```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
        "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" []>

<book id="debug-objects-guide">
 <bookinfo>
  <title>Debug objects life time</title>

  <authorgroup>
   <author>
    <firstname>Thomas</firstname>
    <surname>Gleixner</surname>
    <affiliation>
     <address>
      <email>tglx@linutronix.de</email>
     </address>
    </affiliation>
   </author>
  </authorgroup>

  <copyright>
   <year>2008</year>
   <holder>Thomas Gleixner</holder>
  </copyright>

  <legalnotice>
   <para>
     This documentation is free software; you can redistribute
     it and/or modify it under the terms of the GNU General Public
     License version 2 as published by the Free Software Foundation.
   </para>

   <para>
     This program is distributed in the hope that it will be
     useful, but WITHOUT ANY WARRANTY; without even the implied
     warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
     See the GNU General Public License for more details.
   </para>

   <para>
     You should have received a copy of the GNU General Public
     License along with this program; if not, write to the Free
     Software Foundation, Inc., 59 Temple Place, Suite 330, Boston,
     MA 02111-1307 USA
   </para>

   <para>
     For more details see the file COPYING in the source
     distribution of Linux.
   </para>
  </legalnotice>
 </bookinfo>

<toc></toc>

  <chapter id="intro">
  <title>Introduction</title>
  <para>
    debugobjects is a generic infrastructure to track the life time
    of kernel objects and validate the operations on those.
  </para>
  <para>
    debugobjects is useful to check for the following error patterns:
      <itemizedlist>
        <listitem><para>Activation of uninitialized objects</para></listitem>
        <listitem><para>Initialization of active objects</para></listitem>
        <listitem><para>Usage of freed/destroyed objects</para></listitem>
      </itemizedlist>
  </para>
  <para>
    debugobjects is not changing the data structure of the real
    object so it can be compiled in with a minimal runtime impact
    and enabled on demand with a kernel command line option.
  </para>
</chapter>

<chapter id="howto">
  <title>Howto use debugobjects</title>
  <para>
    A kernel subsystem needs to provide a data structure which
    describes the object type and add calls into the debug code at
    appropriate places. The data structure to describe the object
    type needs at minimum the name of the object type. Optional
    functions can and should be provided to fixup detected problems
    so the kernel can continue to work and the debug information can
    be retrieved from a live system instead of hard core debugging
    with serial consoles and stack trace transcripts from the
    monitor.
  </para>
  <para>
    The debug calls provided by debugobjects are:
    <itemizedlist>
      <listitem><para>debug_object_init</para></listitem>
      <listitem><para>debug_object_init_on_stack</para></listitem>
      <listitem><para>debug_object_activate</para></listitem>
      <listitem><para>debug_object_deactivate</para></listitem>
      <listitem><para>debug_object_destroy</para></listitem>
      <listitem><para>debug_object_free</para></listitem>
    </itemizedlist>
    Each of these functions takes the address of the real object and
    a pointer to the object type specific debug description
    structure.
  </para>
  <para>
    Each detected error is reported in the statistics and a limited
    number of errors are printk'ed including a full stack trace.
  </para>
  <para>
    The statistics are available via /sys/kernel/debug/debug_objects/stats.
    They provide information about the number of warnings and the
    number of successful fixups along with information about the
    usage of the internal tracking objects and the state of the
```

```
      internal tracking objects pool.
    </para>
  </chapter>
  <chapter id="debugfunctions">
    <title>Debug functions</title>
    <sect1 id="prototypes">
      <title>Debug object function reference</title>
!Elib/debugobjects.c
    </sect1>
    <sect1 id="debug_object_init">
      <title>debug_object_init</title>
      <para>
        This function is called whenever the initialization function
        of a real object is called.
      </para>
      <para>
        When the real object is already tracked by debugobjects it is
        checked, whether the object can be initialized.  Initializing
        is not allowed for active and destroyed objects. When
        debugobjects detects an error, then it calls the fixup_init
        function of the object type description structure if provided
        by the caller. The fixup function can correct the problem
        before the real initialization of the object happens. E.g. it
        can deactivate an active object in order to prevent damage to
        the subsystem.
      </para>
      <para>
        When the real object is not yet tracked by debugobjects,
        debugobjects allocates a tracker object for the real object
        and sets the tracker object state to ODEBUG_STATE_INIT. It
        verifies that the object is not on the callers stack. If it is
        on the callers stack then a limited number of warnings
        including a full stack trace is printk’ed. The calling code
        must use debug_object_init_on_stack() and remove the object
        before leaving the function which allocated it. See next
        section.
      </para>
    </sect1>

    <sect1 id="debug_object_init_on_stack">
      <title>debug_object_init_on_stack</title>
      <para>
        This function is called whenever the initialization function
        of a real object which resides on the stack is called.
      </para>
      <para>
        When the real object is already tracked by debugobjects it is
        checked, whether the object can be initialized. Initializing
        is not allowed for active and destroyed objects. When
        debugobjects detects an error, then it calls the fixup_init
        function of the object type description structure if provided
        by the caller. The fixup function can correct the problem
        before the real initialization of the object happens. E.g. it
        can deactivate an active object in order to prevent damage to
        the subsystem.
      </para>
  <para>
    When the real object is not yet tracked by debugobjects
    debugobjects allocates a tracker object for the real object
    and sets the tracker object state to ODEBUG_STATE_INIT. It
    verifies that the object is on the callers stack.
  </para>
  <para>
    An object which is on the stack must be removed from the
    tracker by calling debug_object_free() before the function
    which allocates the object returns. Otherwise we keep track of
    stale objects.
  </para>
</sect1>

<sect1 id="debug_object_activate">
  <title>debug_object_activate</title>
  <para>
    This function is called whenever the activation function of a
    real object is called.
  </para>
  <para>
    When the real object is already tracked by debugobjects it is
    checked, whether the object can be activated.  Activating is
    not allowed for active and destroyed objects. When
    debugobjects detects an error, then it calls the
    fixup_activate function of the object type description
    structure if provided by the caller. The fixup function can
    correct the problem before the real activation of the object
    happens. E.g. it can deactivate an active object in order to
    prevent damage to the subsystem.
  </para>
  <para>
    When the real object is not yet tracked by debugobjects then
    the fixup_activate function is called if available. This is
    necessary to allow the legitimate activation of statically
    allocated and initialized objects. The fixup function checks
    whether the object is valid and calls the debug_objects_init()
    function to initialize the tracking of this object.
  </para>
  <para>
    When the activation is legitimate, then the state of the
    associated tracker object is set to ODEBUG_STATE_ACTIVE.
  </para>
</sect1>

<sect1 id="debug_object_deactivate">
  <title>debug_object_deactivate</title>
  <para>
    This function is called whenever the deactivation function of
    a real object is called.
  </para>
  <para>
    When the real object is tracked by debugobjects it is checked,
    whether the object can be deactivated. Deactivating is not
    allowed for untracked or destroyed objects.
  </para>
```

```
    <para>
      When the deactivation is legitimate, then the state of the
      associated tracker object is set to ODEBUG_STATE_INACTIVE.
    </para>
  </sect1>

  <sect1 id="debug_object_destroy">
    <title>debug_object_destroy</title>
    <para>
      This function is called to mark an object destroyed. This is
      useful to prevent the usage of invalid objects, which are
      still available in memory: either statically allocated objects
      or objects which are freed later.
    </para>
    <para>
      When the real object is tracked by debugobjects it is checked,
      whether the object can be destroyed. Destruction is not
      allowed for active and destroyed objects. When debugobjects
      detects an error, then it calls the fixup_destroy function of
      the object type description structure if provided by the
      caller. The fixup function can correct the problem before the
      real destruction of the object happens. E.g. it can deactivate
      an active object in order to prevent damage to the subsystem.
    </para>
    <para>
      When the destruction is legitimate, then the state of the
      associated tracker object is set to ODEBUG_STATE_DESTROYED.
    </para>
  </sect1>

  <sect1 id="debug_object_free">
    <title>debug_object_free</title>
    <para>
      This function is called before an object is freed.
    </para>
    <para>
      When the real object is tracked by debugobjects it is checked,
      whether the object can be freed. Free is not allowed for
      active objects. When debugobjects detects an error, then it
      calls the fixup_free function of the object type description
      structure if provided by the caller. The fixup function can
      correct the problem before the real free of the object
      happens. E.g. it can deactivate an active object in order to
      prevent damage to the subsystem.
    </para>
    <para>
      Note that debug_object_free removes the object from the
      tracker. Later usage of the object is detected by the other
      debug checks.
    </para>
  </sect1>
</chapter>
<chapter id="fixupfunctions">
  <title>Fixup functions</title>
  <sect1 id="debug_obj_descr">
    <title>Debug object type description structure</title>
```

```
!Iinclude/linux/debugobjects.h
    </sect1>
    <sect1 id="fixup_init">
      <title>fixup_init</title>
      <para>
        This function is called from the debug code whenever a problem
        in debug_object_init is detected. The function takes the
        address of the object and the state which is currently
        recorded in the tracker.
      </para>
      <para>
        Called from debug_object_init when the object state is:
        <itemizedlist>
          <listitem><para>ODEBUG_STATE_ACTIVE</para></listitem>
        </itemizedlist>
      </para>
      <para>
        The function returns 1 when the fixup was successful,
        otherwise 0. The return value is used to update the
        statistics.
      </para>
      <para>
        Note, that the function needs to call the debug_object_init()
        function again, after the damage has been repaired in order to
        keep the state consistent.
      </para>
    </sect1>

    <sect1 id="fixup_activate">
      <title>fixup_activate</title>
      <para>
        This function is called from the debug code whenever a problem
        in debug_object_activate is detected.
      </para>
      <para>
        Called from debug_object_activate when the object state is:
        <itemizedlist>
          <listitem><para>ODEBUG_STATE_NOTAVAILABLE</para></listitem>
          <listitem><para>ODEBUG_STATE_ACTIVE</para></listitem>
        </itemizedlist>
      </para>
      <para>
        The function returns 1 when the fixup was successful,
        otherwise 0. The return value is used to update the
        statistics.
      </para>
      <para>
        Note that the function needs to call the debug_object_activate()
        function again after the damage has been repaired in order to
        keep the state consistent.
      </para>
      <para>
        The activation of statically initialized objects is a special
        case. When debug_object_activate() has no tracked object for
        this object address then fixup_activate() is called with
        object state ODEBUG_STATE_NOTAVAILABLE. The fixup function
        needs to check whether this is a legitimate case of a
        statically initialized object or not. In case it is it calls
        debug_object_init() and debug_object_activate() to make the
        object known to the tracker and marked active. In this case
        the function should return 0 because this is not a real fixup.
      </para>
    </sect1>

    <sect1 id="fixup_destroy">
      <title>fixup_destroy</title>
      <para>
        This function is called from the debug code whenever a problem
        in debug_object_destroy is detected.
      </para>
      <para>
        Called from debug_object_destroy when the object state is:
        <itemizedlist>
          <listitem><para>ODEBUG_STATE_ACTIVE</para></listitem>
        </itemizedlist>
      </para>
      <para>
        The function returns 1 when the fixup was successful,
        otherwise 0. The return value is used to update the
        statistics.
      </para>
    </sect1>
    <sect1 id="fixup_free">
      <title>fixup_free</title>
      <para>
        This function is called from the debug code whenever a problem
        in debug_object_free is detected. Further it can be called
        from the debug checks in kfree/vfree, when an active object is
        detected from the debug_check_no_obj_freed() sanity checks.
      </para>
      <para>
        Called from debug_object_free() or debug_check_no_obj_freed()
        when the object state is:
        <itemizedlist>
          <listitem><para>ODEBUG_STATE_ACTIVE</para></listitem>
        </itemizedlist>
      </para>
      <para>
        The function returns 1 when the fixup was successful,
        otherwise 0. The return value is used to update the
        statistics.
      </para>
    </sect1>
  </chapter>
  <chapter id="bugs">
    <title>Known Bugs And Assumptions</title>
    <para>
        None (knock on wood).
    </para>
  </chapter>
</book>
```