Introduction

This document describes how to use the dynamic debug (ddebug) feature.

Dynamic debug is designed to allow you to dynamically enable/disable kernel code to obtain additional kernel information. Currently, if CONFIG_DYNAMIC_DEBUG is set, then all pr_debug()/dev_debug() calls can be dynamically enabled per-callsite.

Dynamic debug has even more useful features:

 * Simple query language allows turning on and off debugging statements by matching any combination of:

   - source filename
   - function name
   - line number (including ranges of line numbers)
   - module name
   - format string

 * Provides a debugfs control file: <debugfs>/dynamic_debug/control which can be read to display the complete list of known debug statements, to help guide you

Controlling dynamic debug Behaviour

The behaviour of pr_debug()/dev_debug()s are controlled via writing to a control file in the 'debugfs' filesystem. Thus, you must first mount the debugfs filesystem, in order to make use of this feature. Subsequently, we refer to the control file as: <debugfs>/dynamic_debug/control. For example, if you want to enable printing from source file 'svcsock.c', line 1603 you simply do:

```
nullarbor:~ # echo 'file svcsock.c line 1603 +p' >
                              <debugfs>/dynamic_debug/control
```

If you make a mistake with the syntax, the write will fail thus:

```
nullarbor:~ # echo 'file svcsock.c wtf 1 +p' >
                              <debugfs>/dynamic_debug/control
-bash: echo: write error: Invalid argument
```

Viewing Dynamic Debug Behaviour

You can view the currently configured behaviour of all the debug statements via:

```
nullarbor:~ # cat <debugfs>/dynamic_debug/control
# filename:lineno [module]function flags format
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.c:323
[svcxprt_rdma]svc_rdma_cleanup - "SVCRDMA Module Removed, deregister RPC RDMA
transport\012"
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.c:341
```

```
[svcxprt_rdma]svc_rdma_init - "\011max_inline       : %d\012"
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.c:340
[svcxprt_rdma]svc_rdma_init - "\011sq_depth         : %d\012"
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svc_rdma.c:338
[svcxprt_rdma]svc_rdma_init - "\011max_requests     : %d\012"
...
```

You can also apply standard Unix text manipulation filters to this data, e.g.

```
nullarbor:~ # grep -i rdma <debugfs>/dynamic_debug/control  | wc -l
62

nullarbor:~ # grep -i tcp <debugfs>/dynamic_debug/control | wc -l
42
```

Note in particular that the third column shows the enabled behaviour flags for each debug statement callsite (see below for definitions of the flags). The default value, no extra behaviour enabled, is "-". So you can view all the debug statement callsites with any non-default flags:

```
nullarbor:~ # awk '$3 != "-"' <debugfs>/dynamic_debug/control
# filename:lineno [module]function flags format
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svcsock.c:1603
[sunrpc]svc_send p "svc_process: st_sendto returned %d\012"
```

Command Language Reference

At the lexical level, a command comprises a sequence of words separated by whitespace characters. Note that newlines are treated as word separators and do *not* end a command or allow multiple commands to be done together. So these are all equivalent:

```
nullarbor:~ # echo -c 'file svcsock.c line 1603 +p' >
                              <debugfs>/dynamic_debug/control
nullarbor:~ # echo -c '  file   svcsock.c     line  1603 +p  ' >
                              <debugfs>/dynamic_debug/control
nullarbor:~ # echo -c 'file svcsock.c\nline 1603 +p' >
                              <debugfs>/dynamic_debug/control
nullarbor:~ # echo -n 'file svcsock.c line 1603 +p' >
                              <debugfs>/dynamic_debug/control
```

Commands are bounded by a write() system call. If you want to do multiple commands you need to do a separate "echo" for each, like:

```
nullarbor:~ # echo 'file svcsock.c line 1603 +p' > /proc/dprintk ;\
> echo 'file svcsock.c line 1563 +p' > /proc/dprintk
```

or even like:

```
nullarbor:~ # (
> echo 'file svcsock.c line 1603 +p' ;\
> echo 'file svcsock.c line 1563 +p' ;\
```

>) > /proc/dprintk

At the syntactical level, a command comprises a sequence of match specifications, followed by a flags change specification.

command ::= match-spec* flags-spec

The match-spec's are used to choose a subset of the known dprintk() callsites to which to apply the flags-spec. Think of them as a query with implicit ANDs between each pair. Note that an empty list of match-specs is possible, but is not very useful because it will not match any debug statement callsites.

A match specification comprises a keyword, which controls the attribute of the callsite to be compared, and a value to compare against. Possible keywords are:

```
match-spec ::= 'func' string |
               'file' string |
               'module' string |
               'format' string |
               'line' line-range

line-range ::= lineno |
               '-'lineno |
               lineno'-' |
               lineno'-'lineno
// Note: line-range cannot contain space, e.g.
// "1-30" is valid range but "1 - 30" is not.

lineno ::= unsigned-int
```

The meanings of each keyword are:

func
    The given string is compared against the function name
    of each callsite. Example:

    func svc_tcp_accept

file
    The given string is compared against either the full
    pathname or the basename of the source file of each
    callsite. Examples:

    file svcsock.c
    file
/usr/src/packages/BUILD/sgi-enhancednfs-1.4/default/net/sunrpc/svcsock.c

module
    The given string is compared against the module name
    of each callsite. The module name is the string as
    seen in "lsmod", i.e. without the directory or the .ko
    suffix and with '-' changed to '_'. Examples:

    module sunrpc

```
    module nfsd

format
    The given string is searched for in the dynamic debug format
    string.  Note that the string does not need to match the
    entire format, only some part.  Whitespace and other
    special characters can be escaped using C octal character
    escape \ooo notation, e.g. the space character is \040.
    Alternatively, the string can be enclosed in double quote
    characters (") or single quote characters (').
    Examples:

    format svcrdma:         // many of the NFS/RDMA server dprintks
    format readahead        // some dprintks in the readahead cache
    format nfsd:\040SETATTR // one way to match a format with whitespace
    format "nfsd: SETATTR"  // a neater way to match a format with whitespace
    format 'nfsd: SETATTR'  // yet another way to match a format with whitespace

line
    The given line number or range of line numbers is compared
    against the line number of each dprintk() callsite.  A single
    line number matches the callsite line number exactly.  A
    range of line numbers matches any callsite between the first
    and last line number inclusive.  An empty first number means
    the first line in the file, an empty line number means the
    last number in the file.  Examples:

    line 1603       // exactly line 1603
    line 1600-1605  // the six lines from line 1600 to line 1605
    line -1605      // the 1605 lines from line 1 to line 1605
    line 1600-      // all lines from line 1600 to the end of the file

The flags specification comprises a change operation followed
by one or more flag characters.  The change operation is one
of the characters:

-
    remove the given flags

+
    add the given flags

=
    set the flags to the given flags

The flags are:

p
    Causes a printk() message to be emitted to dmesg

Note the regexp ^[-+=][scp]+$ matches a flags specification.
Note also that there is no convenient syntax to remove all
the flags at once, you need to use "-psc".

Examples
========
```

```
// enable the message at line 1603 of file svcsock.c
nullarbor:~ # echo -n 'file svcsock.c line 1603 +p' >
                                <debugfs>/dynamic_debug/control

// enable all the messages in file svcsock.c
nullarbor:~ # echo -n 'file svcsock.c +p' >
                                <debugfs>/dynamic_debug/control

// enable all the messages in the NFS server module
nullarbor:~ # echo -n 'module nfsd +p' >
                                <debugfs>/dynamic_debug/control

// enable all 12 messages in the function svc_process()
nullarbor:~ # echo -n 'func svc_process +p' >
                                <debugfs>/dynamic_debug/control

// disable all 12 messages in the function svc_process()
nullarbor:~ # echo -n 'func svc_process -p' >
                                <debugfs>/dynamic_debug/control

// enable messages for NFS calls READ, READLINK, READDIR and READDIR+.
nullarbor:~ # echo -n 'format "nfsd: READ" +p' >
                                <debugfs>/dynamic_debug/control
```