torvalds / linux Public

Code Pull requests 312 Actions Projects Security Insights

master

linux / fs / cifs / cifs_debug.c

Steve French smb3: add dynamic trace points for tree disconnect … History

16 contributors +4

1048 lines (934 sloc) 30.3 KB

```
// SPDX-License-Identifier: GPL-2.0-or-later
/*
 *
 *   Copyright (C) International Business Machines  Corp., 2000,2005
 *
 *   Modified by Steve French (sfrench@us.ibm.com)
 */
#include <linux/fs.h>
#include <linux/string.h>
#include <linux/ctype.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>
#include "cifspdu.h"
#include "cifsglob.h"
#include "cifsproto.h"
#include "cifs_debug.h"
#include "cifsfs.h"
#include "fs_context.h"
#ifdef CONFIG_CIFS_DFS_UPCALL
#include "dfs_cache.h"
#endif
```

Follow

JoePerches

Committed to this repository

```
{
        pr_debug("%s: dump of %d bytes of data at 0x%p\n", label, length, data);
        print_hex_dump(KERN_DEBUG, "", DUMP_PREFIX_OFFSET, 16, 4,
                       data, length, true);
}

void cifs_dump_detail(void *buf, struct TCP_Server_Info *server)
{
#ifdef CONFIG_CIFS_DEBUG2
        struct smb_hdr *smb = buf;

        cifs_dbg(VFS, "Cmd: %d Err: 0x%x Flags: 0x%x Flgs2: 0x%x Mid: %d Pid: %d\n",
                 smb->Command, smb->Status.CifsError,
                 smb->Flags, smb->Flags2, smb->Mid, smb->Pid);
        cifs_dbg(VFS, "smb buf %p len %u\n", smb,
                 server->ops->calc_smb_size(smb));
#endif /* CONFIG_CIFS_DEBUG2 */
}

void cifs_dump_mids(struct TCP_Server_Info *server)
{
#ifdef CONFIG_CIFS_DEBUG2
        struct mid_q_entry *mid_entry;

        if (server == NULL)
                return;

        cifs_dbg(VFS, "Dump pending requests:\n");
        spin_lock(&server->mid_lock);
        list_for_each_entry(mid_entry, &server->pending_mid_q, qhead) {
                cifs_dbg(VFS, "State: %d Cmd: %d Pid: %d Cbdata: %p Mid %llu\n",
                         mid_entry->mid_state,
                         le16_to_cpu(mid_entry->command),
                         mid_entry->pid,
                         mid_entry->callback_data,
                         mid_entry->mid);
#ifdef CONFIG_CIFS_STATS2
                cifs_dbg(VFS, "IsLarge: %d buf: %p time rcv: %ld now: %ld\n",
                         mid_entry->large_buf,
                         mid_entry->resp_buf,
                         mid_entry->when_received,
                         jiffies);
#endif /* STATS2 */
                cifs_dbg(VFS, "IsMult: %d IsEnd: %d\n",
                         mid_entry->multiRsp, mid_entry->multiEnd);
                if (mid_entry->resp_buf) {
                        cifs_dump_detail(mid_entry->resp_buf, server);
                        cifs_dump_mem("existing buf: ",
                                mid_entry->resp_buf, 62);
                }
        }
        spin_unlock(&server->mid_lock);
#endif /* CONFIG_CIFS_DEBUG2 */
}

#ifdef CONFIG_PROC_FS
static void cifs_debug_tcon(struct seq_file *m, struct cifs_tcon *tcon)
{
        __u32 dev_type = le32_to_cpu(tcon->fsDevInfo.DeviceType);

        seq_printf(m, "%s Mounts: %d ", tcon->tree_name, tcon->tc_count);
        if (tcon->nativeFileSystem)
                seq_printf(m, "Type: %s ", tcon->nativeFileSystem);
        seq_printf(m, "DevInfo: 0x%x Attributes: 0x%x\n\tPathComponentMax: %d Status: %d",
                   le32_to_cpu(tcon->fsDevInfo.DeviceCharacteristics),
                   le32_to_cpu(tcon->fsAttrInfo.Attributes),
                   le32_to_cpu(tcon->fsAttrInfo.MaxPathNameComponentLength),
                   tcon->status);
        if (dev_type == FILE_DEVICE_DISK)
                seq_puts(m, " type: DISK ");
        else if (dev_type == FILE_DEVICE_CD_ROM)
                seq_puts(m, " type: CDROM ");
        else
                seq_printf(m, " type: %d ", dev_type);

        seq_printf(m, "Serial Number: 0x%x", tcon->vol_serial_number);

        if ((tcon->seal) ||
            (tcon->ses->session_flags & SMB2_SESSION_FLAG_ENCRYPT_DATA) ||
            (tcon->share_flags & SHI1005_FLAGS_ENCRYPT_DATA))
                seq_printf(m, " Encrypted");
        if (tcon->nocase)
                seq_printf(m, " nocase");
        if (tcon->unix_ext)
                seq_printf(m, " POSIX Extensions");
        if (tcon->ses->server->ops->dump_share_caps)
                tcon->ses->server->ops->dump_share_caps(m, tcon);
        if (tcon->use_witness)
                seq_puts(m, " Witness");
        if (tcon->broken_sparse_sup)
                seq_puts(m, " nosparse");
        if (tcon->need_reconnect)
                seq_puts(m, "\tDISCONNECTED ");
        seq_putc(m, '\n');
}

static void
cifs_dump_channel(struct seq_file *m, int i, struct cifs_chan *chan)
```

```
{
        struct TCP_Server_Info *server = chan->server;

        seq_printf(m, "\n\n\t\tChannel: %d ConnectionId: 0x%llx"
                   "\n\t\tNumber of credits: %d Dialect 0x%x"
                   "\n\t\tTCP status: %d Instance: %d"
                   "\n\t\tLocal Users To Server: %d SecMode: 0x%x Req On Wire: %d"
                   "\n\t\tIn Send: %d In MaxReq Wait: %d",
                   i+1, server->conn_id,
                   server->credits,
                   server->dialect,
                   server->tcpStatus,
                   server->reconnect_instance,
                   server->srv_count,
                   server->sec_mode,
                   in_flight(server),
                   atomic_read(&server->in_send),
                   atomic_read(&server->num_waiters));
}

static void
cifs_dump_iface(struct seq_file *m, struct cifs_server_iface *iface)
{
        struct sockaddr_in *ipv4 = (struct sockaddr_in *)&iface->sockaddr;
        struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)&iface->sockaddr;

        seq_printf(m, "\tSpeed: %zu bps\n", iface->speed);
        seq_puts(m, "\t\tCapabilities: ");
        if (iface->rdma_capable)
                seq_puts(m, "rdma ");
        if (iface->rss_capable)
                seq_puts(m, "rss ");
        seq_putc(m, '\n');
        if (iface->sockaddr.ss_family == AF_INET)
                seq_printf(m, "\t\tIPv4: %pI4\n", &ipv4->sin_addr);
        else if (iface->sockaddr.ss_family == AF_INET6)
                seq_printf(m, "\t\tIPv6: %pI6\n", &ipv6->sin6_addr);
        if (!iface->is_active)
                seq_puts(m, "\t\t[for-cleanup]\n");
}

static int cifs_debug_files_proc_show(struct seq_file *m, void *v)
{
        struct TCP_Server_Info *server;
        struct cifs_ses *ses;
        struct cifs_tcon *tcon;
        struct cifsFileInfo *cfile;

        seq_puts(m, "# Version:1\n");
        seq_puts(m, "# Format:\n");
        seq_puts(m, "# <tree id> <persistent fid> <flags> <count> <pid> <uid>");
#ifdef CONFIG_CIFS_DEBUG2
        seq_printf(m, " <filename> <mid>\n");
#else
        seq_printf(m, " <filename>\n");
#endif /* CIFS_DEBUG2 */
        spin_lock(&cifs_tcp_ses_lock);
        list_for_each_entry(server, &cifs_tcp_ses_list, tcp_ses_list) {
                list_for_each_entry(ses, &server->smb_ses_list, smb_ses_list) {
                        list_for_each_entry(tcon, &ses->tcon_list, tcon_list) {
                                spin_lock(&tcon->open_file_lock);
                                list_for_each_entry(cfile, &tcon->openFileList, tlist) {
                                        seq_printf(m,
                                                "0x%x 0x%llx 0x%x %d %d %d %pd",
                                                tcon->tid,
                                                cfile->fid.persistent_fid,
                                                cfile->f_flags,
                                                cfile->count,
                                                cfile->pid,
                                                from_kuid(&init_user_ns, cfile->uid),
                                                cfile->dentry);
#ifdef CONFIG_CIFS_DEBUG2
                                        seq_printf(m, " %llu\n", cfile->fid.mid);
#else
                                        seq_printf(m, "\n");
#endif /* CIFS_DEBUG2 */
                                }
                                spin_unlock(&tcon->open_file_lock);
                        }
                }
        }
        spin_unlock(&cifs_tcp_ses_lock);
        seq_putc(m, '\n');
        return 0;
}

static int cifs_debug_data_proc_show(struct seq_file *m, void *v)
{
        struct mid_q_entry *mid_entry;
        struct TCP_Server_Info *server;
        struct cifs_ses *ses;
        struct cifs_tcon *tcon;
        struct cifs_server_iface *iface;
        int c, i, j;

        seq_puts(m,
                    "Display Internal CIFS Data Structures for Debugging\n"
                    "---------------------------------------------------\n");
```

```c
        seq_printf(m, "CIFS Version %s\n", CIFS_VERSION);
        seq_printf(m, "Features:");
#ifdef CONFIG_CIFS_DFS_UPCALL
        seq_printf(m, " DFS");
#endif
#ifdef CONFIG_CIFS_FSCACHE
        seq_printf(m, ",FSCACHE");
#endif
#ifdef CONFIG_CIFS_SMB_DIRECT
        seq_printf(m, ",SMB_DIRECT");
#endif
#ifdef CONFIG_CIFS_STATS2
        seq_printf(m, ",STATS2");
#else
        seq_printf(m, ",STATS");
#endif
#ifdef CONFIG_CIFS_DEBUG2
        seq_printf(m, ",DEBUG2");
#elif defined(CONFIG_CIFS_DEBUG)
        seq_printf(m, ",DEBUG");
#endif
#ifdef CONFIG_CIFS_ALLOW_INSECURE_LEGACY
        seq_printf(m, ",ALLOW_INSECURE_LEGACY");
#endif
#ifdef CONFIG_CIFS_POSIX
        seq_printf(m, ",CIFS_POSIX");
#endif
#ifdef CONFIG_CIFS_UPCALL
        seq_printf(m, ",UPCALL(SPNEGO)");
#endif
#ifdef CONFIG_CIFS_XATTR
        seq_printf(m, ",XATTR");
#endif
        seq_printf(m, ",ACL");
#ifdef CONFIG_CIFS_SWN_UPCALL
        seq_puts(m, ",WITNESS");
#endif
        seq_putc(m, '\n');
        seq_printf(m, "CIFSMaxBufSize: %d\n", CIFSMaxBufSize);
        seq_printf(m, "Active VFS Requests: %d\n", GlobalTotalActiveXid);

        seq_printf(m, "\nServers: ");

        c = 0;
        spin_lock(&cifs_tcp_ses_lock);
        list_for_each_entry(server, &cifs_tcp_ses_list, tcp_ses_list) {
                /* channel info will be printed as a part of sessions below */
                if (CIFS_SERVER_IS_CHAN(server))
                        continue;

                c++;
                seq_printf(m, "\n%d) ConnectionId: 0x%llx ",
                        c, server->conn_id);

                if (server->hostname)
                        seq_printf(m, "Hostname: %s ", server->hostname);
#ifdef CONFIG_CIFS_SMB_DIRECT
                if (!server->rdma)
                        goto skip_rdma;

                if (!server->smbd_conn) {
                        seq_printf(m, "\nSMBDirect transport not available");
                        goto skip_rdma;
                }

                seq_printf(m, "\nSMBDirect (in hex) protocol version: %x "
                        "transport status: %x",
                        server->smbd_conn->protocol,
                        server->smbd_conn->transport_status);
                seq_printf(m, "\nConn receive_credit_max: %x "
                        "send_credit_target: %x max_send_size: %x",
                        server->smbd_conn->receive_credit_max,
                        server->smbd_conn->send_credit_target,
                        server->smbd_conn->max_send_size);
                seq_printf(m, "\nConn max_fragmented_recv_size: %x "
                        "max_fragmented_send_size: %x max_receive_size:%x",
                        server->smbd_conn->max_fragmented_recv_size,
                        server->smbd_conn->max_fragmented_send_size,
                        server->smbd_conn->max_receive_size);
                seq_printf(m, "\nConn keep_alive_interval: %x "
                        "max_readwrite_size: %x rdma_readwrite_threshold: %x",
                        server->smbd_conn->keep_alive_interval,
                        server->smbd_conn->max_readwrite_size,
                        server->smbd_conn->rdma_readwrite_threshold);
                seq_printf(m, "\nDebug count_get_receive_buffer: %x "
                        "count_put_receive_buffer: %x count_send_empty: %x",
                        server->smbd_conn->count_get_receive_buffer,
                        server->smbd_conn->count_put_receive_buffer,
                        server->smbd_conn->count_send_empty);
                seq_printf(m, "\nRead Queue count_reassembly_queue: %x "
                        "count_enqueue_reassembly_queue: %x "
                        "count_dequeue_reassembly_queue: %x "
                        "fragment_reassembly_remaining: %x "
                        "reassembly_data_length: %x "
                        "reassembly_queue_length: %x",
                        server->smbd_conn->count_reassembly_queue,
                        server->smbd_conn->count_enqueue_reassembly_queue,
                        server->smbd_conn->count_dequeue_reassembly_queue,
                        server->smbd_conn->fragment_reassembly_remaining,
                        server->smbd_conn->reassembly_data_length,
                        server->smbd_conn->reassembly_queue_length);
                seq_printf(m, "\nCurrent Credits send_credits: %x "
                        "receive_credits: %x receive_credit_target: %x",
                        atomic_read(&server->smbd_conn->send_credits),
                        atomic_read(&server->smbd_conn->receive_credits),
                        server->smbd_conn->receive_credit_target);
                seq_printf(m, "\nPending send_pending: %x ",
                        atomic_read(&server->smbd_conn->send_pending));
                seq_printf(m, "\nReceive buffers count_receive_queue: %x "
                        "count_empty_packet_queue: %x",
                        server->smbd_conn->count_receive_queue,
                        server->smbd_conn->count_empty_packet_queue);
                seq_printf(m, "\nMR responder_resources: %x "
                        "max_frmr_depth: %x mr_type: %x",
                        server->smbd_conn->responder_resources,
                        server->smbd_conn->max_frmr_depth,
                        server->smbd_conn->mr_type);
                seq_printf(m, "\nMR mr_ready_count: %x mr_used_count: %x",
                        atomic_read(&server->smbd_conn->mr_ready_count),
                        atomic_read(&server->smbd_conn->mr_used_count));
skip_rdma:
#endif
                seq_printf(m, "\nNumber of credits: %d Dialect 0x%x",
                        server->credits,  server->dialect);
                if (server->compress_algorithm == SMB3_COMPRESS_LZNT1)
                        seq_printf(m, " COMPRESS_LZNT1");
                else if (server->compress_algorithm == SMB3_COMPRESS_LZ77)
                        seq_printf(m, " COMPRESS_LZ77");
                else if (server->compress_algorithm == SMB3_COMPRESS_LZ77_HUFF)
                        seq_printf(m, " COMPRESS_LZ77_HUFF");
                if (server->sign)
                        seq_printf(m, " signed");
                if (server->posix_ext_supported)
                        seq_printf(m, " posix");
                if (server->nosharesock)
                        seq_printf(m, " nosharesock");

                if (server->rdma)
                        seq_printf(m, "\nRDMA ");
                seq_printf(m, "\nTCP status: %d Instance: %d"
                                "\nLocal Users To Server: %d SecMode: 0x%x Req On Wire: %d",
                                server->tcpStatus,
                                server->reconnect_instance,
                                server->srv_count,
                                server->sec_mode, in_flight(server));

                seq_printf(m, "\nIn Send: %d In MaxReq Wait: %d",
                                atomic_read(&server->in_send),
                                atomic_read(&server->num_waiters));

                seq_printf(m, "\n\n\tSessions: ");
                i = 0;
                list_for_each_entry(ses, &server->smb_ses_list, smb_ses_list) {
                        i++;
                        if ((ses->serverDomain == NULL) ||
                                (ses->serverOS == NULL) ||
                                (ses->serverNOS == NULL)) {
                                seq_printf(m, "\n\t%d) Address: %s Uses: %d Capability: 0x%x\tSess
                                        i, ses->ip_addr, ses->ses_count,
                                        ses->capabilities, ses->ses_status);
                                if (ses->session_flags & SMB2_SESSION_FLAG_IS_GUEST)
                                        seq_printf(m, "Guest ");
                                else if (ses->session_flags & SMB2_SESSION_FLAG_IS_NULL)
                                        seq_printf(m, "Anonymous ");
                        } else {
                                seq_printf(m,
                                    "\n\t%d) Name: %s  Domain: %s Uses: %d OS: %s "
                                    "\n\tNOS: %s\tCapability: 0x%x"
                                        "\n\tSMB session status: %d ",
                                i, ses->ip_addr, ses->serverDomain,
                                ses->ses_count, ses->serverOS, ses->serverNOS,
                                ses->capabilities, ses->ses_status);
                        }

                        seq_printf(m, "\n\tSecurity type: %s ",
                                get_security_type_str(server->ops->select_sectype(server, ses->sec

                        /* dump session id helpful for use with network trace */
                        seq_printf(m, " SessionId: 0x%llx", ses->Suid);
                        if (ses->session_flags & SMB2_SESSION_FLAG_ENCRYPT_DATA)
                                seq_puts(m, " encrypted");
                        if (ses->sign)
                                seq_puts(m, " signed");

                        seq_printf(m, "\n\tUser: %d Cred User: %d",
                                   from_kuid(&init_user_ns, ses->linux_uid),
                                   from_kuid(&init_user_ns, ses->cred_uid));

                        spin_lock(&ses->chan_lock);
                        if (CIFS_CHAN_NEEDS_RECONNECT(ses, 0))
                                seq_puts(m, "\tPrimary channel: DISCONNECTED ");
                        if (CIFS_CHAN_IN_RECONNECT(ses, 0))
                                seq_puts(m, "\t[RECONNECTING] ");

                        if (ses->chan_count > 1) {
                                seq_printf(m, "\n\n\tExtra Channels: %zu ",
                                           ses->chan_count-1);
                                for (j = 1; j < ses->chan_count; j++) {
                                        cifs_dump_channel(m, j, &ses->chans[j]);
                                        if (CIFS_CHAN_NEEDS_RECONNECT(ses, j))
                                                seq_puts(m, "\tDISCONNECTED ");
                                        if (CIFS_CHAN_IN_RECONNECT(ses, j))
                                                seq_puts(m, "\t[RECONNECTING] ");
                                }
                        }
                        spin_unlock(&ses->chan_lock);

                        seq_puts(m, "\n\n\tShares: ");
                        j = 0;

                        seq_printf(m, "\n\t%d) IPC: ", j);
                        if (ses->tcon_ipc)
                                cifs_debug_tcon(m, ses->tcon_ipc);
                        else
                                seq_puts(m, "none\n");

                        list_for_each_entry(tcon, &ses->tcon_list, tcon_list) {
                                ++j;
                                seq_printf(m, "\n\t%d) ", j);
                                cifs_debug_tcon(m, tcon);
                        }

                        spin_lock(&ses->iface_lock);
                        if (ses->iface_count)
                                seq_printf(m, "\n\n\tServer interfaces: %zu",
                                           ses->iface_count);
                        j = 0;
                        list_for_each_entry(iface, &ses->iface_list,
                                                 iface_head) {
                                seq_printf(m, "\n\t%d)", ++j);
                                cifs_dump_iface(m, iface);
                                if (is_ses_using_iface(ses, iface))
                                        seq_puts(m, "\t\t[CONNECTED]\n");
                        }
                        spin_unlock(&ses->iface_lock);
                }
                if (i == 0)
                        seq_printf(m, "\n\t\t[NONE]");

                seq_puts(m, "\n\n\tMIDs: ");
                spin_lock(&server->mid_lock);
                list_for_each_entry(mid_entry, &server->pending_mid_q, qhead) {
                        seq_printf(m, "\n\tState: %d com: %d pid:"
                                        " %d cbdata: %p mid %llu\n",
                                        mid_entry->mid_state,
                                        le16_to_cpu(mid_entry->command),
                                        mid_entry->pid,
                                        mid_entry->callback_data,
                                        mid_entry->mid);
                }
                spin_unlock(&server->mid_lock);
                seq_printf(m, "\n--\n");
        }
        if (c == 0)
                seq_printf(m, "\n\t[NONE]");

        spin_unlock(&cifs_tcp_ses_lock);
        seq_putc(m, '\n');
        cifs_swn_dump(m);

        /* BB add code to dump additional info such as TCP session info now */
        return 0;
}

static ssize_t cifs_stats_proc_write(struct file *file,
                const char __user *buffer, size_t count, loff_t *ppos)
{
        bool bv;
        int rc;
        struct TCP_Server_Info *server;
        struct cifs_ses *ses;
        struct cifs_tcon *tcon;

        rc = kstrtobool_from_user(buffer, count, &bv);
        if (rc == 0) {
#ifdef CONFIG_CIFS_STATS2
                int i;

                atomic_set(&total_buf_alloc_count, 0);
                atomic_set(&total_small_buf_alloc_count, 0);
#endif /* CONFIG_CIFS_STATS2 */
                atomic_set(&tcpSesReconnectCount, 0);
                atomic_set(&tconInfoReconnectCount, 0);

                spin_lock(&GlobalMid_Lock);
                GlobalMaxActiveXid = 0;
                GlobalCurrentXid = 0;
                spin_unlock(&GlobalMid_Lock);
                spin_lock(&cifs_tcp_ses_lock);
                list_for_each_entry(server, &cifs_tcp_ses_list, tcp_ses_list) {
                        server->max_in_flight = 0;
#ifdef CONFIG_CIFS_STATS2
                        for (i = 0; i < NUMBER_OF_SMB2_COMMANDS; i++) {
                                atomic_set(&server->num_cmds[i], 0);
                                atomic_set(&server->smb2slowcmd[i], 0);
                                server->time_per_cmd[i] = 0;
                                server->slowest_cmd[i] = 0;
                                server->fastest_cmd[0] = 0;
                        }
#endif /* CONFIG_CIFS_STATS2 */
                        list_for_each_entry(ses, &server->smb_ses_list, smb_ses_list) {
                                list_for_each_entry(tcon, &ses->tcon_list, tcon_list) {
                                        atomic_set(&tcon->num_smbs_sent, 0);
                                        spin_lock(&tcon->stat_lock);
                                        tcon->bytes_read = 0;
                                        tcon->bytes_written = 0;
                                        spin_unlock(&tcon->stat_lock);
                                        if (server->ops->clear_stats)
                                                server->ops->clear_stats(tcon);
                                }
                        }
                }
                spin_unlock(&cifs_tcp_ses_lock);
        } else {
                return rc;
        }

        return count;
}

static int cifs_stats_proc_show(struct seq_file *m, void *v)
{
        int i;
#ifdef CONFIG_CIFS_STATS2
        int j;
#endif /* STATS2 */
        struct TCP_Server_Info *server;
        struct cifs_ses *ses;
        struct cifs_tcon *tcon;

        seq_printf(m, "Resources in use\nCIFS Session: %d\n",
                        sesInfoAllocCount.counter);
        seq_printf(m, "Share (unique mount targets): %d\n",
                        tconInfoAllocCount.counter);
        seq_printf(m, "SMB Request/Response Buffer: %d Pool size: %d\n",
                        buf_alloc_count.counter,
                        cifs_min_rcv + tcpSesAllocCount.counter);
        seq_printf(m, "SMB Small Req/Resp Buffer: %d Pool size: %d\n",
                        small_buf_alloc_count.counter, cifs_min_small);
#ifdef CONFIG_CIFS_STATS2
        seq_printf(m, "Total Large %d Small %d Allocations\n",
                                atomic_read(&total_buf_alloc_count),
                                atomic_read(&total_small_buf_alloc_count));
```

```c
#endif /* CONFIG_CIFS_STATS2 */

        seq_printf(m, "Operations (MIDs): %d\n", atomic_read(&mid_count));
        seq_printf(m,
                "\n%d session %d share reconnects\n",
                tcpSesReconnectCount.counter, tconInfoReconnectCount.counter);

        seq_printf(m,
                "Total vfs operations: %d maximum at one time: %d\n",
                GlobalCurrentXid, GlobalMaxActiveXid);

        i = 0;
        spin_lock(&cifs_tcp_ses_lock);
        list_for_each_entry(server, &cifs_tcp_ses_list, tcp_ses_list) {
                seq_printf(m, "\nMax requests in flight: %d", server->max_in_flight);
#ifdef CONFIG_CIFS_STATS2
                seq_puts(m, "\nTotal time spent processing by command. Time ");
                seq_printf(m, "units are jiffies (%d per second)\n", HZ);
                seq_puts(m, "  SMB3 CMD\tNumber\tTotal Time\tFastest\tSlowest\n");
                seq_puts(m, "  --------\t------\t----------\t-------\t-------\n");
                for (j = 0; j < NUMBER_OF_SMB2_COMMANDS; j++)
                        seq_printf(m, "  %d\t\t%d\t%llu\t\t%u\t%u\n", j,
                                atomic_read(&server->num_cmds[j]),
                                server->time_per_cmd[j],
                                server->fastest_cmd[j],
                                server->slowest_cmd[j]);
                for (j = 0; j < NUMBER_OF_SMB2_COMMANDS; j++)
                        if (atomic_read(&server->smb2slowcmd[j]))
                                seq_printf(m, "  %d slow responses from %s for command %d\n",
                                        atomic_read(&server->smb2slowcmd[j]),
                                        server->hostname, j);
#endif /* STATS2 */
                list_for_each_entry(ses, &server->smb_ses_list, smb_ses_list) {
                        list_for_each_entry(tcon, &ses->tcon_list, tcon_list) {
                                i++;
                                seq_printf(m, "\n%d) %s", i, tcon->tree_name);
                                if (tcon->need_reconnect)
                                        seq_puts(m, "\tDISCONNECTED ");
                                seq_printf(m, "\nSMBs: %d",
                                           atomic_read(&tcon->num_smbs_sent));
                                if (server->ops->print_stats)
                                        server->ops->print_stats(m, tcon);
                        }
                }
        }
        spin_unlock(&cifs_tcp_ses_lock);

        seq_putc(m, '\n');
        return 0;
```

```
}

static int cifs_stats_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, cifs_stats_proc_show, NULL);
}

static const struct proc_ops cifs_stats_proc_ops = {
        .proc_open      = cifs_stats_proc_open,
        .proc_read      = seq_read,
        .proc_lseek     = seq_lseek,
        .proc_release   = single_release,
        .proc_write     = cifs_stats_proc_write,
};

#ifdef CONFIG_CIFS_SMB_DIRECT
#define PROC_FILE_DEFINE(name) \
static ssize_t name##_write(struct file *file, const char __user *buffer, \
        size_t count, loff_t *ppos) \
{ \
        int rc; \
        rc = kstrtoint_from_user(buffer, count, 10, & name); \
        if (rc) \
                return rc; \
        return count; \
} \
static int name##_proc_show(struct seq_file *m, void *v) \
{ \
        seq_printf(m, "%d\n", name ); \
        return 0; \
} \
static int name##_open(struct inode *inode, struct file *file) \
{ \
        return single_open(file, name##_proc_show, NULL); \
} \
\
static const struct proc_ops cifs_##name##_proc_fops = { \
        .proc_open      = name##_open, \
        .proc_read      = seq_read, \
        .proc_lseek     = seq_lseek, \
        .proc_release   = single_release, \
        .proc_write     = name##_write, \
}

PROC_FILE_DEFINE(rdma_readwrite_threshold);
PROC_FILE_DEFINE(smbd_max_frmr_depth);
PROC_FILE_DEFINE(smbd_keep_alive_interval);
PROC_FILE_DEFINE(smbd_max_receive_size);
PROC_FILE_DEFINE(smbd_max_fragmented_recv_size);
PROC_FILE_DEFINE(smbd_max_send_size);
PROC_FILE_DEFINE(smbd_send_credit_target);
PROC_FILE_DEFINE(smbd_receive_credit_max);
#endif

static struct proc_dir_entry *proc_fs_cifs;
static const struct proc_ops cifsFYI_proc_ops;
static const struct proc_ops cifs_lookup_cache_proc_ops;
static const struct proc_ops traceSMB_proc_ops;
static const struct proc_ops cifs_security_flags_proc_ops;
static const struct proc_ops cifs_linux_ext_proc_ops;
static const struct proc_ops cifs_mount_params_proc_ops;

void
cifs_proc_init(void)
{
        proc_fs_cifs = proc_mkdir("fs/cifs", NULL);
        if (proc_fs_cifs == NULL)
                return;

        proc_create_single("DebugData", 0, proc_fs_cifs,
                        cifs_debug_data_proc_show);

        proc_create_single("open_files", 0400, proc_fs_cifs,
                        cifs_debug_files_proc_show);

        proc_create("Stats", 0644, proc_fs_cifs, &cifs_stats_proc_ops);
        proc_create("cifsFYI", 0644, proc_fs_cifs, &cifsFYI_proc_ops);
        proc_create("traceSMB", 0644, proc_fs_cifs, &traceSMB_proc_ops);
        proc_create("LinuxExtensionsEnabled", 0644, proc_fs_cifs,
                    &cifs_linux_ext_proc_ops);
        proc_create("SecurityFlags", 0644, proc_fs_cifs,
                    &cifs_security_flags_proc_ops);
        proc_create("LookupCacheEnabled", 0644, proc_fs_cifs,
                    &cifs_lookup_cache_proc_ops);

        proc_create("mount_params", 0444, proc_fs_cifs, &cifs_mount_params_proc_ops);

#ifdef CONFIG_CIFS_DFS_UPCALL
        proc_create("dfscache", 0644, proc_fs_cifs, &dfscache_proc_ops);
#endif

#ifdef CONFIG_CIFS_SMB_DIRECT
        proc_create("rdma_readwrite_threshold", 0644, proc_fs_cifs,
                &cifs_rdma_readwrite_threshold_proc_fops);
        proc_create("smbd_max_frmr_depth", 0644, proc_fs_cifs,
                &cifs_smbd_max_frmr_depth_proc_fops);
        proc_create("smbd_keep_alive_interval", 0644, proc_fs_cifs,
                &cifs_smbd_keep_alive_interval_proc_fops);
```

```c
        proc_create("smbd_max_receive_size", 0644, proc_fs_cifs,
                &cifs_smbd_max_receive_size_proc_fops);
        proc_create("smbd_max_fragmented_recv_size", 0644, proc_fs_cifs,
                &cifs_smbd_max_fragmented_recv_size_proc_fops);
        proc_create("smbd_max_send_size", 0644, proc_fs_cifs,
                &cifs_smbd_max_send_size_proc_fops);
        proc_create("smbd_send_credit_target", 0644, proc_fs_cifs,
                &cifs_smbd_send_credit_target_proc_fops);
        proc_create("smbd_receive_credit_max", 0644, proc_fs_cifs,
                &cifs_smbd_receive_credit_max_proc_fops);
#endif
}

void
cifs_proc_clean(void)
{
        if (proc_fs_cifs == NULL)
                return;

        remove_proc_entry("DebugData", proc_fs_cifs);
        remove_proc_entry("open_files", proc_fs_cifs);
        remove_proc_entry("cifsFYI", proc_fs_cifs);
        remove_proc_entry("traceSMB", proc_fs_cifs);
        remove_proc_entry("Stats", proc_fs_cifs);
        remove_proc_entry("SecurityFlags", proc_fs_cifs);
        remove_proc_entry("LinuxExtensionsEnabled", proc_fs_cifs);
        remove_proc_entry("LookupCacheEnabled", proc_fs_cifs);
        remove_proc_entry("mount_params", proc_fs_cifs);

#ifdef CONFIG_CIFS_DFS_UPCALL
        remove_proc_entry("dfscache", proc_fs_cifs);
#endif
#ifdef CONFIG_CIFS_SMB_DIRECT
        remove_proc_entry("rdma_readwrite_threshold", proc_fs_cifs);
        remove_proc_entry("smbd_max_frmr_depth", proc_fs_cifs);
        remove_proc_entry("smbd_keep_alive_interval", proc_fs_cifs);
        remove_proc_entry("smbd_max_receive_size", proc_fs_cifs);
        remove_proc_entry("smbd_max_fragmented_recv_size", proc_fs_cifs);
        remove_proc_entry("smbd_max_send_size", proc_fs_cifs);
        remove_proc_entry("smbd_send_credit_target", proc_fs_cifs);
        remove_proc_entry("smbd_receive_credit_max", proc_fs_cifs);
#endif
        remove_proc_entry("fs/cifs", NULL);
}

static int cifsFYI_proc_show(struct seq_file *m, void *v)
{
        seq_printf(m, "%d\n", cifsFYI);
        return 0;
```

```
}

static int cifsFYI_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, cifsFYI_proc_show, NULL);
}

static ssize_t cifsFYI_proc_write(struct file *file, const char __user *buffer,
                size_t count, loff_t *ppos)
{
        char c[2] = { '\0' };
        bool bv;
        int rc;

        rc = get_user(c[0], buffer);
        if (rc)
                return rc;
        if (strtobool(c, &bv) == 0)
                cifsFYI = bv;
        else if ((c[0] > '1') && (c[0] <= '9'))
                cifsFYI = (int) (c[0] - '0'); /* see cifs_debug.h for meanings */
        else
                return -EINVAL;

        return count;
}

static const struct proc_ops cifsFYI_proc_ops = {
        .proc_open      = cifsFYI_proc_open,
        .proc_read      = seq_read,
        .proc_lseek     = seq_lseek,
        .proc_release   = single_release,
        .proc_write     = cifsFYI_proc_write,
};

static int cifs_linux_ext_proc_show(struct seq_file *m, void *v)
{
        seq_printf(m, "%d\n", linuxExtEnabled);
        return 0;
}

static int cifs_linux_ext_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, cifs_linux_ext_proc_show, NULL);
}

static ssize_t cifs_linux_ext_proc_write(struct file *file,
                const char __user *buffer, size_t count, loff_t *ppos)
{
        int rc;

        rc = kstrtobool_from_user(buffer, count, &linuxExtEnabled);
        if (rc)
                return rc;

        return count;
}

static const struct proc_ops cifs_linux_ext_proc_ops = {
        .proc_open      = cifs_linux_ext_proc_open,
        .proc_read      = seq_read,
        .proc_lseek     = seq_lseek,
        .proc_release   = single_release,
        .proc_write     = cifs_linux_ext_proc_write,
};

static int cifs_lookup_cache_proc_show(struct seq_file *m, void *v)
{
        seq_printf(m, "%d\n", lookupCacheEnabled);
        return 0;
}

static int cifs_lookup_cache_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, cifs_lookup_cache_proc_show, NULL);
}

static ssize_t cifs_lookup_cache_proc_write(struct file *file,
                const char __user *buffer, size_t count, loff_t *ppos)
{
        int rc;

        rc = kstrtobool_from_user(buffer, count, &lookupCacheEnabled);
        if (rc)
                return rc;

        return count;
}

static const struct proc_ops cifs_lookup_cache_proc_ops = {
        .proc_open      = cifs_lookup_cache_proc_open,
        .proc_read      = seq_read,
        .proc_lseek     = seq_lseek,
        .proc_release   = single_release,
        .proc_write     = cifs_lookup_cache_proc_write,
};

static int traceSMB_proc_show(struct seq_file *m, void *v)
```

```
{
        seq_printf(m, "%d\n", traceSMB);
        return 0;
}

static int traceSMB_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, traceSMB_proc_show, NULL);
}

static ssize_t traceSMB_proc_write(struct file *file, const char __user *buffer,
                size_t count, loff_t *ppos)
{
        int rc;

        rc = kstrtobool_from_user(buffer, count, &traceSMB);
        if (rc)
                return rc;

        return count;
}

static const struct proc_ops traceSMB_proc_ops = {
        .proc_open      = traceSMB_proc_open,
        .proc_read      = seq_read,
        .proc_lseek     = seq_lseek,
        .proc_release   = single_release,
        .proc_write     = traceSMB_proc_write,
};

static int cifs_security_flags_proc_show(struct seq_file *m, void *v)
{
        seq_printf(m, "0x%x\n", global_secflags);
        return 0;
}

static int cifs_security_flags_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, cifs_security_flags_proc_show, NULL);
}

/*
 * Ensure that if someone sets a MUST flag, that we disable all other MAY
 * flags except for the ones corresponding to the given MUST flag. If there are
 * multiple MUST flags, then try to prefer more secure ones.
 */
static void
cifs_security_flags_handle_must_flags(unsigned int *flags)
{
        unsigned int signflags = *flags & CIFSSEC_MUST_SIGN;

        if ((*flags & CIFSSEC_MUST_KRB5) == CIFSSEC_MUST_KRB5)
                *flags = CIFSSEC_MUST_KRB5;
        else if ((*flags & CIFSSEC_MUST_NTLMSSP) == CIFSSEC_MUST_NTLMSSP)
                *flags = CIFSSEC_MUST_NTLMSSP;
        else if ((*flags & CIFSSEC_MUST_NTLMV2) == CIFSSEC_MUST_NTLMV2)
                *flags = CIFSSEC_MUST_NTLMV2;

        *flags |= signflags;
}

static ssize_t cifs_security_flags_proc_write(struct file *file,
                const char __user *buffer, size_t count, loff_t *ppos)
{
        int rc;
        unsigned int flags;
        char flags_string[12];
        bool bv;

        if ((count < 1) || (count > 11))
                return -EINVAL;

        memset(flags_string, 0, 12);

        if (copy_from_user(flags_string, buffer, count))
                return -EFAULT;

        if (count < 3) {
                /* single char or single char followed by null */
                if (strtobool(flags_string, &bv) == 0) {
                        global_secflags = bv ? CIFSSEC_MAX : CIFSSEC_DEF;
                        return count;
                } else if (!isdigit(flags_string[0])) {
                        cifs_dbg(VFS, "Invalid SecurityFlags: %s\n",
                                        flags_string);
                        return -EINVAL;
                }
        }

        /* else we have a number */
        rc = kstrtouint(flags_string, 0, &flags);
        if (rc) {
                cifs_dbg(VFS, "Invalid SecurityFlags: %s\n",
                                flags_string);
                return rc;
        }

        cifs_dbg(FYI, "sec flags 0x%x\n", flags);

        if (flags == 0)  {
                cifs_dbg(VFS, "Invalid SecurityFlags: %s\n", flags_string);
                return -EINVAL;
        }

        if (flags & ~CIFSSEC_MASK) {
                cifs_dbg(VFS, "Unsupported security flags: 0x%x\n",
                         flags & ~CIFSSEC_MASK);
                return -EINVAL;
        }

        cifs_security_flags_handle_must_flags(&flags);

        /* flags look ok - update the global security flags for cifs module */
        global_secflags = flags;
        if (global_secflags & CIFSSEC_MUST_SIGN) {
                /* requiring signing implies signing is allowed */
                global_secflags |= CIFSSEC_MAY_SIGN;
                cifs_dbg(FYI, "packet signing now required\n");
        } else if ((global_secflags & CIFSSEC_MAY_SIGN) == 0) {
                cifs_dbg(FYI, "packet signing disabled\n");
        }
        /* BB should we turn on MAY flags for other MUST options? */
        return count;
}

static const struct proc_ops cifs_security_flags_proc_ops = {
        .proc_open      = cifs_security_flags_proc_open,
        .proc_read      = seq_read,
        .proc_lseek     = seq_lseek,
        .proc_release   = single_release,
        .proc_write     = cifs_security_flags_proc_write,
};

/* To make it easier to debug, can help to show mount params */
static int cifs_mount_params_proc_show(struct seq_file *m, void *v)
{
        const struct fs_parameter_spec *p;
        const char *type;

        for (p = smb3_fs_parameters; p->name; p++) {
                /* cannot use switch with pointers... */
                if (!p->type) {
                        if (p->flags == fs_param_neg_with_no)
                                type = "noflag";
                        else
                                type = "flag";
                } else if (p->type == fs_param_is_bool)
```

```
                        type = "bool";
                else if (p->type == fs_param_is_u32)
                        type = "u32";
                else if (p->type == fs_param_is_u64)
                        type = "u64";
                else if (p->type == fs_param_is_string)
                        type = "string";
                else
                        type = "unknown";

                seq_printf(m, "%s:%s\n", p->name, type);
        }

        return 0;
}

static int cifs_mount_params_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, cifs_mount_params_proc_show, NULL);
}

static const struct proc_ops cifs_mount_params_proc_ops = {
        .proc_open      = cifs_mount_params_proc_open,
        .proc_read      = seq_read,
        .proc_lseek     = seq_lseek,
        .proc_release   = single_release,
        /* No need for write for now */
        /* .proc_write  = cifs_mount_params_proc_write, */
};

#else
inline void cifs_proc_init(void)
{
}

inline void cifs_proc_clean(void)
{
}
#endif /* PROC_FS */
```