```
**********************************************************
   21005 Thu Jan 17 14:31:31 2019
new/usr/src/lib/libmlrpc/common/ndr_server.c
10103 libmlrpc needs smatch fixes
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * Server side RPC handler.
 */

#include <sys/byteorder.h>
#include <sys/uio.h>
#include <errno.h>
#include <synch.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <thread.h>

#include <libmlrpc.h>

#define NDR_PIPE_SEND(np, buf, len) \
        ((np)->np_send)((np), (buf), (len))
#define NDR_PIPE_RECV(np, buf, len) \
        ((np)->np_recv)((np), (buf), (len))

static int ndr_svc_process(ndr_xa_t *);
static int ndr_svc_bind(ndr_xa_t *);
static int ndr_svc_request(ndr_xa_t *);
static void ndr_reply_prepare_hdr(ndr_xa_t *);
static int ndr_svc_alter_context(ndr_xa_t *);
static void ndr_reply_fault(ndr_xa_t *, unsigned long);

static int ndr_recv_request(ndr_xa_t *mxa);
static int ndr_recv_frag(ndr_xa_t *mxa);
static int ndr_send_reply(ndr_xa_t *);

static int ndr_pipe_process(ndr_pipe_t *, ndr_xa_t *);

/*
 * External entry point called by smbd.
```



```
 */
void
ndr_pipe_worker(ndr_pipe_t *np)
{
        ndr_xa_t        *mxa;
        int rc;

        ndr_svc_binding_pool_init(&np->np_binding, np->np_binding_pool,
            NDR_N_BINDING_POOL);

        if ((mxa = malloc(sizeof (*mxa))) == NULL)
                return;

        do {
                bzero(mxa, sizeof (*mxa));
                rc = ndr_pipe_process(np, mxa);
        } while (rc == 0);

        free(mxa);

        /*
         * Ensure that there are no RPC service policy handles
         * (associated with this fid) left around.
         */
        ndr_hdclose(np);
}
```

*unchanged_portion_omitted_*

```
/*
 * Signal an RPC fault. The stream is reset and we overwrite whatever
 * was in the response header with the fault information.
 */
static void
ndr_reply_fault(ndr_xa_t *mxa, unsigned long drc)
{
        ndr_common_header_t *rhdr = &mxa->recv_hdr.common_hdr;
        ndr_common_header_t *hdr = &mxa->send_hdr.common_hdr;
        ndr_stream_t *nds = &mxa->send_nds;
        unsigned long fault_status;

        (void) NDS_RESET(nds);
        NDS_RESET(nds);

        hdr->rpc_vers = 5;
        hdr->rpc_vers_minor = 0;
        hdr->pfc_flags = NDR_PFC_FIRST_FRAG + NDR_PFC_LAST_FRAG;
        hdr->packed_drep = rhdr->packed_drep;
        hdr->frag_length = sizeof (mxa->send_hdr.fault_hdr);
        hdr->auth_length = 0;
        hdr->call_id = rhdr->call_id;
#ifdef _BIG_ENDIAN
        hdr->packed_drep.intg_char_rep = NDR_REPLAB_CHAR_ASCII
            | NDR_REPLAB_INTG_BIG_ENDIAN;
#else
        hdr->packed_drep.intg_char_rep = NDR_REPLAB_CHAR_ASCII
            | NDR_REPLAB_INTG_LITTLE_ENDIAN;
#endif

        switch (drc & NDR_DRC_MASK_SPECIFIER) {
        case NDR_DRC_FAULT_OUT_OF_MEMORY:
        case NDR_DRC_FAULT_ENCODE_TOO_BIG:
                fault_status = NDR_FAULT_NCA_OUT_ARGS_TOO_BIG;
                break;

        case NDR_DRC_FAULT_REQUEST_PCONT_INVALID:
                fault_status = NDR_FAULT_NCA_INVALID_PRES_CONTEXT_ID;
```

```
669                 break;

671         case NDR_DRC_FAULT_REQUEST_OPNUM_INVALID:
672                 fault_status = NDR_FAULT_NCA_OP_RNG_ERROR;
673                 break;

675         case NDR_DRC_FAULT_DECODE_FAILED:
676         case NDR_DRC_FAULT_ENCODE_FAILED:
677                 fault_status = NDR_FAULT_NCA_PROTO_ERROR;
678                 break;

680         default:
681                 fault_status = NDR_FAULT_NCA_UNSPEC_REJECT;
682                 break;
683         }

685         mxa->send_hdr.fault_hdr.common_hdr.ptype = NDR_PTYPE_FAULT;
686         mxa->send_hdr.fault_hdr.status = fault_status;
687         mxa->send_hdr.response_hdr.alloc_hint = hdr->frag_length;
688 }
```

*unchanged_portion_omitted_*