The Memory checker is a memory and context debugger tool for SoCLib.

Overview

Tracking software bugs is not an easy task and working with super user privileged code or without a memory management unit may even increase difficulty. Some hard-to-find bugs may stay hidden under certain conditions and suddenly appear depending on undefined memory content for instance. Sometimes, unexpected embedded software crashes are simply due to stack overflow.

The Memory checker tool can help to track these problems down by watching processor memory accesses and keeping an up-to-date representation of running contexts and memory allocation map. It's able to report memory accesses which are valid for hardware but invalid from a software point of view given current software state. It is similar to the ?valgrind memory checker tool available on GNU/Linux and MacOs, but aims at checking Operating System level memory operations rather than Unix user process memory operations.

Like the Gdb Server, the Memory checker contains no processor-specific code and can be used to manage any Soclib processor model using the generic Iss2 interface. It is implemented as an Iss wrapper class.

When the Memory checker is in use, it internally traces all events between the processor Iss model and the SoCLib platform. The running Operating System must be instrumented slightly to let the Memory checker be aware of valid stack ranges for each contexts and allocation ranges. The ?MutekH Operating System has support for the Memory checker.

What is being checked

All memory accesses are monitored and checked for read to non previously initialized (written) words.

Context and stacks related checks:

- The stack pointer register must stay in the range given by the Operating System for each software context in use.
- The frame pointer register (if any) must stay in the range given by the Operating System for each software context in use.
- Contexts stack ranges can not overlap (checked on context creation).
- Stack ranges must be in allocated memory at context creation (as soon as allocation checks are enabled).
- The stack memory is marked as non-initialized when a new execution context is created.
- The stack memory is always considered as non-initialized below the stack pointer.
- Memory r/w accesses in stack can not occur below the stack pointer.
- Switch to invalidated contexts.

Memory allocation and region checks:

- Write accesses can not occur in read-only preloaded sections.
- Preloaded sections are marked as uninitialized when appropriate.
- Memory is marked as uninitialized on `malloc()` invocation.
- Memory is marked as uninitialized on `free()` invocation.
- Memory r/w accesses can not occur in freed memory.
- Allocation are only allowed in free memory.

Irq and spin-lock checks:

- Irqs are disabled when accessing memory checker registers to perform context switch or region changes.

- Irqs are disabled when taking a spin-lock.
- Irqs are disabled when a temporary context with small stack is running.
- Processor doesn't deadlock on already held spinlock

What is being reported

The default behavior is to report each suspicious memory access with a message on simulator output.

More diagnostic messages can be reported, see configuration section below.

A trap exception can be reported to an optional Gdb Server module to stop processor execution when a suspicious memory access happened. This allows further analysis of buggy software. When using the Memory checker with the `GdbServer`, the Memory checker must wrap the processor directly and must be wrapped in the GdbServer.

Adding Memory checker support to your SoCLib platform

Declaration

Adding the Memory checker to your topcell is easy. First include the header:

```
#include "iss_memchecker.h"
```

Then call the init function with mapping table and loader parameters and replace processor instantiation:

```
// Without Memory checker
// soclib::caba::VciXcacheWrapper<soclib::common::Mips32ElIss> cpu0("cpu0", 0, maptab, IntTab

// With Memory checker
soclib::common::IssMemchecker<soclib::common::Mips32ElIss>::init(maptab, loader, "tty,ramdac_
soclib::caba::VciXcacheWrapper<soclib::common::IssMemchecker<soclib::common::Mips32ElIss> > c
```

Finally do not forget to update the platform description file:

```
Uses('vci_xcache_wrapper', iss_t = 'common:iss_memchecker', iss_memchecker_t = 'common:mips32el'
```

Initialization

The line:

```
soclib::common::Memchecker<soclib::common::Mips32ElIss>::init(maptab, loader, "tty,ramdac_ctrl")
```

takes the following arguments:

- `maptab`: the platform's `!MappingTable`, in order to know where memory is mapped.
- `loader`: the platform's `!ElfLoader`, in order to know memory layout, initialized or constant parts, ...
- `"tty,ramdac_ctrl"` a list of exclusions in the `!MappingTable` segment's names. You should ignore any segment mapped to a device.

Some platforms shipped with SoCLib uses the Memory checker, have a look to the trunk/soclib/soclib/platform/topcells/caba-vgmn-mutekh_soclib_tutorial source code.

Using an instrumented Operating System

The running Operating System must communicate with the Memory checker to report information about context creation (stack range), and memory-allocator operations. This is done through read/write operations in specific memory locations which are intercepted by the Memory checker and not forwarded to the rest of the platform.

Currently the only known supported Operating System is ?MutekH with Mips32, Arm and Powerpc processors. Adding Memory Checker support to an other Operating System is a trivial task though.

MutekH example

Here is an example of MutekH build invocation for SoCLib Mips32 with Memory checker support:

```
make CONF=examples/hello/config BUILD=soclib-mips32el:pf-tutorial:memcheck
```

The kernel can be use with the caba-vgmn-mutekh_kernel_tutorial SoCLib platform:

```
./system.x mips32el:4 .../mutekh/hello-soclib-mips32el.out
```

Note:

- An instrumented Operating System can not be used without the ISS Memory checker module as memory accesses won't be intercepted and may cause bus error or side effects.
- The default base address for the register bank of the memory checker is 0x00004200. This address can be changed but must stay close to 0 to fit on some processor instruction immediate field. You should consider this if you already have components at these addresses.
- The Memory checker registers bank is protected by a magic value and is unlikely to be modified by an other running software.

Configuration

The Memory checker tool behavior can be configured through the use of the `SOCLIB_MEMCHK` environment variable. This variable may contains some flag letters:

- The **T** flag can be used to raise a trap exception on suspicious memory access. This trap can be caught by the Gdb Server tool.
- The **R** flag can be added to report all allocator related operations reported by the Operating System.
- The **C** flag can ba added to report all execution contexts creation and deletion operations.
- The **S** flag can be added to report all context switch operations.
- The **I** flag can be added to display all processor registers on each errors.
- The **A** flag can be added to display offending memory access details.
- The **L** flag can be added to show all spin-lock takes and release.

The `SOCLIB_MEMCHK_REPORT` and `SOCLIB_MEMCHK_TRAPON` variables may be set to an error id mask to precisely choose which errors are being reported and which errors raise a trap. The default value for `SOCLIB_MEMCHK_REPORT` is all one, and the default value for `SOCLIB_MEMCHK_TRAPON` depends on the `T` flag. Error ids list can be found on the top of this file?.

Here is an example invocation to obtain an interlaced Memory checker and `GdbServer` function calls trace, with trap exception and wait for gdb client connection on error:

```
SOCLIB_MEMCHK=T SOCLIB_GDB=SC ./system.x ......
```

Output example

This is an example output of running a MutekH Memory checker enabled kernel with a buggy application:

Each error is reported along with information about current running context and memory region associated with the error.