IA320S Project Presentation

A Case Study in an Architectural Design Proceess Applying ADD & ATAM Methods

吴继鹏　吴圻　薛懿超　王子杰　吴磊　阎庆庆　吴超

Software Institute
Nanjing University

Architecture Final Presentations, 2014

Outline

这篇报告包括：

- 主要功能和操作场景
- 两种体系结构选择的组件图
- 6 元素法形式化表述的所有 ASR 的列表
- 描述代码结构的模型
- 代码与组件的映射
- 每个体系结构选择的优缺点
- 每个决策（设计关注点，体系结构 pattern/tactic）的理由
- 验证文档（包括 sensitivity points，trade-off points，utility tree）
- 如何应用 ADD 和 ATAM 方法
- 经历这个项目后的体会以及各组员贡献

我们选择的项目是一个基于 IA32 硬件平台的操作系统内核，该项目的逻辑复杂度足以支撑严肃的体系结构方法的使用与验证。

Outline

- 如果我们的目标仅仅是熟悉 ADD 与 ATAM 的应用，那么只需要按照范例与指导步骤完成各阶段工作即可，但是这种非创造性工作并不意味着工作者真正理解其意义。
- 在上一次应用 ADD 与 ATAM 方法时，我们已经初步熟悉了它们的使用，但是仍然对其必要性、有效性和优越性缺乏认识。
  - 举个具体的例子，当我们不理解 [1] 作者的真实意图时，介绍自己项目的功能需求仅仅是因为作业中有这个要求，而当我们理解了 ADD 的真实意图后，我们就自然而然地想要在展示 ADD 过程前介绍自己项目的功能需求，因为功能需求是 ADD first iteration 的输入之一。

- 想要深入理解体系结构方法，可以将先验的基于直觉的判断[1]与基于现有的完善方法论的工作成果进行比较，从而得出应用现有的体系结构设计与评审过程方法（ADD，ATAM）的有效性。
- 我们的项目最终成果里会包括基于直觉的体系结构设计与基于严肃体系结构方法指导的设计这两种原型 kernel 的代码，够包含完整的 Makefile 和构建环境、调试环境搭建说明。

[1]可以参见文档中 Section 4 中的基于直觉设计的体系结构模型

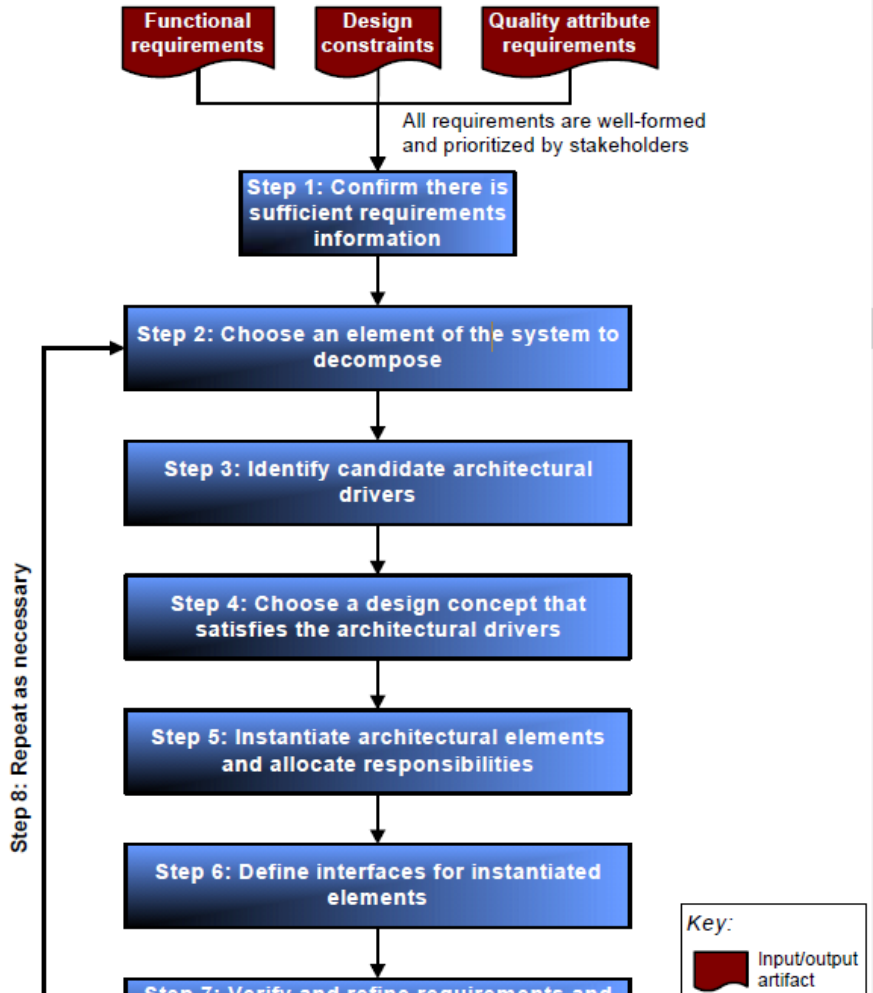
Outline

- 我们以 ADD 方法分解组件的过程为主线，穿插介绍各个迭代周期的输出、决策与决策理由。

1 中提供的 6 元素表示法在实际使用中仍然有不足之处，因为它无法清晰表达该质量属性针对的抽象层次。

- 在不同抽象层次上，6 元素表示法表示的质量属性是互相独立的，但是又糅合在一篇文档中，容易造成误解，因此我们会将文档组织格式进行调整。
- 在验证阶段使用的技术是 ATAM，我们会给出项目中的 sensitivity points，trade-off points，utility tree 等决策评价手段对体系结构设计进行评审。

- 该报告记录了我们第二次进行完整的 ADD 过程的尝试。ADD 的方法来自 [1]，组件分解的方法来自 [5]。
- Fig.1 总结了 ADD 过程的具体步骤。



Outline

- 在进行 ADD 时，第一步就是确认有足够多的输入信息，即三种需求的信息。
- 这次项目由我们自定义 ADD 的 Input。
- 其中功能性需求和约束使用结构化自然语言组织
- 非功能性需求使用“六元素表示法”[1][2]的方式形式化表示。

[2] source, stimulus, artifact, environment, response, response measures.

- 严格的操作系统内核的功能需求是比我们实现的目标版本复杂的。
- 鉴于缺乏一个需求规格，我们只能提供一个结构化的自然语言文本规格化功能需求。

1 中介绍的 ADD Input 只要求简单的用户需求描述，一句话总结一个功能需求，这种需求通常表述为“系统允许…”。[3]

[3]按照严格的 ADD 方法要求，应该对功能需求进行优先级排序，下面对需求的排序是按优先级降序排列的。

- 内核应该允许从 IA32 平台加电自载。
- 内核应该实现 IA32 平台的异常向量表初始化并提供异常处理。
- 内核应该实现 IA32 平台的分段初始化。
- 内核应该允许且仅允许内核级代码使用一个仅对内核级代码可见的栈。
- 内核应该允许且仅允许内核级代码使用一个仅对内核级代码可见的堆。
- 内核应该允许内核及代码使用一个仅对内核级代码可见的无论多么简陋都没关系的内存分配器（不对外界开放，只需开发者熟悉）。
- 内核应该初始化并且维护一个频率恰当的时钟，不仅内核可见，而且要为 User-level 客户程序提供一个访问它的系统调用。
- 内核应该初始化并维护一个显卡驱动并且为 User-level 客户程序提供一个足够友好的服务接口。

- 内核应该初始化并维护一个进程调度器，允许多进程调度。
- 内核应该初始化并维护一个消息同步机制（semaphore & PV）。
- 内核应该初始化并维护一个进程通讯机制（IPC），每个进程都应该有自己的“邮箱”。
- 内核应该初始化并维护硬盘驱动和一个虚拟文件系统（VFS）。[4]
- 内核应该初始化并维护网络协议。[5]

[4]时间有限，没有实现。
[5]太遥远，没有实现。

- boot: from floppy only
- platform: IA32
- pic: 8259A
- size of boot image: <512K
- size of other images: not limited[6]

[6]没有可能写的太大。

- 这里暂时略去全部的质量属性
- 因为我们会在文档后续的 ADD 迭代过程中让让它们逐一登场，以避免不同抽象层次的质量属性挤在一起导致规格上的混乱。

- 目前只能分解元素池中的唯一元素——Kernel。

- 在第一次迭代中，我们的元素池除了一个 Kernel 之外一无所有，所以考虑的质量属性就是 buildability 和 testability。其他质量属性不是不重要，而是目前无法作为第一个迭代的决策依据。比如说 security 属性至少要求你的系统能够承载一个用户登录机制，这根本就是非常靠后的需求，而且实现难度和硬件接口、ISR、进程调度等比起来不值一提。
- 操作系统内核的构建本身就是最艰巨的任务，将其反复重构必须建立在一个成功的构建之上，也必须建立在无数次失败的构建之上。
- 操作系统内核的可测试性是需要解决的第一要务，操作系统是最典型的 reactive system，同时还是最典型的 environmental free-standing system。这意味着它先天缺乏反馈，而且即使人为的使之具有反馈，反馈也具有并发性质，在一个基本的测试环境建立之前，来自系统边界的输入无法与输出建立一一映射，这就是最典型的“不可测试性”。这和很多其他系统的开发不同，测试平台必须优先搭建。

- 可构建性是指将一个未来系统从概念上的设计转化为实现的容易程度。[7]

Quality Attributes Specified in 6-elements Form		remarks
Portion of Scenario	Buildability(Constructability)	from design to completion
Source	developers, designers	
Stimulus	A new design is adopted.	
Architect	the future system to be built	
Environment	the future system is being built	
Response	1. The developers can understand the design concerns.	
	2. The developers can build the future system consistent with the designers' conceptual system.	
	3. The cost of construction should be less than the budget.	
Response Measures	1. The developers can capture designers' design within 2 hours.	
	2. The percentage of type-2 bugs is less than 25%.	definition of bug types: (1) type-1: The specification is wrong. (2) type-2: The specification is correct while the impl is wrong.
	3. The system can be built successfully within 3 months	

Figure : Buildability Specified in 6-elements Form

[7]我们组员曾经误解为代码编译链接成可执行程序的容易程度。

- 我们对可测试性的理解基本上没有偏差，只需要注意更严谨的说法是可测试性应用于一个 empirical hypothesis[8]，而非一个系统。因此上文中提及的操作系统内核的可测试性更准确的说应该是内核的某一功能可以正常运作的假设的可测试性。

Quality Attributes Specified in 6-elements Form		remarks
Portion of Scenario	Testability	for empirical hypothesis
Source	developers, designers	
Stimulus	every dangerous step of programming(e.g., integration of modules, change to old codes)	
Architect	correct code	
Environment	the future system is being built	
Response	1. The developers can access data in memory & CPU.	What is data in CPU? I mean data stored by various registers.
	2. The system provides error message for different bugs.	
	3. The system does not collapse silently.	
Response Measures	1. A developer can access logical address in memory and value in the stack/heap with a single call or 2~3 commands within 1 minute.	
	2. The system provide at least 12 different messages for different types of bugs.	
	3. The percentage of debugged LOC should be greater than 75%.	LOC: Line of Code

Figure : Testability Specified in 6-elements Form

[8] A hypothesis is testable if there is some real hope of deciding

- 我们的 candidate architectural drivers 毫无疑问只能是 Testability 和 Buildability。
- 下面的图表中给出其各自的相关的 design concerns。对于每个 design concerns 都会列出所有可能的 patterns 和 tactics，并且召开会议对其进行取舍。

ASR	Design Concerns	Patterns /Tactics	Detailed Rationale(Cons & Pros)	Remarks
可构建性	使复杂系统从概念上的设计到成功的实现的转化的成功率上升，成本降低。	采用分层架构。	操作系统内核的复杂度决定了使用平铺式的架构会导致混乱与逻辑爆炸，引发编程上的极大不便。分层可以让复杂度降低到人类可以接受的范围内，隐藏不必要的技术细节。	
			分层在理论上会导致性能的降低和级联修改的可修改性降低。采用分层架构的缺陷在操作系统内核编程中是可以忽视的，它的优点远远大于缺点。级联修改的可修改性的重要程度远远低于可构建性。	平铺式的架构是指面向对象和不采用主程序-子程序的面向过程架构。
可测试性	使缺乏测试环境的反应是系统能够具备初步的测试环境。	DBC	在编写操作系统内核时，不得不借助大量的DBC方法来代替测试调试，因为测试环境无论如何都是极为简陋的。	这里的DBC只能是简陋的DBC，取其本义，而不求形式上的符合。
			DBC的缺陷在于无法探测到一些规格错误和由于隐式的约束导致的错误，例如说代码量太大导致编译成果体积过大，加载到内存时可能无意中覆盖了不该覆盖的部分。	
		Watchdog	Watchdog是指一个内置的异常处理机制，和IA32的硬件平台有紧密联系。采用Watchdog是environmental-freestanding项目的自然而然的选择。	
			Watchdog是几乎没有缺陷的，不会引入任何风险，唯一的麻烦是编写watchdog代码本身，这和它所带来的便利相比不值一提。	

Figure : Tactics/Patterns & Their Rationale For Some Quality Attributes

- 按照严格的 ADD 方法，还应该列出所有可能的 Patterns 和 Tactics，并且对其进行利益分析，选择最佳组合，但是第一个迭代的选择非常局限（即使我再编造出一个面向对象风格的 pattern，也只是形式上更严谨，实际上世界上根本不存在这种操作系统编程方式。），所以这次迭代并没有放弃哪个 pattern 或者 tactic，直接应用所有的 patterns/tactics。
- ADD 要求进行的 evaluation 在文档中不进行展示，更正式的 evaluation 会以 ATAM 方法呈现。
- 此外，在做出设计决策之后，还应该对元素类型进行定义，找出不同 pattern 中各元素之间的关系。在我们的设计决策中，产生元素的 pattern/tactics 是 layered pattern 和 watchdog 这两者，DBC 仅仅是一种编程方式，无法产生任何元素。watchdog 包含分层体系的 superstructure 层之中。下一章节会对元素的属性进行详细讨论，给出各元素的定义。

- 采用分层架构，Kernel 产生了下面几个元素：
  - protocol：定义了所有组件之间的通信协议。即任何一个组件只要包含对 protocol 的引用即可与其他任何组件通信。
  - library：定义了一组实用工具。因为操作系统的环境独立性，所有基础的底层的工具都需要自行实现，比如说 printf，比如说 memcpy。
  - resource_abstraction：包含大部分资源的抽象。一个计算机的操作系统就是对硬件的利用挖掘的软件，所以必然存在大量需要抽象的资源。例如说 process 是对 CPU 资源的抽象。
  - superstructure：这个元素的命名并不是很好，实际上一个操作系统必须遵循严苛的硬件接口，不存在随心所欲地组织代码的可能，这一元素的工作就是处理操作系统内核编程中最痛苦的部分——与 Intel 的硬件接口进行交互，严格按照 IA32 的要求进行大量完全反软件工程的编码。例如说我们看这个函数：

```
static void                                    init_pic()
{
  outb(0x20,    0x11);
  outb(0xa0,    0x11);
  outb(0x21,    0x20);
  outb(0xa1,    0x28);
  outb(0x21,    0x4);
```

Outline

- superstructure

- 这是一个编码非常困难的层次，拥有各种反人类设定。
- Candidate architectural drivers 的选择必须继续考虑可构建性。
- 这一抽象层次的可构建性与第一次迭代有所不同，它面临的主要挑战来自 IA32 硬件工程师的思维和软件工程师思维之间的显著差异。

Quality Attributes Specified in 6-elements Form		remarks
Portion of Scenario	Buildability(Constructability)	
Source	developers, designers	
Stimulus	A new design is adopted.	
Architect	the future system to be built	
Environment	the future system is being built	
Response	1. The designers should provide sufficient explanation to some unreasonable strange data regulation.	E.g., you have to initiate protected-mode on IA32 platform.
	2. The developers can implement paging, segment & isr consistent with hardware interface.	
	3. The time devoted in this module should be acceptable.	
Response Measures	1. The developers can translate each hardware specification to the counterpart software code within 2 hours.	
	2. The percentage of type-2 bugs is less than 25%.	definition of bug types: (1) type-1: The specification is wrong. (2) type-2: The specification is correct while the impl is wrong.
	3. The time devoted in this module should be less than half of the total.	

Figure : Buildability At Superstructure Level Specified in 6-elements Form

- 此外另一个值得考虑的质量属性是易用性，因为该层次作为一个特殊的最接近底层的层次将要考虑非内核级代码如何方便地使用它的问题，即客户如何方便地使用它。实际上这一层次不需要考虑内核级代码如何方便地使用它，因为在内核级代码中这一层次调用了其他所有层次的功能，而非相反。

Quality Attributes Specified in 6-elements Form		remarks
Portion of Scenario	Usability	
Source	3rd-party developers	
Stimulus	They want to code on the kernel more easily and more efficiently.	
Architect	superstructure of kernel	
Environment	runtime	
Response	1. The interface should be clear and unambiguous.	
	2. There should be samples about how to use the syscalls.	
Response Measures	1. More than 20% users feel comfortable with the syscalls.	
	2. Less than 50% users feel bad with the syscalls.	
	3. The percentage of successful syscall should be more than 95%.	

Figure : Usability Specified in 6-elements Form

ASR	Design Concerns	Patterns /Tactics	Detailed Rationale(Cons & Pros)	Remarks
superstructure层次的可构建性	使遭遇大量硬件接口的系统边界实现的转化的成功率上升，成本降低。	采用分层架构。	继续分层，否则无法应对繁琐的硬件初始化逻辑。优点是使得构建更加轻松。	
			缺点是多写点Makefile，级联修改可修改性降低。	
		采用数据抽象。	data abstraction是指将数据与数据之上的操作绑定在一起的做法。优点是使得逻辑更加内聚，降低耦合，同样有利于修改，这里的修改不是因为需求变更而修改，而是因为构建失败而修改。	原型中也用了一些，但是效果不是特别好，仍然应该依靠程序员经验酌情把握。
			缺点在于繁琐，冗余，浪费内存空间存储多余的函数指针。	
易用性	使基于我们的kernel的第三方程序开发者能够满意地使用kernel。	采用微内核	微内核意味着官方的系统调用是由user-level程序或者比user-level高一个特权级但是形式上差不多的程序提供的，这就使得内核源码中存在大量User-levl的参考代码。	这里为什么我们不认定这是一种可扩展性呢？因为事实上扩展的主体是客户，本质上是我们的系统服务的对象。
			缺点是微内核的调用略微繁琐，不如宏内核那么直接。	主要由于是消息同步机制在宏内核通信中的应用，要求用户掌握消息同步机制信号量（semaphore）的使用方法。
		采用宏内核	可构建性强	
			不够模块化，且缺乏供第三方临摹的程序。	

- main：所有数据初始化流程。
- traps：异常处理机制，系统调用。
- segment：分段初始化，分页初始化，中断处理例程。

Outline

resource_abstraction

Quality Attributes Specified in 6-elements Form		remarks
Portion of Scenario	Performance	
Source	system events; input reaching system border	
Stimulus	Events created by processes;Events caused by final users' input; Events caused by inner machinism, e.g., the clock cycle and scheduling.	
Architect	resource_abstraction of kernel	
Environment	runtime	
Response	1. Correctly respond to events.	
	2. Respond to user input in time.	
Response Measures	1. More than 80% events are successfully responded.	
	2. An keyboard input can be responded in 1 second.	
	3. Less than 10% processes stay hungry for more than 10 minutes.	

Figure : Performance Specified in 6-elements Form

ASR	Design Concerns	Patterns /Tactics	Detailed Rationale(Cons & Pros)	Remarks
性能	资源抽象是决定操作系统性能最重要的模块，我们希望这一模块重点考虑性能这一质量属性，提升进程调度的敏捷程度以及进程通讯的快捷程度。事实上，也仅有这一模块才能考虑性能，在此之前，由于构建本身就异常困难，举步维艰，所以根本没有余力考虑这些。	采用彩票调度。	彩票调度本身并不是个低效的调度算法，虽然应用得不多，但是完全堪用。而且这一算法非常有趣，为什么有趣可以参见我的源码。	原谅我实在太累不想多列举其他调度策略了。
			随机数生成过程是其相对其他算法的冗余步骤，有一点点性能损耗。	
		采用semaphore消息同步机制。	semaphore可以完美地解决消息同步，而且不引人重量级的代码，非常高效。	事实上我只熟悉这一个同步机制，其他的同步机制我们似乎学过，不过我不会。
			semaphore的使用比较繁琐，需要用户至少有过这方面的训练。	
		采用给每个pcb一个独立的mailbox的进程通讯机制。	每个进程都可以通过pcb直接索引到它的私有的mailbox空间，效率比公共空间中增设管理器略高。	
			总的内存占用较高。	

Figure : Tactics/Patterns & Their Rationale For Some Quality Attributes At Resource abstraction Level

- process: 进程抽象。
- service: 系统服务。
- terminal: 终端。
- vfs：虚拟文件系统。[9]
- mm：复杂的分页管理。[10]

[9] 在原型中完全未实现。
[10] 我们的逻辑地址与物理地址的映射非常简陋。

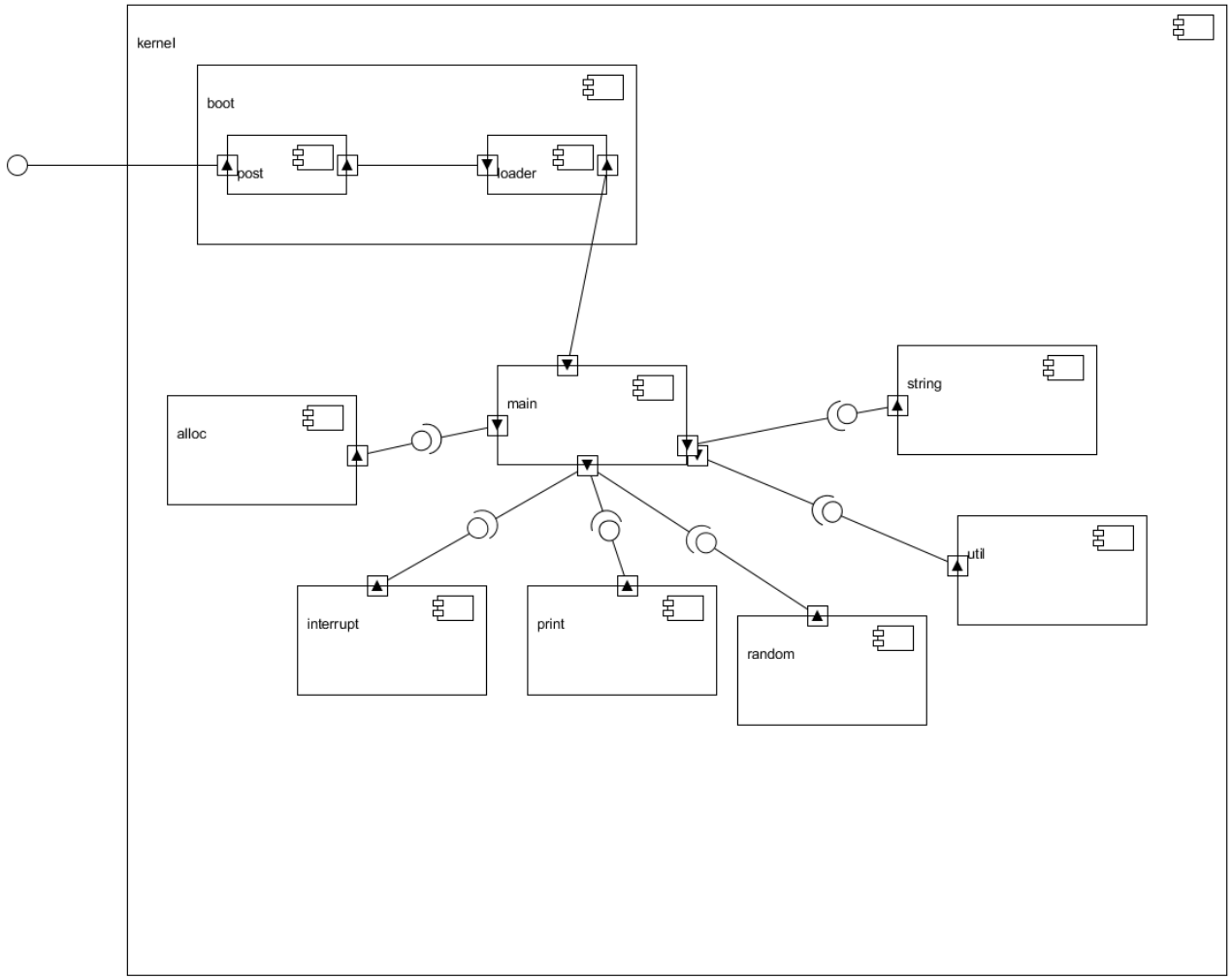


Figure : 面向过程的基于直觉的设计组件图

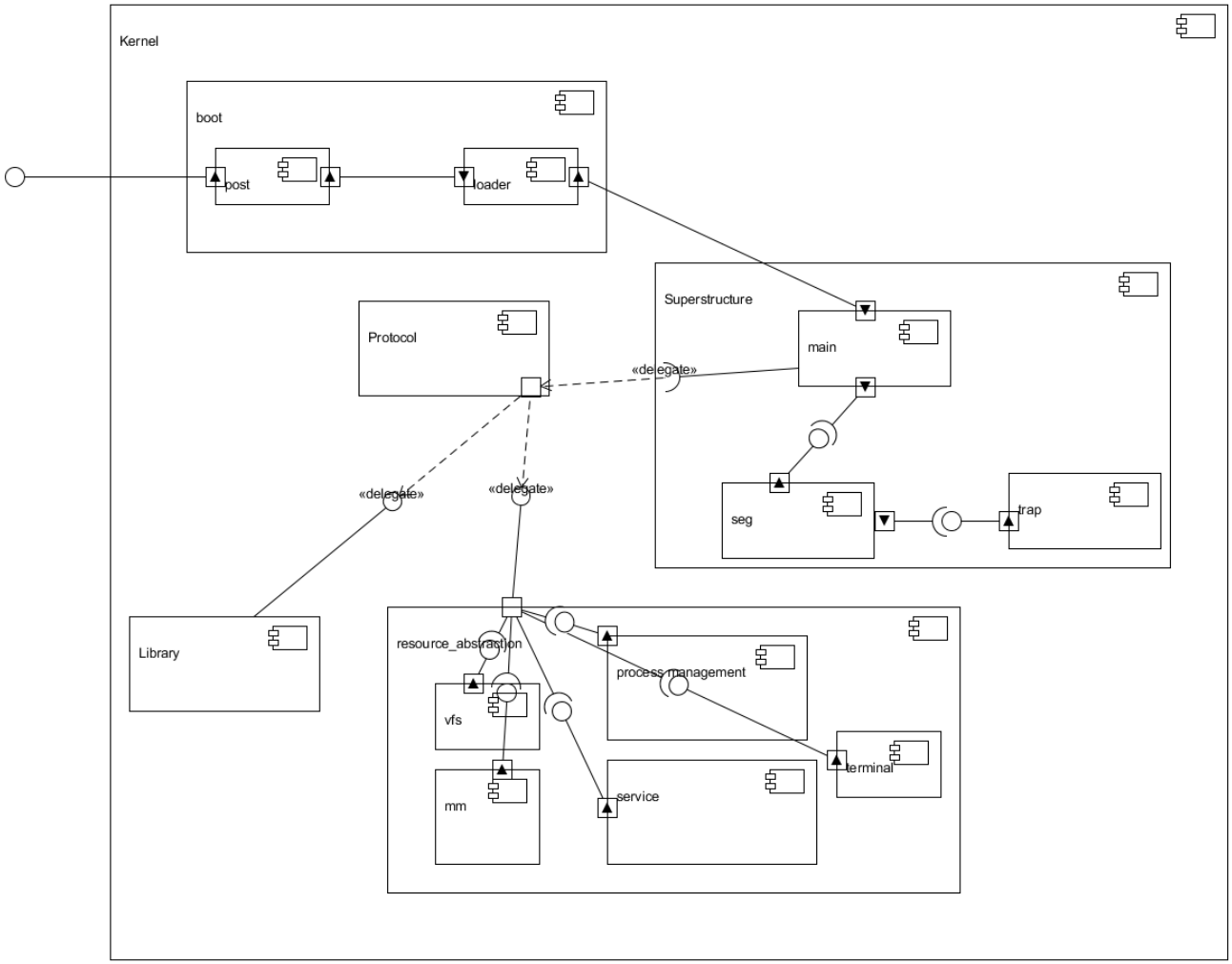


Figure : 基于 ADD 与 ATAM 的设计组件图

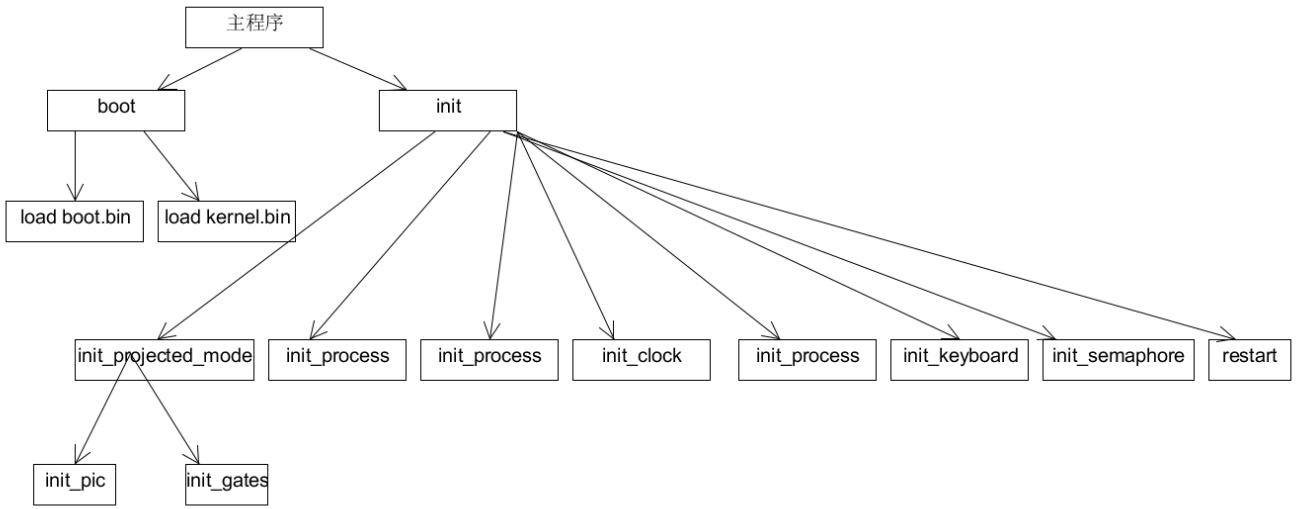


Figure ： 面向过程的基于直觉的体系结构风格图

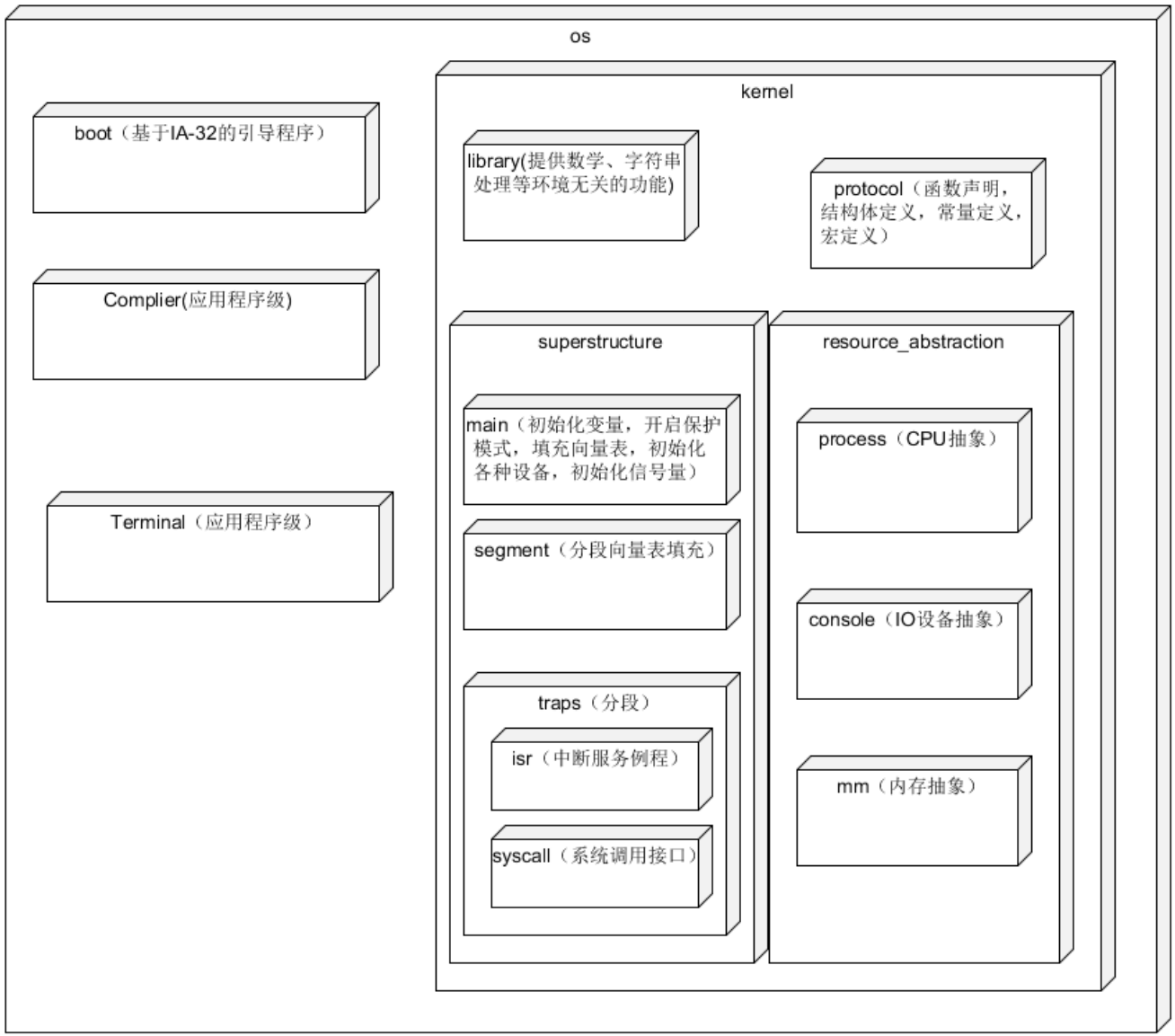


Figure : 基于 ADD 与 ATAM 的体系结构风格图

- 鉴于 UML 类图只能描述面向对象程序的代码结构，这里我们为了描述我们项目的代码结构，直接以目录层次的方法展示所有源码文件。

```
.:
Makefile
README
a.img
bochsrc
boot
kernel

./boot:
Makefile
boot.asm
include
loader.asm

./boot/include:
fat12hdr.inc
load.inc
pm.inc

./kernel:
Makefile
library
protocols
resource_abstract
superstructure

./kernel/library:
Makefile
display.c
exception.c
queue.c
random.c
string.c
string.s
syscall.s
vga.s
x86.s

./kernel/protocols:
code_version.h
consts.h
declaration.h
global
macros.h
producer_consumer_problem.h
structs
types.h
variables.h

./kernel/protocols/structs:
naive_console.h
naive_structs.h
process.h
queue.h
semaphore.h

./kernel/resource_abstraction:
Makefile
process
service
terminal

./kernel/resource_abstraction/process:
process.c
process.h
schedule.c
semaphore.c

./kernel/resource_abstraction/service:
console_service.c
ipc_test.c
producer_consumer_problem.c
sleep_for_a_while.c

./kernel/resource_abstraction/terminal:
console.c
console.h

./kernel/superstructure:
Makefile
main
segment
traps

./kernel/superstructure/main:
head.s
initialization.c

./kernel/superstructure/segment:
isr.s
seg.c
seg.h

./kernel/superstructure/traps:
keyboard.c
keymap.h
pit.c
syscall.c
```

图中的代码的所在路径表明了它所在的组件，所有组件名都与文件夹名一一对应。

Outline

Quality Attribute	Refinement	Scenario
Buildability	Design comprehensibility	When new design specification yielded, the developers can capture designers' design within 2 hours.
	Less cost	The system can be built successfully within 3 months.
	Consistency with design	The percentage of bugs caused by inconsistency with design specification is less than 25%.
	Explanation to hardware specification	An experienced kernel developers can translate each hardware specification to the counterpart software code within 2 hours.
Testability	Acess to data	A developer can access logical address in memory and value in the stack/heap with a single call or 2~3 commands within 1 minute.
	Error message	The system provide at least 12 different messages for different types of bugs.
	Coverage	The percentage of debugged LOC should be greater than 75%.
Usability	Proficiency training	A experienced user-level application developer can be trained to be proficient to complete a program using syscalls.
	User satisfaction	More than 20% users feel comfortable with the syscall and are able to adapt to the way we regulate to program on the kernel.
	Normal operations	The percentage of successful syscall should be more than 95%.
Performance	Transaction response time	More than 80% events are successfully responded.
	Throughput	The system allow 16 processes at running state.

Figure : Utility Tree

Outline

ID	Sensitivity points
S1	采用分层架构
S2	DBC
S3	Watchdog
S4	在superstructure采用分层架构
S5	采用数据抽象
S6	采用微内核
S7	采用宏内核
S8	采用彩票调度
S9	采用semaphore同步机制
S10	采用为每个pcb建立mailbox的通信机制
ID	Trade-off points
T1	采用为每个pcb建立mailbox的通信机制
T2	采用微内核
T3	采用彩票调度

ID	Risks
R1	在superstructure采用分层架构
R2	采用彩票调度
ID	NonRisks
N1	Watchdog
N2	采用数据抽象
N3	DBC
N4	采用semaphore同步机制

Figure : List of Risks & NonRisks Points

Outline

场景ID：1		场景：When new design specification yielded, the developers can capture designers' design within 2 hours.		
属性	Buildability			
环境	Build time			
刺激	A new design is adopted.			
响应	1. The developers can understand the design concerns.			
	2. The developers can build the future system consistent with the designers' conceptual system.			
	3. The cost of construction should be less than the budget.			
体系结构决策	Sensitivity Points	Tradeoff Points	Risks	Nonrisks
采用分层	S1			
推理过程	操作系统内核的复杂度决定了使用平铺式的架构会导致混乱与逻辑爆炸，引发编程上的极大不便。分层可以让复杂度降低到人类可以接受的范围内，隐藏不必要的技术细节。			

场景ID：2		场景：A developer can access logical address in memory and value in the stack/heap with a single call or 2~3 commands within 1 minute.		
属性	Testability			
环境	Build time			
刺激	every dangerous step of programming(e.g., integration of modules, change to old codes)			
响应	The developers can access data in memory & CPU.			
体系结构决策	Sensitivity Points	Tradeoff Points	Risks	Nonrisks
DBC	S2			N3
Watchdog	S3			N1
推理过程	在编写操作系统内核时，不得不借助大量的DBC方法来代替测试调试，因为测试环境无论如何都是极为简陋的。Watchdog是指一个内置的异常处理机制，和IA32的硬件平台有紧密联系。采用Watchdog是environmental-freestanding项目的自然而然的选择。			

场景ID：3		场景：More than 20% users feel comfortable with the syscall and are able to adapt to the way we regulate to program on the kernel.		
属性	Usability			
环境	Run time			
刺激	They want to code on the kernel more easily and more efficiently.			
响应	There should be samples about how to use the syscalls.			
体系结构决策	Sensitivity Points	Tradeoff Points	Risks	Nonrisks
微内核	S6	T2		
推理过程	微内核意味着官方的系统调用是由user-level程序或者比user-level高一个特权级但是形式上差不多的程序提供的，这就使得内核源码中存在大量User-levl的参考代码。			