

CS 615 - FSVP Project Report

Naseerudin, Niranjan, Devendra, Aparna

December 2, 2007

Contents

1	Introduction	2
1.1	Verification	2
1.2	Tool	2
2	Verification of <code>do_msgsnd()</code>	2
2.1	Experiments done on verifying the code with the blast	2
3	Results	3
4	Conclusion	3
5	References	3

Abstract

Our project aims to verify certain interesting properties of message passing mechanism implemented in linux kernel 2.6. For this project, we have focused our attention on message sending function `do_msgsnd()` which is a helper routine for `sys_msgsnd`. Here we are tracking whether proper message queue locking and unlocking is happening in sequence. Moreover this message queue locking is done by first acquiring RCU lock and after unlocking the message queue, unlocking of RCU is done. So the properties tracked here are proper sequence of RCU, message queue locking and unlocking.

1 Introduction

1.1 Verification

Verification refers to proving correctness or incorrectness of programs with reference to its intended desired properties. A program is said to be correct if and only if it performs desired functions and nothing less or nothing more. Program verification can be done with approaches like model checking, abstract interpretation, predication abstraction.

1.2 Tool

For verification of program there are many tools like BLAST, ARMC, SLAM, DAGGER each having its own advantages and disadvantages. For our project, we have used tool BLAST (*Berkeley Lazy Abstraction Software Verification Tool*). BLAST is a software model checker for C programs. The goal of BLAST is to be able to check whether software satisfies behavioural 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. We have chosen BLAST because it is able to handle structures, pointers and other basic operations which are there in our chosen code to verify. ARMC could not be selected for our code to verify because it could not handle pointers which are there in our program. SLAM is provided with WINDOWS driver kit, as we are working on linux environment we haven't chosen SLAM to verify our code.

2 Verification of `do_msgsnd()`

`do_msgsnd` is a helper routine which when given message, message queue id and permissions to send, it sends the message into the respective message queue. The interesting property to be tracked here is checking whether message sending is being carried out safely or not that is before sending a message into the message queue we need to get the lock on the message queue and unlock when finished. The other interesting property tracked from `do_msgsnd` is whether RCU lock is carried alternatively or not. These properties are interesting because if these are not verified then the program may go into an inconsistent state.

To track the message queue, RCU locking and unlocking property we need to include predicates such as `rcu_lock_status` and message `lock_status`. To track these predicates we first used arrays to store the status of the lock, but found that BLAST does not support arrays. Then BLAST was not tracking some required properties so we need to do refinement for our set of predicates that are to be tracked

2.1 Experiments done on verifying the code with the blast

- Initially we kept two observer variable one for rcu lock status and another for verifying msq queue lock status and verified if status is 0 before locking and status has to be 1 before unlocking.
Problem: since we are using observer variables this can be correct when there is single message queue, but would not be correct abstraction when

we have multiple queues,i.e we cannot differentiate between different queues, to do this we need arrays to keep separate status for different msg queues. since arrays were not supported in BLAST we could not do this. So we have planned to store the status of msg queue in its kern ipc perm struct by adding `lock_status` which we have abstracted out for `spinlock_t` structure as one variable in structure.

- We have verified whether we can pass the pointer to the structure to the function by using some sample programs ,but the thing we observed was we need to initialize that struct before passing it to the other function so before calling to the function we have initialise the struct whenever required.we have modified the value of variables in struct and verified if it is reflected in the caller function.we have observed that this was not working for most of the options , it was working correctly with `talias` so we have used `-talias` option through out the verification of our program. Result we were able to pass pointer to the function and the values modified in the callee was reflected in the caller.But it could not track if we have changed more than two variables of struct, for this we had to add explicitly what all predicates to be tracked. so we have asked blast to track the predicates by using if statement.So this helped us in learning how to add predicates to the program.
- We have lot of type casting of structures in our program so we have verified if structure type casting was working in blast but it seems not supported.

3 Results

The code was proved to be SAFE when the necessary predicates were tracked. This has been clearly tested by inserting the code which violates safetiness, and BLAST has given output as UNSAFE. In other words if the sequence of message queue, RCU locking and unlocking was changed it was proved to be UNSAFE by BLAST.

4 Conclusion

A function called `do_msgsnd` from linux kernel 2.6 code has been taken which sends message into the message queue, for verifying its correctness. For its verification a tool called BLAST has been taken as it handles the requirements of the code. Set of predicates were taken and verified, and it has been proved to be SAFE by BLAST. During verification we learnt that BLAST cannot handle arrays, and whenever it is not tracking a certain property then we need to specify BLAST to track that property. Different options of BLAST has been used to verify the code. Finally BLAST was able to handle the required properties in a simpler way, and had proved the chose code to be SAFE.

5 References

1. Linux kernel code was taken from <http://lxr.linux.no/linux/ipc/msg.c#L627>