

## Overview

With the release of ReplayEngine 1.7, a Backwards Continue feature has been added, with support for breakpoints, watchpoints, and expression points.

MPI support has been expanded to include Nemesis and SSM channels for MPICH2, and shared memory configuration for OpenMPI, IntelMPI, HP MPI, and LAM MPI. This release also features enhanced stability and performance for multi-threaded programs using OpenMP

## Limitations

- **Obscure instructions:** Use of AMD 3DNow! and other extended AMD instructions is not supported (though Intel SSE, SSE2, SSE3 and SSE4 instructions are supported). Instructions that modify CS, DS, ES or SS registers are also not supported.
- **AsynclIO:** ReplayEngine does not support asynchronous IO operations: `io_cancel`, `io_destroy`, `io_getevents`, `ioperm`, `iopl`, `io_setup`, and `io_submit` system calls are all unsupported.
- **Exec:** ReplayEngine does not support the `execve` syscall, as used by `libc`'s `execl()`, `execlp()`, `execle()`, `execvp()`, and `execve()` functions. If the target program attempts to issue this system call, forward execution will not be possible beyond this point (though reverse execution is still possible).
- **Obscure system calls:** Certain rarely used system calls are not supported. If the target program attempts to issue an unsupported system call, forward execution will not be possible beyond this point (though reverse execution is still possible). The following system calls are either esoteric or obsolete, and only maintained in the kernel for backward compatibility with binaries written for early 2.x series kernels: `ssetmask`, `modify_ldt`, `pivot_root`, `vm86`, and `unshare`.
- **Use of `setrlimit()`:** If the target program uses `setrlimit` to reduce the amount of memory, processes, or other resources consumed, ReplayEngine may not be able to operate properly due to lack of resources.
- **Use of x86 inter-segment (aka 'far') jumps/calls:** ReplayEngine does not support the use of far jumps/calls in the target program. Any such attempt will result in forward execution not being able to continue from the point at which the far jump/call instruction is issued.
- **Non-executable memory:** ReplayEngine ignores the executable status of memory when running code, so code that would usually fail because it is in non-executable memory will run successfully.
- **Disk usage:** Depending on the target program, ReplayEngine can create large temporary files within `/tmp`. See "System Resources ReplayEngine Uses" in Getting Started with ReplayEngine for information on how to use alternative temporary directories.
- **Statically-linked target programs:** ReplayEngine cannot start a statically-linked target program. However, it can attach to an existing statically-linked target program process.
- **Self-modifying code:** Self-modifying code is supported as long as the target program executes at least one branch instruction between the writing of the code and its execution.
- **Shared memory accesses straddling valid and invalid pages:** Accessing shared memory where the instruction's operand straddles a page boundary such that the first part of the

operand is in accessible shared memory, but the second part is in mapped shared memory which is not backed by a valid shared object (e.g. because the file which is mapped has been truncated) should receive signal SIGBUS. Under ReplayEngine, a target program making such an access will not receive SIGBUS but will read zeroes for the part of the operand that straddles into unbacked memory. Note that normal attempted access to shared memory not backed by a shared object will generate a SIGBUS as normal; the issue is only with a single instruction's access that lies half in valid memory and half in invalid memory that should generate a SIGBUS.

- **Breakpoints:** All breakpoints used with ReplayEngine work like hardware breakpoints. In particular, if the code where the breakpoint resides is not modified, writing to that code will not remove the breakpoint, and setting a breakpoint that is not at the first byte of an instruction will have no effect.
- **System call output buffers:** Any system calls that write to memory must be passed a buffer entirely within writable memory. For example, if read() is passed an 8k buffer of which only the first 4k is in user-writable memory, if that read() would normally return 4k or fewer characters then natively it may succeed, but on ReplayEngine it will fail with EFAULT. If a system call that writes to memory is passed a buffer which is not in writable memory at all, but fails for some other reason before the kernel tries to write to the buffer, then natively it may fail with some error other than EFAULT, but on ReplayEngine it may fail with EFAULT. If two buffers which overlap are passed to a system call which writes to both of them or reads from one and writes to the other, the behavior in ReplayEngine may differ from the native behavior (although behavior in such cases is liable to vary between kernel versions, too.)
- **Adjust Flag:** According to the Intel manuals, the state of the Adjust Flag (AF) after some instructions is "undefined." On some processor models, different executions of the same code can produce different states of AF. If the behavior of a program depends on the state of AF when it is supposed to be undefined, the program may not run correctly with ReplayEngine.
- **SIGCHLD while attaching:** If a SIGCHLD arrives for a process while ReplayEngine is in the middle of attaching to the process, the SIGCHLD may be silently lost. Once the process has been attached to, SIGCHLD is handled normally.

## Performance Issues

### High TLB rates with certain multi-threaded target programs

When reverse debugging an application in which many threads make frequent system calls on a multi-processor platform, binding the application process to a single processor can improve performance. This is because such applications put stress on ReplayEngine's heap management, which in turn stresses the processor's TLB (translation lookaside buffer). If the application is bound to a single processor, it is less likely to suffer TLB misses caused by process migration. Since user threads are automatically serialized during reverse debugging, there is no loss of concurrency due to binding.

If the application is to be launched under TotalView, one way to accomplish binding is to preface the TotalView command with a taskset(1) command specifying a single processor. For example:

```
taskset --cpu-list 3 totalview -replay myapp
```

To accomplish binding when TotalView is to be attached to a running application, find the PID (process identifier) of the application process, and use taskset to bind that process to a single processor before attaching to it with TotalView. For example:

```
taskset --pid --cpu-list 3 <PID of myapp>
```

We have noticed the need for such binding when debugging MySQL applications with ReplayEngine.

### **Avoiding self-contention in OpenMP target programs**

Because threads are serialized during reverse debugging, OpenMP implementations that use non-yielding spins for synchronization can experience self-contention, resulting in poor performance. ReplayEngine has internal knowledge of several OpenMP implementations and tries to avoid this situation. Since this aspect of OpenMP is somewhat loosely standardized, however, ReplayEngine may not always be able to avoid self-contention.

For the Portland Group compilers in particular, ReplayEngine uses environment variables to avoid self-contention. It inserts the settings `OMP_WAIT_POLICY=ACTIVE` and `MP_SPIN=0` into the environment. The effects are, respectively, to cause idle threads to wait using a semaphore check loop, and to cause the semaphore check loop to call `sched_yield` in every iteration. If the user has pre-set either of these environment variables, ReplayEngine will not alter the settings.

### **Known Problems:**

- Viewing thread local storage is not yet supported for OpenMP programs compiled with gcc. Data in thread local storage should be visible in OpenMP programs generated by other compilers, including Intel, gfortran and PGI. (CR 11173)
- Automatic acquisition of processes created via fork is currently not working. You will see only the parent process in your debugging session. (CR 11122)
- MVAPICH over Infiniband is not supported. If you have configured mvapich 1.1 or 2.0 using the `osu_ch3:mrail` channel, you will be unable to use ReplayEngine. (CR 11757)
- There is a known performance issue with applications using OpenGL and either Nvidia or AMD/ATI hardware accelerated graphics.
- ReplayEngine does not support the debugging of applications using CUDA, OpenCL, PGI Accelerated Fortran, PGI CUDA for Fortran, or CAAs computational accelerators using either NVIDIA or AMD/ATI graphics processing units. (CR 12110)