**Hyperkernel:** **Push-Button Verification of an OS Kernel**

**陈双-P17206032-华北计算所**

**介绍：**

本文描述了一种设计、实现和正式验证OS内核功能正确性的方法，称为超核，具有高度的证明自动化和低证明负担。本文在XV6上设计了超内核的接口，这是一个UNIX类的教学操作系统。超内核引入三个关键思想来实现证明自动化：它细化内核接口以避免无界的循环或递归；它分离内核和用户地址空间，以简化关于虚拟内存的推理；并且它在LLVM中间表示L执行验证，避免建模复杂的C语义学。

用Z3SMT求解器验证了超内核的实现，检查了总共50个系统调用和其他陷阱处理程序。经验表明，超核可以避免类似于XV6中发现的错误，并且可以用低的证明负担来实现超核的验证。

操作系统内核是计算机系统最重要的组成部分之一，因为它为用户应用程序提供必要的抽象和服务。例如，内核强制执行进程之间的隔离，允许多个应用程序安全地共享诸如CPU和内存等资源。因此，内核中的错误对正确性和安全性有严重的影响，导致单个应用程序中的错误行为允许恶意应用危及整个系统。

本文探讨了一种构建具有可证明正确的OS内核和低证明负担的按钮方法。我们以XV6教学操作系统为出发点，对X86的UNIX V6进行了现代的重新实现。

**挑战性和难点：**

验证超内核的一个关键挑战是接口设计之一，它需要在可用性和证明自动化之间取得平衡。一方面，内核维护一组丰富的数据结构和不变量来管理进程、虚拟内存和设备，以及其他资源。因此，超内核接口需要支持高级别属性（例如，进程隔离）的规范和验证，这为用户应用的正确性推理提供了基础。另一方面，这个接口也必须以一种能够实现SMT求解器的这种属性的完全自动化验证的方式来实现。

第二个挑战源于内核代码中的虚拟内存管理。x86上的典型地址空间布局：内核和用户空间都位于同一虚拟地址空间中，内核取上半部并将下半部留给用户空间。内核虚拟内存通常不是内射映射-写入一个内核内存地址可以改变另一个地址的值，因为两个虚拟地址都可以映射到相同的物理地址。因此，对内核数据结构的推理需要对虚拟物理映射的推理。由于内核代码可以在执行过程中改变虚拟拓扑映射，这一推理任务进一步复杂化。在这样的设置中，证明内核代码的特性特别困难。

最后一个挑战是超核，像许多其他OS内核，用C语言编写，这是众所周知的使形式推理复杂化的编程语言。由于指针运算和内存访问等低级操作，C语言的语义和原因难以精确建模。此外，C标准允许编译器利用未定义行为来产生高效代码。

**解决方案：**

超核解决这些挑战有三个想法。首先，它的内核接口被设计成有限的：系统调用、异常和中断中的所有处理程序都是无界的循环和递归，使得可以使用SMT对它们进行编码和验证。

第二，超内核在用户空间中的一个单独的地址空间中运行，使用内核的身份映射；这简化了对内核代码的推理。为了有效地实现这种分离，超内核利用了英特尔VTX和AMDV提供的X86虚拟化支持：内核和用户进程分别在根（主机）和非根（客户）模式中运行，使用单独的页表。

第三，超核在LLVM中间表示的级别上执行验证，其语义比C简单得多，同时保持足够高的级别以避免推理机器细节。

超内核的内核接口由50个陷阱处理程序组成，为进程、虚拟内存、文件描述符、设备、进程间通信和调度提供支持。当前的超核原型在单处理器系统上运行；验证多处理器支持超出了本文的范围。我们选择不验证内核初始化和胶粘代码，例如，用于寄存器保存和恢复的程序集，而依赖于一组自定义的检查器来提高其正确性的信心。

为了演示内核接口的可用性，我们将XV6用户程序移植到超内核，包括实用程序和外壳。我们还移植了XV6日志文件系统和LWIP网络栈，它们都作为用户空间进程运行。我们已经开发了几个应用程序，包括Linux二进制仿真器和Web服务器，可以承载本文的GIT存储库。

**实现方法：**

本文做了两个主要贡献：构建一个验证的OS内核的按钮方法和一个适合SMT解决的内核接口设计。内核接口的精心设计是实现高证明自动化的关键，应用超核方法来验证现有内核是不可能扩展的。我们选择XV6作为一个起点，因为它提供了经典的UNIX抽象，最终的超内核接口，它适合于自动验证，类似于ExoClinux内核。

为了证明超核的两个正确性定理，验证者对SMT中的核属性进行编码，以进行自动验证。为了验证可扩展性，验证者限制其验证。

如文中图5超内核中的状态转换。在每个步骤过程中，由于系统调用、异常或中断，执行可能停留在用户空间或陷阱到内核中。内核中的每个陷阱处理程序都运行到中断中断完成。将SMT应用于一阶逻辑的有效可判定片段。本节介绍了如何使用这种限制来指导形式化设计

本节介绍如何应用有限接口设计，使超核适合于自动验证。像Unix一样的内核，超内核提供了使用英特尔VT-X和AMDV虚拟化支持的过程的抽象。内核作为主机运行，用户进程作为来宾运行。陷阱处理程序实现为VM退出处理程序，响应超调用（实现系统调用）、抢占定时器期满、异常和中断。

这种方法有两个优点。首先，它允许内核和用户空间有单独的页表；内核只使用身份映射来映射自己的地址空间。与以前的地址空间设计相比，该设计回避了需要对内核代码进行虚拟地形映射的原因，简化了验证。第二，使用虚拟化安全地将中断描述符表（IDT）暴露给用户进程。这允许CPU将异常直接传递给用户空间，从大多数异常处理路径中移除内核。除了性能效益之外，该设计减少了需要验证超核处理由于处理过程中的“三重故障”导致的VM退出的内核代码的数量，将其余的异常处理留给用户空间。

超核需要用户空间明确地做出资源分配决策。例如，用于页面分配的系统调用需要用户空间来提供页码。内核只是检查给定的资源是否是免费的，而不是搜索一个空闲的资源。这种方法有两个优点。首先，它避免了内核中的循环，从而验证了可扩展性。第二，它可以使用基于数组的数据结构来实现，验证者可以容易地将其转换为SMT；它避免了关于链接的数据结构（例如，列表和树）的推理，这不被求解器所支持。另一方面，回收资源通常需要一个循环，例如从僵尸进程释放所有页面。为了保持内核接口的有限性，超内核将这些循环安全地推送到用户空间。例如，超内核提供了一个系统调用，用于用户空间显式地回收页面。这样做时，内核在每个系统调用中只回收有限数量的资源，避免了长时间运行的内核操作。注意，回收系统调用允许任何进程从僵尸进程中回收页面，而不需要特殊的用户进程来执行垃圾收集。

我们已经设计了超内核的系统调用接口，遵循4.1的原理。特别是，以XV6和POSIX为基础，使超核接口有限且适合于按钮验证。这种设计导致了下面描述的几种常见模式。我们还展示了这些模式如何帮助确保所需的横切属性，验证为声明性规范的一部分。

堆栈检查程序，LLVM IR不建模机器细节，如堆栈。因此，在LLVM IR级别上对超内核进行验证并不排除由于堆栈溢出而导致的运行时故障。我们实现了LLVM后端生成的X86程序集的静态检查器，它构建了一个调用图，并保守地估计了陷阱处理程序所使用的最大堆栈空间。运行检查器表明它们都在内核堆栈大小（4 Kib）内执行。链接检查器，验证者假设符号在LLVM IR不重叠。我们实现一个静态检查器，以确保链接器在最终内核映像中维护此属性。检查器从内核映像读取每个符号的内存地址和大小，并检查所有符号驻留在不相交的存储区域中。

**结论：**

我们可以得知，超核是一种OS内核，具有高度的证明自动化和低证明负担。它通过细化内核接口实现按钮验证，使用硬件虚拟化简化关于虚拟存储器的推理，并在LLVM IR级工作，以避免建模C语义。经验表明，超内核可以防止一大类错误，包括那些在XV6内核中发现的错误。我们认为，超核为验证内核和其他低级软件的设计提供了一个有前途的方向，可以通过与验证自动化共同设计系统。