Expect 教程中文版

Expect 教程中文版

本文出自: 作者: 葫芦娃 翻译 (2001-09-12 10:00:00)

[版权声明]

[BUG]

有不少部分，翻译的时候不能作到“信，达”。当然了，任何时候都没有做到“雅”，希望各位谅解。

[原著]

Don Libes: National Institute of Standards and Technology
libes@cme.nist.gov

[目录]

1.[摘要]

现代的Shell对程序提供了最小限度的控制(开始，停止，等等)，而把交互的特性留给了用户。 这意味着有些程序，你不能非交互的运行，比如说passwd。有一些程序可以非交互的运行，但在很大程度上丧失了灵活性，比如说fsck。这表明Unix的工具构造逻辑开始出现问题。Expect恰恰填补了其中的一些裂痕，解决了在Unix环境中长期存在着的一些问题。

Expect使用Tcl作为语言核心。不仅如此，不管程序是交互和还是非交互的，Expect都能运用。这是一个小语言和Unix的其他工具配合起来产生强大功能的经典例子。

本部分教程并不是有关Expect的实现，而是关于Expect语言本身的使用，这主要也是通过不同的脚本描述例子来体现。其中的几个例子还例证了Expect的几个新特征。

2.[关键字]

Expect,交互，POSIX,程序化的对话，Shell,Tcl,Unix;

3.[简介]

一个叫做fsck的Unix文件系统检查程序，可以从Shell里面用-y或者-n选项来执行。在手册[1]里面，-y选项的定义是象这样的。

“对于fsck的所有问题都假定一个“yes”响应；在这样使用的时候，必须特别的小心，因为它实际上允许程序无条件的继续运行，即使是遇到了一些非常严重的错误”

dear xiao zeng:

I give an indexed symbolic scheme for the spec for one round of round-robin arbitration.

constr_0_0′= ¬yGrantV ∧ ¬xGrantV ∧ ¬yGrantV′∧¬xGrantV′,

constr_0_1′= ¬yGrantV ∧ ¬xGrantV ∧ yGrantV′∧¬xGrantV′,

constr_0_2′= ¬yGrantV ∧ ¬xGrantV∧¬yGrantV′∧xGrantV′,

constr_0_3′= ¬yGrantV ∧ ¬xGrantV ∧ yGrantV′∧xGrantV′,

constr_1_1′= yGrantV ∧ ¬xGrantV ∧ yGrantV′∧¬xGrantV′,

constr_1_2′= yGrantV ∧ ¬xGrantV ∧ ¬yGrantV′∧xGrantV′,

constr_1_3′= yGrantV ∧ ¬xGrantV ∧ yGrantV′∧xGrantV′,

constr_1_0′= yGrant ∧ ¬xGrantV ∧ ¬yGrantV′∧¬xGrantV′,

constr_2_2′= ¬yGrantV ∧ xGrant ∧ ¬yGrantV′∧xGrantV′,

constr_2_3′= ¬yGrantV ∧ xGrant ∧ yGrantV′∧xGrantV′,

constr_2_0′= ¬yGrantV ∧ xGrant ∧ ¬yGrantV′∧¬xGrantV′,

constr_2_1′= ¬yGrantV ∧ xGrant ∧ yGrantV′∧¬xGrantV′,

constr_3_3′= yGrant ∧ xGrant ∧ yGrantV′∧xGrantV′,

constr_3_0′= yGrant ∧ xGrant ∧ ¬yGrantV′∧¬xGrantV′,

constr_3_1′= yGrant ∧ xGrant ∧ yGrantV′∧¬xGrantV′,

constr_3_2′= yGrant ∧ xGrant ∧ ¬yGrantV′∧xGrantV$'_0$

ant$_0$ = When constr_0_0′ AndList[Is0 req1, Is0 req2, Is0 req3],

ant$_1$ = When constr_0_1′ AndList[Is1 req1],

ant$_2$ = When constr_0_2′ AndList[Is0 req1, Is1 req2],

ant$_3$ = When constr_0_3′ AndList[Is0 req1, Is0 req2, Is1 req3],

ant$_4$ = When constr_1_1′ AndList[Is0 req0, Is0 req2, Is0 req3],

ant$_5$ = When constr_1_2′ AndList[Is1 req2],

ant$_6$ = When constr_1_3′ AndList[Is0 req2, Is1 req3],

ant$_7$ = When constr_1_0′ AndList[Is0 req2, Is0 req3, Is1 req0],

ant$_8$ = When constr_2_2′ AndList[Is0 req0, Is0 req1, Is0 req3],

ant$_9$ = When constr_2_3′ AndList[Is1 req3],

ant$_{10}$ = When constr_2_0′ AndList[Is0 req3, Is1 req0],

ant$_{11}$ = When constr_2_1′ AndList[Is0 req3, Is0 req0, Is1 req1],

ant$_{12}$ = When constr_3_3′ AndList[Is0 req0, Is0 req1, Is0 req2],

ant$_{13}$ = When constr_3_0′ AndList[Is1 req0],

ant$_{14}$ = When constr_3_1′ AndList[Is0 req0, Is1 req1],

ant$_{15}$ = When constr_3_2′ AndList[Is0 req0, Is0 req1, Is1 req2],

ant = AndList[ant$_0$, ..., ant$_{15}$]

cons = Next (grant bvAre grantV′),

assert = ant $\leadsto$ cons

```
let transIJ i 0 width N req =
  (i, i,
  AndList (map Is0 (req subtract [req!i])))
/\ transIJ i j width N req =
  let j′ = (i + j)%N in
  let negReqs = 1 upto (j − 1) in
  let negReqs = map (\k.(req!((k + i)%N))) negReqs in
  let ant = AndList (((map Is0 negReqs)
    union [Is1 (req!j′)])) in
  (i, j′, ant);
let transFromI WIDTH req i =
  let NUM_PORTS = 2 ∗ ∗WIDTH in
  map (\j.transIJ i j WIDTH NUM_PORTS req)
(0 upto (NUM_PORTS − 1));
let transform width grantV grantV′ triple =
val(i, j′, ant) = triple in
  let last = (encode i width ) in
  let newLast = (encode j′ width ) in
  let constr = (constrOfReq last grantV) ∧ constrOfReq newlast grantV′) in
  when constr ant
let transFrom WIDTH req =
  let NUM_PORTS = 2 ∗ ∗WIDTH in
  let triples = flat (map (transFromI WIDTH req) (0 upto (NUM_PORTS − 1)));
  in map (transform width grantV grantV′) triples
let symbIdexAssert WIDTH req grant =
let grantV = vect2Val grant in
let grant′ = map (\str.str”′”) grant in
let grantV′ = vect2Val grant′ in
let ant = AndList (transFrom WIDTH req)@[grant bvAre grantV)in
letcons = Next (grant bvAre grantV′)
in ant ⇝ cons
```