

# 基于 XYZ/E 的 Sun Open Solaris 内核 进程形式化描述与分析

王少将<sup>1</sup>, 张广泉<sup>1</sup>, 吴国伟<sup>2</sup>

(1 苏州大学 计算机科学与技术学院, 江苏 苏州 215006)

2 江苏省教育装备与勤工俭学管理中心, 江苏 南京 210024)

[摘要] 建立并分析了 Open Solaris 内核进程模型, 采用 XYZ/E 语言对模型进行了形式化描述. 分析了内核进程数据结构、内核进程创建、系统调用以及时钟等概念, 并对其进行 XYZ/AE 的描述和 XYZ/EE 编程. 最后给出了内核进程规范, 实现多任务、多用户操作系统内核进程的描述, 对内核进程进行了逐步求精工作.

[关键词] XYZ/E, 开放多任务操作系统, 内核进程, 形式化描述, 逐步求精

[中图分类号] TP311 [文献标识码] A [文章编号] 1672-1292(2008)03-0082-06

## Description and Analyses of Open Solaris Kernel Process Based on XYZ/E

Wang Shaojiang<sup>1</sup>, Zhang Guangquan<sup>1</sup>, Wu Guowei<sup>2</sup>

(1. School of Computer Science and Technology, Soochow University, Suzhou 215006 China)

2 Educational Equipment & Work-Study Strategy Center of Jiangsu Province, Nanjing 210024 China)

**Abstract** This paper builds up an Open Solaris Kernel Process Analyzing model and refine model using XYZ/E. We mainly analyze kernel process data structure, kernel process establishment, system scheduling and check all the above with XYZ/AE. Finally, kernel process criterion is displayed, multi-task and multi-user operating system kernel process are described. Stepwise refinement is used to refine kernel process.

**Key words** XYZ/E, open multi-task operating system, kernel process, criterion description, stepwise refinement

随着软件应用的深入发展, 由设计产生的正确性问题难以得到验证, 解决此问题的一种手段是采用形式化方法<sup>[1, 2]</sup>, 其已发展成软件工程<sup>[3]</sup>的重要分支之一. XYZ 理论基于 Manna-Pnueli 线性时序逻辑, 其中 XYZ/E 是时序逻辑系统和程序语言的有机统一, 可以描述形式规范和可执行算子不同层次的抽象. 作为验证软件可靠性和生产软件工具引擎, 具有重要理论价值和实践意义.

形式化操作系统内核通常是基于专用操作系统, 这导致开发和维护的难度都很大. 而对通用操作系统内核进行形式化分析, 由于没有充分开放的授权, 给实际应用带来专利壁垒.

Open Solaris 是 Sun 公司开发的一个多任务、多用户操作系统, 具有高度实时性、最佳利用率、高可用性、高性能、极高安全性及平台选择多样性. 其基于 CDDL(通用开发和发布许可) 开放标准, 给研究操作系统和工业应用带来一次契机.

本文介绍了 Open Solaris 内核进程机制, 阐述了基于线性时序逻辑语言 XYZ/E 描述多任务操作系统内核进程, 建立多任务、多用户操作系统内核进程规范, 最后采用基于组件的逐步求精的方法, 用 XYZ/E 描述了 Open Solaris 内核进程结构.

## 1 时序逻辑语言 XYZ/E

### 1.1 XYZ/E 基本语言成分

条件元是 XYZ/E 语言的最基本组成成分, 有两种命令形式:  $LB = y \wedge R \Rightarrow @ (Q \wedge LB = z)$  和  $LB = y \wedge R$

收稿日期: 2007-10-09

基金项目: 江苏省高校自然科学基金(05KJB520119)和重庆市自然科学基金(CSTC, 2006BB2259)资助项目.

通讯联系人: 张广泉, 教授, 研究方向: 软件工程与形式化方法. E-mail: gqzhang@suda.edu.cn

$\Rightarrow \$ O(v_1, \dots, v_k) = (e_1, \dots, e_k) \wedge \$ OLB = z$ . 前者可表示是抽象规范的状态, 用于描述静态语义. 后者可表示是状态转换的可执行状态元, 用于描述动态语义. 其中“@”指  $\$ O$  或  $\diamond$ , “R”“Q”分别为条件元的条件部分和动作部分.

单元是构成 XYZ/E 程序的基本构件, 其形式为:

$[A_1; \dots; A_n]$

WHERE  $B_1 \wedge \dots \wedge B_m$

其中每一  $A_i (i = 1 \dots n)$  为一条件原子式, “WHERE”后是约束部分或约定部分.

## 1.2 通信命令

通信命令有输入命令和输出命令:  $LB_j = y \wedge R \Rightarrow \$ OChNm? x \wedge \$ OLB_k = w$  和  $LB_j = y \wedge R \Rightarrow \$ OChNm! x \wedge \$ OLB_k = w$ . 对应通道的接收进程通过通道从发送进程接收信息, 发送进程通过此通道向接收进程发送信息. 通信又分为同步式通信和异步式通信, 同步式通信是收、送双方均为直接等待方式的通信, 异步式通信是通过信箱作中介存放信件的间接等待方式的通信.

# 2 Sun Open Solaris 操作系统内核进程体系结构

## 2.1 进程模型

进程是操作系统提供的最基础的、功能性的抽象之一. 它是占据包含特定物理内存段的可执行对象, 这些内存段中包含有代码栈空间、数据空间以及进程执行所必需的其它部分<sup>[4]</sup>. Solaris 内核支持进程实体, 每个进程由一个进程标识号 (process identification number, PID) 串成一个系统级进程表.

Solaris 内核进程模型主要由 4 个部分. 一是内核线程, 是在 CPU 上被执行和调度的对象. 二是用户线程, 用以维护用户级线程状态. 三是进程、程序可执行存在方式, 用户程序的可执行环境. 四是 LWP (轻量级进程), 用户线程内核的执行上下文.

## 2.2 Open Solaris 内核组件

- (1) 调度与调配器: 管理和调度可运行线程在多个处理器上运行, 其内核调配器用以选择线程.
- (2) 进程间通信: 是进程之间的数据共享和事件同步问题. 其中 System V IPC 对 Solaris 有较大影响, 其框架提供了 3 种服务: 消息队列、信号量阵列和共享内存块.
- (3) 进程权限: 以最小权限满足需求的执行, 但不影响其正常资源的申请和执行. 权限配置当中有一些重要特性支撑: 内核支持完全的向后兼容性和可扩展性; 细粒度化权限控制; 权限可继承、可转让.

# 3 Sun Open Solaris 内核进程的形式化描述

Open Solaris 内核是多任务系统, 为避免状态爆炸, 对其进行了抽象. 首先基于 XYZ/AE 描述内核规范的形式化描述, 然后采用基于组件的逐步求精方法, 同时引入 linux2.6 内核当中  $O(1)$  调度算法与 Open Solaris 内核进程 lwp 思想结合, 进行了算法性能优化<sup>[5]</sup>.

## 3.1 第一层 Sun Open Solaris 内核规范描述

### (1) 描述主要数据结构

% Var[

/\* 进程结构

p\_as /\* 地址空间数据结构指针  
p\_pid int /\* 进程 ID 标识  
p\_tlist int /\* 关联线程表 ID 标识  
p\_user int /\* 进程所属用户 ID 标识

/\* LWP 结构

lw\_thread int /\* LWP ID 标识  
lw\_proc int /\* LWP 所属进程 ID

/\* 线程结构

t\_proc int /\* 线程所属进程标识  
t\_lwp int /\* 线程一一对应 LWP 标识

]

### (2) 描述内核进程时钟机制

Solaris 采用时钟线程作为周期定时器 (cyclic), 缺省时间为 100 次 /s. 第次时钟中断处理程序可归为 3 类: 设置系统时间; 处理内核延迟任务; 处理内核周期性任务<sup>[6]</sup>, 分别用谓词 TimeSetPty, DelayProcPty, CycleProcPty 描述.

形式化描述如下:

$TimeMagProperty := (ch \hat{=} event \wedge \#a \ kernel = a \ kernel \wedge event \ eventtype = NTERRUPT \wedge event \ input = TME\_EVENT \wedge event \ output = TME\_EVENT) \rightarrow (TimeSetPty \wedge DelayProsPty \wedge CycleProsPty)$

综上所述,至此得到 Open Solaris 操作系统的简化内核规范:如内核启动时能够满足输入参数约束,满足交互式环境的要求,则必须满足进程创建 SolarisProcCreate 范式,系统调用 SolarisSysCalSpecification 和时钟管理模块性质 TimeMagProperty

归纳以上范式,则内核规范可用 XYZ/AE 描述如下:其中以 In 部分为输入参数及交互必须满足的约束,内核运行必须满足的性质在 Where 中定义.

In iProcCreate /\* In i...部分\* /  
Kernel: = (% CHN in ( event NM, \* ): InputEveType; % CHN out(\* , event NM ): OutputResType% INP In iProcCreate  
int)  
Where  
SolarisSysCalSpecification  $\wedge$  TimeMagProperty

3.2 逐步求精到第二层

进程创建 Open Solaris 系统调用创建新进程由 Fork( 2) 函数功能实现. 每个新进程会被分配一个进程标识,与调用进程是父子进程关系. Fork( 2) 函数创建成功则完成以下工作: 将 0 代表执行成功返回值; 将子进程标识 PID 号返回给父进程. 父进程接收到子进程返回成功信号即创建完成.

Sun 在 Solaris 中研发中创建了另外一个优秀的工具: DTrace 针对 Solaris 内核有 650 万行代码, 22 000 个资源文件,采用 DTrace 工具运用其 SDT 探针技术,可以定点跟踪功能函数进行口,并且获取返回值.

调用 Fork( 2) 伪代码如下:

In iProcCreate  
% TYPE[ argc NT; argv PO NT( node); envp PO NT( node) ];  
% LOC[ p1, q q1 q2 p1: PO NT( node) ];  
% ALG[  
LB= START  $\Rightarrow$  \$ Op= ch iL\_ p id  $\wedge$  \$ OLB= ll;  
LB= ll  $\Rightarrow$  \$ Op1= Op= fork()  $\wedge$  \$ OLB= l;  
LB= l  $\wedge$  ( p1= 1)  $\Rightarrow$  \$ q= perror(“ fork”)  $\wedge$  \$ OLB= STOP;  
LB= l  $\wedge$  ( p1= 0)  $\Rightarrow$  \$ OLB= l;  
LB= l  $\wedge$  ( p1  $\neq$  1  $\wedge$  p1  $\neq$  0)  $\Rightarrow$  \$ Oq1= wait();  
LB= B  $\wedge$  \$ q2= execv(“ /path/new\_ binary”, argv)  $\wedge$  \$ OLB= STOP].

3.3 逐步求精到第三层

系统调用是一组应用编程接口库,应用程序可通过其执行内核权限操作,常见有内存分配、信号管理、进程间通信和文件 I/O 等. 本文分析 32 位的系统调用执行过程. 除了不需对参数的高 32 位进行清 0 和做数据宽度相关的处理外, 64 位系统调用的执行进程和 32 位系统调用非常相似<sup>[4]</sup>. Linux 是另外的处理方式<sup>[5]</sup>.

其源代码如下:

Syscall\_trap32( struct regs\* rp);  
ENTRY\_NP( syscall\_trap32)  
ldx [THREAD\_REG+ T\_CPU], % g1 ! 取 CPU 地址指针  
mov % o7, % D ! 保存返回地址  
...  
Jmp % D+ 8  
Nop

从实际应用角度出发,重点关心系统调用前当前进程与被访问临界区关系. 其基于两点原因: 基于某一特定的系统调用,涉及大量数据,形式化描述系统调用前后内核变量值是不可思议的举措;众所周知,应用进程最关心系统调用完成后输出返回的结果,以及其是否能继续被 CPU 资源选中为当前进程.

系统调用描述是各个系统调用描述的交集,这里采用时序逻辑式 SolarisSysCalSpecification 其定义如下:

- ① 系统调用编号 SysCallNo 对应 Solaris 内核系统调用编号, 目前最大号为 256(NSYSCALL).
- ② 所有输入参数 AllnParams 第  $i$  个参数用分量表示.
- ③ 基于对所有输入参数是否合法的断言 AllnParaConstraint 此断言对系统调用输入参数进行检查, 以排除非法调用.
- ④ 所有输出参数 AllOutParams 第  $i$  个参数用分量表示.
- ⑤ 基于对所有输出参数是否合法的断言 AllOutParaConstraint 此断言对系统调用输出参数进行检查, 以约束返回值:

SolarisSysCallSpecification:  $= \exists \text{ input}(\text{chir? event} \wedge \# \alpha \text{ kernel} = \alpha \text{ kernel} \wedge \text{event eventtype} = \text{SysCall} \wedge \text{event runPID} = \text{runPID}) \rightarrow \wedge \text{input}(\text{event SysCallNo} = i) \rightarrow \wedge \text{output}(\text{SysCallPID} \text{chout result})$

对内核进程中 LWP 进行求精:

```
% PROS[
  % PROSLWP(% CHN ASYN wp1(*, pout NM): STRING;
    % CHN ASYN wp2(pir NM, *): STRING) //LWP 输入输出参数
= = [
  % ALG[
    % LOC[ p_ wp p_ nuke_ sigwaiting p_ count p_ runnable STRING;
      p_ wp/kthread ARRAY[ array STRING ];
      LB= START⇒$ OLB= m1;
      LB= m1⇒$ O wp2? p_ wp p_ $ OLB= m2;
      LB= m2∧ p_ wp= 1⇒$ OLB= m3 /判定是否为 LWP
      LB= m2∧ ¬ p_ wp= 1⇒$ OLB= m4;
      LB= m3⇒$ O p_ wp= p_ nuke_ sigwaiting∧ $ OLB= m14 /线程状态转换
      LB= m4∧ p_ wp/kthread next= p_ sleeping⇒$ OLB= m5;
      LB= m4∧ p_ wp/kthread next= p_ runnable⇒$ OLB= m6;
      LB= m5⇒$ O p_ wp/kthread[ p_ wp] = p_ runnable∧ $ OLB= m11;
      LB= m6⇒$ O p_ wp/kthread[ p_ wp]. t astflag= 1∧ $ OLB= m7;
      LB= m7⇒poke_cpu(% ip p_ wp|p_ wp/kthread[ p_ wp])∧ $ OLB= m8 //CPU 指令
      LB= m8⇒wp_ exit(% ip p_ wp|p_ wp/kthread[ p_ wp])∧ $ OLB= m9;
      LB= m9⇒$ O p_ wp/kthread[ p_ wp] = zombie∧ $ OLB= m10;
      LB= m10⇒$ O p_ wp= bop∧ $ OLB= m11;
      LB= m11∧ p_ wp/kthread[ p_ wp] = end⇒$ OLB= m12;
      LB= m11∧ ¬ p_ wp/kthread[ p_ wp] = end⇒$ OLB= m4;
      LB= m12⇒$ O p_ wp/kthread= deathrow∧ $ OLB= m13;
      LB= m13⇒$ O wp1! exit∧ $ OLB= m14;
      LB= m14⇒$ OLB= m1;
    ]
  ]
]
```

### 3.4 逐步求精到第四层

引入  $O(1)$  调度算法, 取代采用显式的方法重新计算每个进程的时间片, 其对于有  $n$  个进程的系统复杂度可能达到  $O(n)^{[5]}$ .  $O(1)$  调度思想为每个处理器维护两个优先级数组, 一个为活动数组, 执行尚有时间片的进程; 当时间片用完时, 被移至过期数组, 时间片已经计算完成. 所以交换数组是  $O(1)$  级调度程序的核心, 其时间开销是换数组指针所需要的时间.

```
% PROS[
  % PROSPROcesstim eschedule(% CHN ASYN p1(*, pout NM): STRING;
    % CHN ASYN p2(pir NM, *): STRING)
= = [
  % ALG[
```

```
% LOC[ p_ thread bck p_ structure p_ priority_max p_ timeleft p_ count STRNG;
  thread release array,  $\vec{rq}$  active,  $\vec{rq}$  expired ARRAY[ array STR NG];
  pid sys ppr pt STRNG] //进程时间片处理参数
LB= START⇒$ OLB= b1;
LB= b1⇒$ O p id= 0∧ $ OLB= b2 //取得父进程 ID
LB= b2∧ p2' ⇒$ OLB= b3;
LB= b2∧ ¬ p2' ⇒$ OLB= b1;
LB= b3⇒$ O p2' ∧ pid∧ $ OLB= b4;
LB= b4⇒$ O p_ structure= pid∧ $ OLB= b5;
LB= b5⇒ p_ count(% iop p_ count l coun p id); //计数
LB= b6⇒$ O p_ count= p_ count+ 1∧ $ OLB= b7;
LB= b7∧ ppr= sys⇒$ OLB= b21;
LB= b7∧ ppr= ¬ sys⇒$ OLB= b8;
LB= b8⇒$ O pt= p_ structure p_ timeleft∧ $ OLB= b9;
LB= b9∧ pt= 0⇒$ OLB= b10;
LB= b9∧ ¬ pt= 0⇒$ OLB= b17;
LB= b10⇒$ O  $\vec{rq}$  expired=  $\vec{rq}$  expired+  $\vec{rq}$   $\vec{rq}$  active[ pid]∧ $ OLB= b11;
  //引入 O(1)算法,运行性能数据将由后续实验工作论述.
LB= b11⇒$ O  $\vec{rq}$  active=  $\vec{rq}$  active −  $\vec{rq}$  active[ pid]∧ $ OLB= b12;
LB= b12∧  $\vec{rq}$  active[ pid]= 0⇒$ OLB= b13;
LB= b12∧ ¬  $\vec{rq}$  active[ pid]= 0⇒$ OLB= b1;
LB= b13⇒$ O p_ array=  $\vec{rq}$  > active∧ $ OLB= b14;
LB= b14⇒$ O  $\vec{rq}$  active=  $\vec{rq}$  expired∧ $ OLB= b15;
LB= b15⇒$ O  $\vec{rq}$  ewpired= p_ array∧ $ OLB= b16;
LB= b16⇒ p_ priority_max(% iop pid/max pid);
LB= b17⇒$ O  $\vec{rq}$  active[ pid]= p_ priority_max∧ $ OLB= b18;
LB= b18⇒$ p1!  $\vec{rq}$  active[ pid] ∧ $ OLB= b19;
LB= b19⇒$ O p1= release∧ $ OLB= b20;
LB= b20⇒$ OLB= b1;
}
}
}
```

逐步求精到本层,已经达到可执行的代码级.

## 4 结语

本文以时序逻辑语言 XYZ/E 对 Open Solaris 多任务、多用户操作系统内核进程进行了形式化描述. 整个过程在统一的一阶时序逻辑架下进行, 确保了系统的语义一致性. 本文对相关技术工作进行了参考与比较<sup>[7-11]</sup>. Ford R L 在文献 [7] 中基于测试的思想对 MK++ 内核及数据结构断言进行分析, 由于测试方法本身的局限性, 不能保证内核实现算法的正确性. Bevier W R 在文献 [9-10] 中对内核保密性, 采用对进程互相隔离, 但它基于 Z 语言并不能将形式语义和可执行算法在统一的时序逻辑框架下自动转换生成. 郭亮在文献 [6] 中基于 XYZ/E 重构 SZRTOS 内核, 给本文工作很好的启示. 用时序逻辑语言 XYZ/E 有利于对 Open Solaris 整个内核进程进行分析、演化及管理. 由于采用了基于逐步求精方法, 引入了 Linux O(1) 调度算法, 提高了进程调度效率, 也增加了系统的灵活性. 因此, 下一步将在此基础上对已经完成的描述与逐步求精内容进行验证和体系结构<sup>[12-13]</sup>建模等工作.

## [参考文献] (References)

- [1] Roger S Pressman 软件工程——实践者的研究方法 [M]. 黄柏素, 梅宏, 译. 北京: 机械工业出版社, 1999.

- Roger S Pressman. Software Engineering——A Practitioner's Approach[M]. Huang Baisu MeiHong Translated. Beijing China Machine Press. 1999 (in Chinese)
- [2] Wing JM. A specifier's introduction to formal methods[J]. IEEE Computer, 1990, 23(9): 8-24
- [3] 唐稚松. 时序逻辑程序设计与软件工程(上册)[M]. 北京: 科学出版社, 2002  
Tang Zhisong. Sequential Logic Programming and Software Engineering[M]. Beijing Science Press. 2002 (in Chinese)
- [4] McDougall Richard Mauro Jim. Solaris 内核结构[M]. 北京: 机械工业出版社, 2007.  
McDougall Richard Mauro Jim. Solaris Internals[M]. Beijing China Machine Press. 2007. (in Chinese)
- [5] Love Robert. Linux 内核设计与实现[M]. 陈莉君, 译. 北京: 机械工业出版社, 2006  
Love Robert. Linux Kernel Development[M]. Chen Lijun Translated. Beijing China Machine Press. 2006 (in Chinese)
- [6] 郭亮. 基于 XYZ/E 重构 SZRTOS 实时操作系统内核[D]. 北京: 中国科学院软件技术研究所, 2002  
Guo Liang. Reconstructing the kernel of SZRTOS real-time OS in XYZ/E[D]. Beijing Institute of Software, Chinese Academy of Science. 2002 (in Chinese)
- [7] Ford R L. Specification-based Automaticity Testing of the MK++ Kernel[R]. Cambridge MA: The Open Group Research Institute. 1997
- [8] Cattel T. Modelization and Verification of a Multiprocessor Real Time OS Kernel[C] // Proceedings of FORTE 94. Switzerland Bern. 1994
- [9] Bevier W R. A verified operating system kernel[D]. Austin: University of Texas. 1987
- [10] Bevier W R. KII: a study in operating system verification[J]. IEEE Transactions on Software Engineering. 1989, 15(11): 1382-1396
- [11] Shapiro JS, Weber S. Verifying Operating System Security. MS-CIS97-26[R]. Philadelphia PA: University of Pennsylvania. 1995
- [12] 朱雪阳, 唐稚松. 其于时序逻辑的软件结构描述语言 XYZ/ADL[J]. 软件学报, 2003, 14(4): 713-720  
Zhu Xueyang Tang Zhisong. A temporal logic-based software architecture description language XYZ/ADL[J]. Software. 2003, 14(4): 713-720 (in Chinese)
- [13] 张广泉. 基于时序逻辑语言描述的监控系统的软件体系结构求精[J]. 计算机工程与应用, 2003, 39(31): 14-17  
Zhang Guangquan. Software architecture refinement for monitor systems based on temporal logic language[J]. Computer Engineering and Applications. 2003, 39(31): 14-17 (in Chinese)

[责任编辑: 严海琳]