

```

//Editor-Info: *- C++ *-
//
//Subject: Calypso-IP
//
//File: routeadapter.cpp
//
//Version: $Revision: 1.2 $
//
//State: $State: Exp $
//
//Date: $Date: 2000/07/31 08:58:57 $
//
//Organisation:
//    Helsinki University of Technology
//    Laboratory of Telecommunications Software and Multimedia
//
//Author:
//    Juha Hyttinen
//
//Description:
//    See header file.
//
//Copyright:
//    Copyright 1999 Helsinki University of Technology
//    ALL RIGHTS RESERVED BETWEEN JUNE 1999 AND JANUARY 2002.
//
//Licence:
//
//
//History:
//
//
#include "routeadapter.h"

routeAdapter* routeAdapter :: _only = 0;

routeAdapter* routeAdapter :: instance(void)
{
    if ( _only == 0 )
    {
        _only = new routeAdapter(4096);
    }
    return _only;
}

routeAdapter* routeAdapter :: instance(pfUlong pduSize_)
{
    if ( _only == 0 )
    {
        _only = new routeAdapter(pduSize_);
    }
    return _only;
}

```

```

//
//Function: routeAdapter
//
//Description:
// Constructor. Makes new adapter to system route table.
// Given parameter is buffer size
//

routeAdapter :: routeAdapter(pfulong pduSize_)
: pfNETLINKsocket(pduSize_)
{
    _seq = 1;
    _requestBuffer.clear();
    _requestBufferLock = false;
    _lockFlag = false;

    return;
}

routeAdapter :: ~routeAdapter(void)
{
#ifdef DEBUG
    cout << "Routeadapter destructor..." << endl;
#endif // DEBUG
    return;
}

//
//Function: createNETLINKConnection
//
//Description:
// Creates connection to system routetable
//

void routeAdapter :: createNETLINKConnection(int netlinkFamily_)
{
    if ( ! this->openDevice(netlinkFamily_) )
    {
#ifdef DEBUG
        debugUser("Fatal error in openDevice, exiting...");
#endif // DEBUG
        exit(1);
    }
    this->setUsageOfAcknowledge(true);
    this->readDevice();
    return ;
}

//
//Function: setLockFlag
//
//Description:
// If lock is setted true, will this adapter NOT make new read requests

```

```

//
void routeAdapter :: setLockFlag(pfBoolean value_)
{
    _lockFlag = value_;
    return;
}

//
//Function: getLockFlag
//
//Description:
//    Get current value of lock
//

pfBoolean routeAdapter :: getLockFlag(void)
{
    return _lockFlag;
}

//
//Function: addRoute
//
//Description:
//    Add route to system route table. Uses buffer to make call.
//

pfBoolean routeAdapter :: addRoute(int family_,pfUlong dst_,
                                   pfUlong mask_, pfUlong gw_,
                                   pfUlong metric_,pfUlong priority_,
                                   pfUlong oif_, pfByte scope_)
{
    // Even we use directly addRoute, we have to commit also others
    // witch are all ready in _requestBuffer
    addRouteBuffer(family_,dst_, mask_, gw_, metric_, priority_, oif_, scope_);

    if (commitBuffer() == false)
    {
        debugUser("routeAdapter: addRoute: In this method call");
        return false;
    }

    return true;
}

//
//Function: addRouteBuffer
//
//Description:
//    Add "addroute" command to buffer,
//    which is committed with function commitBuffer
//

void routeAdapter :: addRouteBuffer(int family_,pfUlong dst_,

```

```

        pfUlong mask_, pfUlong gw_,
        pfUlong metric_,pfUlong priority_,
        pfUlong oif_, pfByte scope_)

    {
        pfFrame frame;

        frame = this->parseFrame(family_,RTM_NEWROUTE,dst_, mask_, gw_, metric_,
                                priority_, oif_, scope_);

        _requestBuffer.putLast(frame);
        return;
    }

    //
    //Function: delRoute
    //
    //Description:
    //    Delete route from system route table. Uses buffer to make call.
    //
    pfBoolean routeAdapter :: delRoute(int family_,pfUlong dst_,
        pfUlong mask_, pfUlong gw_,
        pfUlong metric_,pfUlong priority_,
        pfUlong oif_, pfByte scope_)
    {
        // Even we use directly delRoute, we have to commit also others,
        // which are at ready in _requestBuffer
        delRouteBuffer(family_, dst_, mask_, gw_, metric_, priority_, oif_, scope_);

        if (commitBuffer() == false)
        {
            debugUser("routeAdapter: delRoute: In this method call");
            return false;
        }

        return true;
    }

    //
    //Function: delRouteBuffer
    //
    //Description:
    //    Add "delroute" command to buffer,
    //    which is committed with function commitBuffer
    //
    void routeAdapter :: delRouteBuffer(int family_,pfUlong dst_,
        pfUlong mask_, pfUlong gw_,
        pfUlong metric_,pfUlong priority_,
        pfUlong oif_, pfByte scope_)
    {
        pfFrame frame;

        frame = this->parseFrame(family_,

```

```

        RTM_DELROUTE,
        dst_,
        mask_,
        gw_,
        metric_,
        priority_,
        oif_,
        scope_);

        _requestBuffer.putLast(frame);
        return;
    }

    //
    //Function: getRouteTable
    //
    //Description:
    //    Get routes from system route table.
    //    This is writed directly to socket
    //    and also received information.
    //

void routeAdapter :: getRouteTable(int family_)
{
    pfFrame frame = this->parseFrame(family_,
        RTM_GETROUTE,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0);

    // Lets make the sending directly
    this->clearReadRequest();

    // Make sure tha,t after we send direct read request,
    // we won't do read request again.
    this->setLockFlag(true);
    this->setStatusPolling();

    if (writeDevice(frame) != 0)
    {
        debugUser("routeAdapter: getRouteTable: Device is busy for write request");
        return;
    }

    // Get info
    if (this->readDevice() != 0)
    {
        debugUser("routeAdapter: getRouteTable: Device is busy for read results ???");
        this->clearStatusPolling();
    }
}

```

```

        this->readDevice();
        return;
    }

    // Get message done frame!
    if (this->readDevice() != 0)
    {
        debugUser("routeAdapter: getRouteTable: Device is busy for read results ?????");
        this->clearStatusPolling();
        this->readDevice();
        return;
    }

    this->setLockFlag(false);
    this->clearStatusPolling();
    this->readDevice();

    return;
}

//Function: readInterfaces
//
//Description:
//    Gets link and address information
//

void routeAdapter :: readInterfaces(void)
{
    // Lets make the sending directly
    this->clearReadRequest();

    // Make sure that after we send direct read request,
    // we won't do read request again.
    this->setLockFlag(true);
    this->setStatusPolling();

#ifdef __DEBUG_USER__
    cout << endl << "GET LINK" << endl;
#endif //__DEBUG_USER__

    pFrame frame;
    frame.clear();

    frame.putLast16bit(RTM_GETLINK);
    frame.putLast16bit(0);
    frame.putLast32bit(111);
    frame.putLast32bit(0);
    frame.putLast32bit(5);
    frame.putLast32bit(AF_INET);

    this->writeDevice(frame);
    this->readDevice();
}

```

```

// Get message done frame?
this->readDevice();

#ifdef __DEBUG_USER__
    cout << endl << "GET ADDR" << endl;
#endif //__DEBUG_USER__
    frame.clear();

    frame.putLast16bit(RTM_GETADDR);
    frame.putLast16bit(0);
    frame.putLast32bit(112);
    frame.putLast32bit(0);
    frame.putLast32bit(5);
    frame.putLast32bit(AF_INET);

    this->writeDevice(frame);
    this->readDevice();
    // Get message done frame?
    this->readDevice();

    this->setLockFlag(false);
    // Set polling to normal stage
    this->clearStatusPolling();
    // Send read request to scheduler
    this->readDevice();

    return;
}

//
//Function: parseFrame
//
//Description:
//    Parses frame for request and uses given values to make it.
//    Here is also checked what is netlink type, and depending
//    what kind it is, is request frame for that family maded
//
pfFrame routeAdapter :: parseFrame(int family_, int frametype_,
    pfUlong dst_, pfUlong mask_,
    pfUlong gw_, pfUlong metric_,
    pfUlong priority_, pfUlong oif_,
    pfByte scope_)
{
    pfFrame frame;
    frame.clear();

    frame.putLast16bit(frametype_);
    frame.putLast16bit(0); // nl_flags
    frame.putLast32bit(getNextSeq()); // Seq
    frame.putLast32bit(0); // Pid
    frame.putLast32bit(12); // length
    frame.putLast32bit(family_);

```

```

frame.putLast(0);
frame.putLast(scope_);
frame.putLast(0);
frame.putLast(0);

// This adapter not yet use IPv6 ,
// even pfNETLINKsocket offers it
switch(family_)
{
    case AF_INET:
        frame.putLast32bit(dst_);
        frame.putLast32bit(gw_);
        frame.putLast32bit(mask_);
        frame.putLast32bit(metric_);
        frame.putLast32bit(priority_);
        frame.putLast32bit(oif_);
        break;
    default:
        break;
}

frame.putLast(0); // Flags
// update length
frame.write32bit(frame.length(),12);

return frame;
}

//
//Function: commitBuffer
//
//Description:
//    Commits all request which are in requestbuffer.
//    This is writed directly to socket and also received information.
//
pfBoolean routeAdapter :: commitBuffer(void)
{
    if (_requestBuffer.length() <= 0)
    {
        debugUser("routeAdapter: commitBuffer: Buffer is empty");
        return false;
    }

    // Lets make the sending directly
    this->clearReadRequest();
    // Make sure tha,t after we send direct read request,
    // we won't do read request again.
    this->setLockFlag(true);
    this->setStatusPolling();

    if (writeDevice(_requestBuffer) != 0)
    {
        debugUser("routeAdapter: commitBuffer: Device is busy, request will be sendd at next time");
    }
}

```



```

this->setLockFlag(false);
this->clearStatusPolling();
this->readDevice();
return false;
}

_requestBuffer.clear();

// Commit buffer sends add/del which will
// send acknowledgement if setted so
if (this->isUsingAcknowledge() == true)
{
    this->readDevice();
}

this->setLockFlag(false);
this->clearStatusPolling();
this->readDevice();

return true;
}

//Function: clearReadRequest
//
//Description:
//    Clear requests for read.
//    This effects to ovops++ io-scheduler
//
void routeAdapter :: clearReadRequest(void)
{
    this->cancelRead();
    _readPending = 0;
    return;
}

//
//Function: getNextSeq
//
//Description:
//    Returns next free sequency nro for request
//
pfUlong routeAdapter :: getNextSeq(void)
{
    if (_seq >= 0XFFFFFFFE)
    {
        _seq = 2 ;
    }

    // _seqList.push_back(_seq);
    return _seq++;
}

```

```

}

//
//Function: readAction
//
//Description:
//    When something is readed from kernel, this method will handles data
//

void routeAdapter :: readAction(pfFrame &frame_, pfUlong code_)
{
#ifdef __DEBUG_USER__
    if(!_debug != 0)
    {
        debugUser("NEW PACKET FROM NETLINK");
        debugPfUlong("Code", code_);
    }
#endif //__DEBUG_USER__

    unsigned int count=0;
    pfFrame    tempFrame;

    // While there is information in frame
    while(count < frame_.length())
    {
        // Lets parse next information (single entry)
        // from frame to subframe
        tempFrame.clear();

        for (unsigned int i = 0; i < frame_.read32bit(count + 12); i++)
        {
            tempFrame.putLast32bit(frame_.read32bit(count + i*4));
        }

        // Case it's IPv6 frame let's continue...
        if (tempFrame.read32bit(16) == AF_INET6)
        {
            debugFrame("IPv6 frame", tempFrame);
            count += 4 * tempFrame.read32bit(12);
            continue;
        }

        if ( tempFrame.read32bit(4) == 0 )
        {
            debugUser("Kernel information", tempFrame);
        }
        else if ((tempFrame.read16bit(0) >= RTM_NEWROUTE) &&
            (tempFrame.read16bit(0) <= RTM_GETROUTE))
        {
            debugFrame("ROUTE FRAME", tempFrame);
        }
        else if ( (tempFrame.read16bit(0) >= RTM_NEWLINK) &&
            (tempFrame.read16bit(0) <= RTM_GETLINK) )
    }
}

```

```

{
    debugFrame("LINK FRAME",tempFrame);
}
else if ( (tempFrame.read16bit(0) >= RTM_NEWADDR) &&
(tempFrame.read16bit(0) <= RTM_GETADDR) )
{
    debugFrame("ADDRESS FRAME",tempFrame);
}
else if (tempFrame.read16bit(0) == NLMMSG_ERROR)
{
    debugString("NLMMSG_ERROR", strerror(tempFrame.read16bit(16)));
}
// Case message done is received we remove that sequency
// number from lists
else if (tempFrame.read16bit(0) == NLMMSG_DONE)
{
    debugPfUlong("NLMMSG_DONE:",tempFrame.read32bit(4));
}
// We won't know what frame contains so we skip it.
else
{
    debugUser("Not known information...skipped");
}

count += 4 * tempFrame.read32bit(12);
}

// If given frame is empty there is somekind error
// occurred when frame is parsed. This is not
// the same thing as error frame informs.
if(frame_.length() <= 0)
{
    debugUser("Not received information from netlink. Maybe error.");
}

// Case lockflag is setted we won't make new read request.
if (this->getLockFlag() != 0)
{
    return;
}
else
{
    this->readDevice();
}

return;
}

//
//Function: writeAction
//
//Description:
//    When something is writed to kernel, this method will be called

```

```
//
void routeAdapter :: writeAction(pfUlong code_)
{
#ifdef DEBUG
    if(!_debug != 0)
    {
        debugUser("writeAction TO NETLINK");
        debugPfUlong("Code", code_);
    }
#endif //DEBUG
    return;
}
```