

Verification of file “kernel/resource.c” using BLAST

Ashish Chiplunkar

ashishc@cse.iitb.ac.in

Sumedh Tirodkar

sumedht@cse.iitb.ac.in

Mugdha Bapat

mbapat@cse.iitb.ac.in

December 2, 2007

Abstract

This document is a report of the project done as a part of the course Formal Specification and Verification of Programs (CS615). The project aims at verifying the correctness of functioning of the resource allocation strategy implemented in the file “kernel/resource.c” in linux kernel source code version 2.6.18. This verification is performed using the verification tool BLAST (Berkeley Lazy Abstraction Software Verification Tool).

1 Data Structures

The resource allocation strategy to be verified is implemented by maintaining the resource structures in a tree. A node of the tree contains the fields: “parent”, “sibling”, “child”, “start”, “end”. The nodes in the tree which have a common parent are linked together by a NULL terminated singly linked list using the sibling link and the parent links of all the nodes point to their common parent. The child link of a node points to the head of the linked list formed from all the children of that node. Thus, in order that the nodes indeed form a tree, the following properties must be satisfied.

1. The parent of the child of a node must be the node itself.
2. Nodes connected through the sibling link must have the same parent.

In addition, the resources are maintained in the tree in such a way that any node ‘n’,

1. start field of ‘n’ should be less than or equal to the start field of all children of ‘n’.
2. end field of ‘n’ should be greater than or equal to the end field of all children of ‘n’.
3. end field of ‘n’ should be less than the start field of the sibling next to ‘n’.

The project aims at verifying only the properties in the first set and the remaining ones could be verified in future.

2 Functions

There are in all three functions in the file which need to be verified because they modify the tree.

1. `__request_resource`
Checks if the resource can be attached as a leaf node in the tree. If this is so, the resource is attached and `NULL` is returned; otherwise a pointer to the conflicting resource is returned.
2. `__release_resource`
Searches for a resource in the tree. If found, the resource is detached.
3. `insert_resource`
Inserts a resource in the resource tree. It returns 0 on success, `-EBUSY` if the resource can not be inserted. This function calls `__request_resource`. If `__request_resource` returns with failure, this function attempts to insert the resource as an internal node by making some of the conflicting resources its children.

3 BLAST

BLAST is a software model checker for C programs. The goal of BLAST is to be able to check that software satisfies behavioral properties of the interfaces it uses. BLAST uses counterexample-driven automatic abstraction refinement to construct an abstract model which is model checked for safety

properties. The abstraction is constructed on-the-fly, and only to the required precision.

BLAST checks the reachability of a location **ERROR** and reports that the system is unsafe, if the location is reachable. To check boolean predicates BLAST provides a function **assert** which takes the predicate as input and jumps to **ERROR** if the predicate evaluates to **FALSE**.

4 Results

The project is able to prove the following properties:

1. Node's child's parent should be the node itself provided the child exists i.e. for a node n ,
if $n \rightarrow child \neq NULL$, then $n \rightarrow child \rightarrow parent = n$
2. Node's parent and Node's sibling's parent should be same provided the sibling exists i.e. for a node n ,
if $n \rightarrow sibling \neq NULL$, then $n \rightarrow parent = n \rightarrow sibling \rightarrow parent$

The following problems were encountered while using BLAST.

1. For specifying a precondition, using

```
{_blockattribute((assume(condition)))}
```

often does not result in assumption of the desired precondition. Therefore the precondition has to be assumed by using

```
if(!condition) goto L;
```

with the target location 'L' having the statement

```
L: goto L;
```
2. There are situations when two properties, say $p1$ and $p2$, are correctly verified by BLAST, but the property $p3$, such that $p1 \wedge p2 \rightarrow p3$ is not verified. The following code snippet from the function `__request_resource` serves as an example.

```
assert(p==&root->child||root->child!=NULL); //This is p1
*p = new;
assert(new!=NULL); //This is p2
//assert(root->child!=NULL); // Doesn't work; This is p3
```

3. When two pointers point to same address, and when one of them is updated, the other remains unaltered. For example, in the following code, $backup \rightarrow sibling$ and $*p$ point to same address, but the assert fails even though $*p$ is assigned new.

```
*p=new;  
assert(backup->sibling==new);
```

5 Conclusion

The “kernel/resource.c” was successfully verified as safe for above specified properties. Also, BLAST can be used to verify pointer intensive programs provided the properties to be verified are kept local.