB S200104e0 R20011060 sock: 0
<1427120204> TCP_Free_TCB R0x20011060
---> FTPCMD IN: CWD : \DATA\POWER\15_03_05 :::  RAM Free 110908 B - 42 p.c.
---> FTPCMD IN: PASV :::  RAM Free 110908 B - 42 p.c.
<1427120204> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120204> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 104396 B - 39 p.c.
<1427120204> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120204> TCP_Clone_TCB S200104e0 R20011060 cloned from socket 0x200011b0
<1427120204> TCP Socket shutdown sock: 0x20001270 CLOSE
<1427120204> TCP_Close_TCB S200151b0 R20015d30 sock: 0
<1427120204> TCP_Free_TCB R0x20015d30
---> FTPCMD IN: CWD : \DATA\POWER\15_03_06 :::  RAM Free 110908 B - 42 p.c.
---> FTPCMD IN: PASV :::  RAM Free 110908 B - 42 p.c.
<1427120204> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120204> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 104396 B - 39 p.c.
<1427120204> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120204> TCP_Clone_TCB S20015170 R20015cf0 cloned from socket 0x200011b0
<1427120204> TCP Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\15_03_07 :::  RAM Free 104396 B - 39 p.c.
---> FTPCMD IN: PASV :::  RAM Free 104396 B - 39 p.c.
<1427120205> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120205> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 97884 B - 37 p.c.
<1427120205> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120205> TCP_Clone_TCB S20016ae0 R20017660 cloned from socket 0x200011b0
<1427120205> TCP_Close_TCB S20013790 R20014310 sock: 0
<1427120205> TCP_Free_TCB R0x20014310
<1427120205> TCP Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\15_03_08 :::  RAM Free 104396 B - 39 p.c.
---> FTPCMD IN: PASV :::  RAM Free 104396 B - 39 p.c.
<1427120205> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120205> TCP_Close_TCB S0 R0 sock: 200011b0
<1427120205> TCP_Close_TCB S20011de0 R20012960 sock: 0
<1427120205> TCP_Free_TCB R0x20012960
---> FTPCMD IN: LIST :::  RAM Free 104396 B - 39 p.c.
<1427120205> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120205> TCP_Clone_TCB S20011de0 R20012960 cloned from socket 0x200011b0
<1427120205> TCP Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\15_03_09 :::  RAM Free 104396 B - 39 p.c.
---> FTPCMD IN: PASV :::  RAM Free 104396 B - 39 p.c.
<1427120206> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120206> TCP_Close_TCB S0 R0 sock: 200011b0
```

Log analysis (4/4)

```
---> FTPCMD IN: LIST :::  RAM Free 97884 B - 37 p.c.
<1427120206> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120206> TCP_Clone_TCB S20013750 R200142d0 cloned from socket 0x200011b0
<1427120206> TCP_Close_TCB S200104e0 R20011060 sock: 0
<1427120206> TCP_Free_TCB R0x20011060
<1427120206> TCP Socket shutdown sock: 0x20001270 CLOSE
---> FTPCMD IN: CWD : \DATA\POWER\15_03_11 :::  RAM Free 104396 B - 39 p.c.
---> FTPCMD IN: PASV :::  RAM Free 104396 B - 39 p.c.
<1427120206> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120206> TCP_Close_TCB S0 R0 2000ffe0 200011b0
---> FTPCMD IN: LIST :::  RAM Free 97884 B - 37 p.c.
<1427120206> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120206> TCP_Clone_TCB S200104b0 R20011030 cloned from socket 0x200011b0
<1427120206> TCP_Close_TCB S20015170 R20015cf0 sock: 0
<1427120206> TCP_Free_TCB R0x20015cf0
<1427120206> TCP Socket shutdown sock: 0x20001270 CLOSE
<1427120207> TCP_Close_TCB S20016ae0 R20017660 sock: 0
<1427120207> TCP_Free_TCB R0x20017660
---> FTPCMD IN: CWD : \DATA\POWER\15_03_12 :::  RAM Free 110908 B - 42 p.c.
---> FTPCMD IN: PASV :::  RAM Free 110908 B - 42 p.c.
<1427120207> TCP Socket shutdown sock: 0x200011b0 ABORT
<1427120207> TCP_Close_TCB S0 R0 sock: 200011b0
---> FTPCMD IN: LIST :::  RAM Free 104396 B - 39 p.c.
<1427120207> FTP_accept_passive_conn listen_sock:0x200011b0 accept_sock:0x20001270
<1427120207> TCP_Clone_TCB S20015130 R20015cb0 cloned from socket 0x200011b0
<1427120207> TCP Socket shutdown sock: 0x20001270 CLOSE
<1427120207> TCP_Close_TCB S20011de0 R20012960 sock: 0
<1427120207> TCP_Free_TCB R0x20012960
<1427120208> TCP_Close_TCB S20013750 R200142d0 sock: 0
<1427120208> TCP_Free_TCB R0x200142d0
<1427120208> TCP_Close_TCB S200104b0 R20011030 sock: 0
<1427120208> TCP_Free_TCB R0x20011030
<1427120209> TCP_Close_TCB S20015130 R20015cb0 sock: 0
<1427120209> TCP_Free_TCB R0x20015cb0
Client disconnect
<1427120212> TCP Socket shutdown sock: 0x200010f0 CLOSE
<1427120212> TCP Socket shutdown sock: 0x200011b0 CLOSE
<1427120212> TCP_Close_TCB S2000eca0 R2000eeb0 sock: 0
<1427120212> TCP_Free_TCB R0x2000eeb0
<1427120212> TCP_Close_TCB S0 R0 sock: 200011b0
```

What is a proper way to solve it?

- How can this reentry be avoided?
- The retranmit timeout cannot be removed as it will violate the TCP protocol compliancy
- Adding a delay before answering the FTP request is not very smart and add unnecessary latency
- The best way looks to release the TCB when the parent socket is aborted. What is the proper way to do this?