New debugging concept for symmetric multiprocessing (SMP)

Lauterbach, the leading manufacturer worldwide of high-quality debuggers and real-time trace tools, is presenting its new debugging concept for SMP systems at the ESC 2008. SMP systems use an operating system to distribute processes dynamically to several cores or hardware threads. SMP Linux on a MIPS 34K will be demonstrated live at the fair.

Hardware parallelization

To achieve higher processor performance for complex applications and to save energy, more and more parallelization of tasks is used. The most popular method is to provide several identical execution units that can all process the same task.

To most readers, the term “identical execution units” means symmetric multi-core processors. It is easy to imagine that the kernel of an operating system is running permanently on a core and ensures that the application processes are evenly distributed to all cores (see Fig. 1).

However, for the parallelization of tasks not necessarily a multi-core processor is required. Hardware multithreading, for example, is an approach that enables parallelization also for single-core processors. Here, we deal with a basic problem of cores with pipeline architecture: cache misses or data dependencies between the instructions mean that the pipelined instruction processing has to be stalled in order to wait for the availability of required data. The greater the difference between instruction processing time and memory access time, the higher the performance loss.

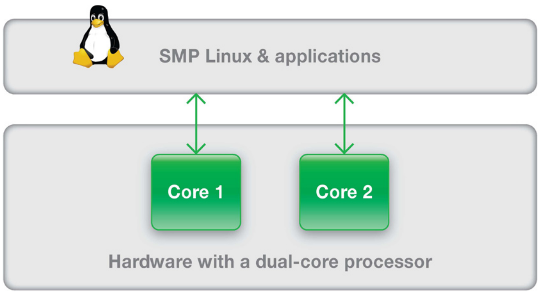


Fig. 1: SMP Linux on a dual-core processor

Hardware multithreading deals with this situation by making the core process several mutually independent tasks quasi-simultaneously. In the case of an operating system, “tasks” and “processes” are equivalent.

In principle, this works as follows: As soon as a task can no longer be further processed because required data is not yet available, the processing of another task is continued.

Fig. 2 shows the processing of 3 tasks on 3 cores with simple pipeline architecture (at the top) and then (at the bottom) shows the execution on a multithreaded core.

In order for hardware multithreading to work, fast switching between tasks must be possible. This can be done simply by giving each task a separate register set for its context. With this method, it is easy to deduce that the number of register sets provided by the core defines how many tasks can be processed in parallel. For the MIPS 34K the number is five. In this way, one ⟫

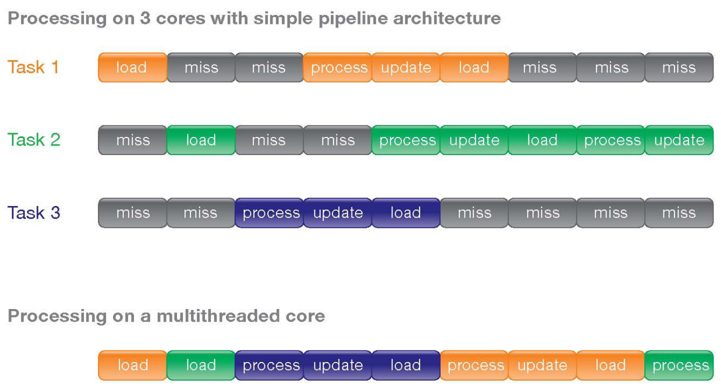

Processing on 3 cores with simple pipeline architecture
Task 1
load miss miss process update load miss miss miss
Task 2
miss load miss miss process update load process update
Task 3
miss miss process update load miss miss miss miss
Processing on a multithreaded core
load load process update load process update load process

Fig. 2: No more stalls in the pipeline, thanks to the quasi-simultaneous processing of several tasks

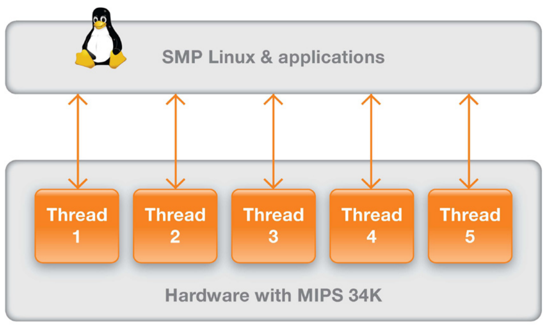


Fig. 3: SMP Linux on a multithreaded core – here MIPS 34K

"execution unit" (a so-called hardware thread) derives from each register set (see Fig. 3).

SMP operating systems

At the hardware level, the parallelization of processes is implemented by symmetric multi-core processors or multithreaded cores. Now software is needed to organize the parallelization. An operating system is normally used for this purpose.

One variant, which is primarily suitable for hardware with identical execution units, is operating systems that implement symmetric multiprocessing (SMP). The central feature of SMP operating systems is the dynamic distribution of task/processes to the available cores or hardware threads at program runtime.

Other features:

- One instance of the operating system operates all cores or hardware threads.
- Each application process can run on each core or hardware thread. As a rule, only the kernel is rigidly assigned to a core or hardware thread.
- All cores or hardware threads have equal rights to request and use resources (e.g.: memory, external interfaces, external devices).
- The operating system provides the functions for distributing resources to the cores or hardware threads.

New debug concept

At present, the TRACE32 concept enables just one core to be debugged with one instance of the debugger (*Core View*). Up to now, this concept has also worked perfectly for multi-core processors since these processors were also operated as asymmetric multiprocessing systems (AMP). Asymmetric multiprocessing means: An independent instance of an operating system runs on each core. For AMP systems, this always statically defines which process is running on which core. In this way, the debug information can also be uniquely assigned to the corresponding core.

In contrast, on SMP systems, the processes are not assigned to the cores or hardware threads until runtime. For this reason, it no longer makes sense to start a debugging instance specifically for the debugging of a selected core or hardware thread.

System View

As an alternative to the *Core View*, which works very well for AMP systems, there is now the *System View* for SMP systems.

With the *System View*, only one instance of the TRACE32 debugger is started to debug all cores or hardware threads (see Fig. 4). Here too, root of information display is one core or hardware thread. The new aspect here is that the information 》

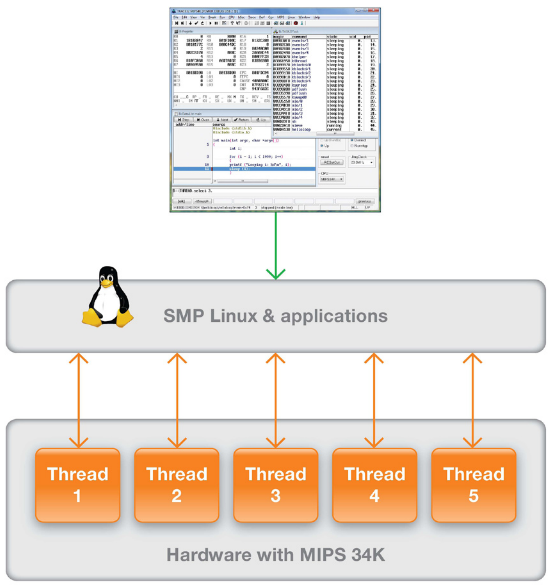
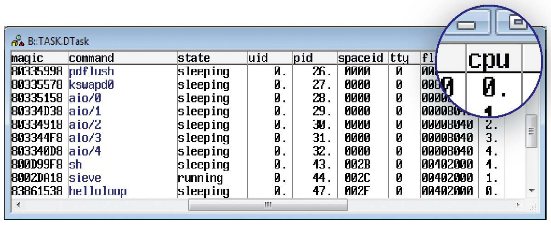


Fig. 4: Only one instance of the TRACE32 debugger is used for debugging all cores or hardware threads.

display can be switched by a command to another core or hardware thread.



```
magic    command      state     uid   pid  spaceid tty fl       cpu
80335998 pdflush      sleeping   0.   26.  0000    0   00       0.
80335578 kswapd0      sleeping   0.   27.  0000    0   000      1
80335158 aio/0        sleeping   0.   28.  0000    0   0000
80334D38 aio/1        sleeping   0.   29.  0000    0   000080
80334918 aio/2        sleeping   0.   30.  0000    0   00008040 2.
803344F8 aio/3        sleeping   0.   31.  0000    0   00008040 3.
803340D8 aio/4        sleeping   0.   32.  0000    0   00008040 4.
800D99F8 sh           sleeping   0.   43.  002B    0   00402000 4.
8002DA18 sieve        running    0.   44.  002C    0   00402000 1.
83861538 helloloop    sleeping   0.   47.  002F    0   00402000 0.
```

Fig. 5: The TASK.DTASK window, showing the core or hardware thread to which SMP Linux has assigned a task/process

Process debugging can now be done as follows:

1. The current list of all active task/processes is used to identify the core or hardware thread on which a task/process is running (Fig. 5). Linux uses here the term CPU instead of using core or hardware thread.
2. A command is used to switch the information displayed in the debugger to the required core or hardware thread (Fig. 6).

Common breakpoints

The fact that an SMP operating system does not assign application processes to a core or hardware thread until runtime also has consequences for the setting of on-chip breakpoints.

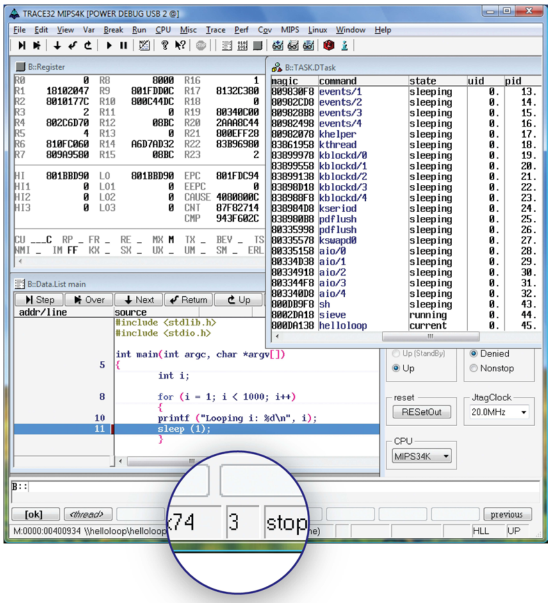
A simple example: The “sieve” process is to be stopped as soon as it writes to the variable “xyz”. In order for the debugger to be able to implement this request, it must program the break logic of all cores or hardware threads for this break condition since it cannot know beforehand the core or hardware thread on which the “sieve” application process will run. This means that even if each core or hardware thread has its own break logic, the debugger programs the breakpoints for the user as if there were only one break logic shared by all.

When setting of software breakpoints, for the implementation of which the original instruction in memory is temporarily overwritten by a break instruction, there are no changes compared to AMP systems. Operating systems protect the address spaces of processes from each other. However, if the same application process is started more than once, Linux (for example) loads the program code only once. Each instance of the application process thus sees the same program memory as well as the software breakpoints set there. To make sure that the program execution is stopped only in the desired application process, the TRACE32 debugger enables the setting of process-specific software breakpoints.

Summary

The systematic extension of the TRACE32 concept enables Lauterbach to offer its customers simple debugging of embedded designs that use an SMP operating system for controlling several cores or hardware threads. To be able to use this new concept for debugging SMP operating systems, you just need a TRACE32 debugger whose debug cable has not only a license for the processor architecture but also a second license such as a multicore license.

Following the support for the MIPS 34K, the debugging of SMP systems for ARM and PowerPC architectures is also planned for early 2008.



```
R0          0  R8       8000  R16        1
R1   18102047  R9   801FDD0C  R17 8132C380
R2   8010177C  R10  800C44DC  R18        0
R3          2  R11         0  R19 80340C00
R4   802C6D70  R12      08BC  R20 2AAA8C44
R5          4  R13         0  R21 800EFF28
R6   810FC060  R14  A6D7AD32  R22 83B96980
R7   809A9580  R15      08BC  R23        2

HI   801BBD90  LO   801BBD90  EPC   801FDC94
HI1         0  LO1         0  EEPC         0
HI2         0  LO2         0  CAUSE 4080800C
HI3         0  LO3         0  CNT   87F82714
                              CMP   943F602C

CU ___C  RP _  FR _  RE _  MX M  TX _  BEV _  TS
NMI _  IM FF  KX _  SX _  UX _  UM _  SM _  ERL
```

```
addr/line  source
           #include <stdlib.h>
           #include <stdio.h>

           int main(int argc, char *argv[])
           {
                   int i;

                   for (i = 1; i < 1000; i++)
                   {
                   printf ("Looping i: %d\n", i);
                   sleep (1);
                   }
```

```
magic    command      state     uid   pid
809830F8 events/1     sleeping   0.   13.
80982CD8 events/2     sleeping   0.   14.
809828B8 events/3     sleeping   0.   15.
80982498 events/4     sleeping   0.   16.
80982078 khelper      sleeping   0.   17.
83861958 kthread      sleeping   0.   18.
83899978 kblockd/0    sleeping   0.   19.
83899558 kblockd/1    sleeping   0.   20.
83899138 kblockd/2    sleeping   0.   21.
83898D18 kblockd/3    sleeping   0.   22.
838988F8 kblockd/4    sleeping   0.   23.
838984D8 kseriod      sleeping   0.   24.
838980B8 pdflush      sleeping   0.   25.
80335998 pdflush      sleeping   0.   26.
80335578 kswapd0      sleeping   0.   27.
80335158 aio/0        sleeping   0.   28.
80334D38 aio/1        sleeping   0.   29.
80334918 aio/2        sleeping   0.   30.
803344F8 aio/3        sleeping   0.   31.
803340D8 aio/4        sleeping   0.   32.
800DB9F8 sh           sleeping   0.   43.
8002DA18 sieve        running    0.   44.
800DA138 helloloop    current    0.   45.
```

```
M:0000:00400934 \\helloloop\helloloop  <74  3  stop  he)
```

Fig. 6: The TRACE32 window, which always shows only the context of a core or hardware thread. The number of the core or hardware thread is shown in the status line.