'Laurel and Hardy' Model for Analyzing Process Synchronization Algorithms and Primitives

Mithun Acharya and **Robert Funderlic**

Department of Computer Science
North Carolina State University
Raleigh, North Carolina USA
<{mpachary,ref}@csc.ncsu.edu>

Abstract

The initial software solutions to solve the Process Synchronization problems, which ultimately led to the use of Semaphores are usually hard for a beginner to understand and appreciate. The reason is that one has to do lots of book keeping in tracing these algorithms, which involve multiple processes and variable instances. In order to prove the correctness of any synchronization problem, it has to be tested over the Critical Section conditions which again might be a tough exercise. In this paper we propose a model called "Laurel and Hardy" to understand and prove the (in)completeness of the software solutions to the Process Synchronization problem. A series of dramas is enacted between the two comedians Laurel and Hardy which makes understanding of synchronization problem solutions and testing it over Critical Section conditions very simple. We attempt to convince the readers that Laurel and Hardy model can be used as a potential tool for the analysis of any synchronization algorithms and also in assessing the (in)completeness of it. In addition, Laurel and Hardy model can be used as a teaching aid in beginner's Operating System course.

1. Introduction

The most important concept in Process Synchronization is that of a Critical Section. In a system comprising of 'n' processes each process has a segment of code called a Critical Section in which the process may be changing common variables, updating a table, writing a file and so on [1]. The challenge is to devise algorithms in which all the process enters the Critical Section in a mutually exclusive fashion while also satisfying other Critical Section conditions that are listed later in section 1. Therefore, once the algorithm is devised it has to be tested over the Critical Section conditions. Laurel and Hardy model does exactly this. The rest of the paper is organized as follows. Section 2 lists the terminologies used in this paper. Section 3 explains the Critical Section conditions and their inter relationship. Section 4 outlines the Laurel and Hardy model. Section 5 lists series of algorithms from [1] and [2] and tests it over the Critical Section conditions. Section 6 concludes the paper.

2. The Terminologies

We use P to denote a process. Unless otherwise stated, we assume only two processes P_0 and P_1 for simplicity. The general structure of any process P_0 is as follows

```
Repeat
    Entry section (E)
    Critical Section (C)
    Exit Section (X)
    Remainder Section (R)
until false
```

We use C', E', etc. to refer to the corresponding sections of process P_1. Throughout the paper we mean P $\rightarrow$ C to mean process P requests the Critical Section or is waiting to get into the Critical Section and P $\leftarrow$ C to mean P gets the Critical Section.

3. The Critical Section Conditions and their relations

Any solution to the synchronization problem is said to be "correct" only if it satisfies the four conditions listed below [1] [2] ([2] modifies [1] to some extent):

1. Mutual Exclusion (m): If P is executing in its Critical Section, then no other process can be executing in the Critical Section. We say a Critical Section algorithm violates 'm' if it is possible, no matter how unlikely, that two processes can have their program counter (PC) pointing at the Critical Section instructions at the same time.

2. Strict Progress (p) : if P $\rightarrow$ C and if {} $\leftarrow$ C then P $\leftarrow$ C. For multiple processes this means that if no process is executing in Critical Section and there exists some process that wish to enter their Critical Sections, only these processes participate in the decision of which will enter its Critical Section next and this decision

cannot be postponed indefinitely. One will surely enter its Critical Section.

3. Bounded Waiting (b): If $P \leftarrow C$ and $Q \rightarrow C$, then there is a least bound beta on the number of times P may exit the Critical Section and return before $Q \leftarrow C$.

4. Starvation Free (s_f): If $P \rightarrow C$ then P cannot wait indefinitely before $P \leftarrow C$. If a Critical Section algorithm is starvation free (s_f) then it can be characterized as
   - Deadlock Free (d_f): No deadlock for Critical Section entry
   - Abandonment Free (a_f): If one process exits then it should not be the case that some other process cannot enter at all.
   - Unlucky Free (u_f): We say that a process P is unlucky if whenever P gets the CPU, it is so unlucky that always it finds another process inside the Critical Section.

We now give some relationship between the Critical Section conditions without rigorous proofs. If an algorithm is not Deadlock Free, it cannot satisfy Strict Progress. This is because there would be processes waiting outside the Critical Section in deadlock, which means that even though the Critical Section is empty, some processes cannot enter the Critical Section. Putting mathematically, this means NOT d_f => NOT p, which is logically equivalent to p => d_f

If an algorithm is not Abandonment Free, it cannot satisfy Strict Progress. This is because there would be processes waiting outside the Critical Section even though there are no processes inside the Critical Section (A process might have finished execution abandoning somehow the other process from entering the Critical Section forever). Therefore NOT a_f => NOT p which is logically equivalent to p => a_f

Similarly one can see that Bounded Waiting implies Unlucky Free since every process will get into the Critical Section eventually. Therefore b => u_f or NOT u_f => NOT b. To summarize,

```
p => d_f   OR   NOT d_f => NOT p
p => a_f   OR   NOT a_f => NOT p
b => u_f   OR   NOT u_f => NOT b
```

We use these results in our analysis. In the rest of the paper, we assume that there are only two processes P_0 and P_1 for the sake of simplicity.

4. The Laurel and Hardy model
We enumerate six points below that forms the model:
1. Two processes are named as Laurel and Hardy
2. The Critical Section is modeled by a room which has transparent walls [for some reason that will become clearer later on]

3. The room has one entry door with lock and one exit-only door
4. A person unfreezes on getting the CPU and freezes on losing the CPU
5. flag = 1 is modeled by a person waving the flag and turn = 1 is modeled as a person getting the key
6. Freezed person can still wave the flag

We note that waiting outside the glass room near the entry door denotes Program Counter (PC) pointing to Entry Section and being outside the exit door to PC pointing to Remainder Section. In the next section, we show how this simple model can be used to test the synchronization algorithms over Critical Section conditions

5. The application of Laurel and Hardy model
The algorithms that are listed below are either in [1] or [2]. We take each algorithm in turn, use the model to understand the algorithm and then test it over the Critical Section conditions.

5.1 Turn algorithm
As said earlier we assume that there are only two processes P_0 and P_1 and all the code is with respect to P_0 and we assume that P_0 is Laurel and P_1 Hardy. Non primed variables like C refer to P_0 and primed variables like C' refer to P_1. The algorithm is as follows

```
boolean turn; // shared variable
turn = 0;

repeat
    while (turn !=  0) do no-op;
        Critical Section
    turn = 1;
    Remainder Section
until false;
```

5.1.1 Explanation of the algorithm using Laurel and Hardy model
If Laurel does not have the key he does nothing. If he gets the key, he enters the room, comes out of the room and gives the key to Hardy. This algorithm does not satisfy Abandonment Free (a_f) and Strict Progress (p). We explain this using the Laurel and Hardy model in the next two subsections

5.1.2 Turn is NOT a_f
We give a sequence that shows that Turn algorithm is NOT a_f. The sequence is E(turn=1)RC'E'(turn=0)R'(P_0exits). Putting in words, this means that P_0 goes to exit section, makes turn = 1 and goes to the Remainder Section. At this point context switch occurs and P_1 gets the chance. Now P_1 enters the critical section, goes to the exit section, makes turn = 0 and goes to the Remainder Section. At this point context switch occurs and P_0 exits. Now since turn = 0 forever, P_1 never gets the chance to enter the Critical Section again. Now we explain this using the Laurel and

Hardy model and see how easy it is to explain the procedure.

Laurel comes out of the room, gives the key to Hardy. Hardy goes into the room, comes out, and gives the key to Laurel and Laurel runs away with the key. Poor Hardy has no keys and he can never enter!

5.1.3 Turn is NOT p

Now we prove, using the Laurel and Hardy model, that Turn does not satisfy the strict progress requirement.

Laurel comes out and gives the key to Hardy and starts reading the newspaper. Hardy goes in, comes out, and gives the key back to Laurel. However, the news hungry Laurel continues to read the newspaper leaving the industrious hardy outside the room even though no one is in the room. The sequence is X(turn = 1)RC'X'(turn = 0)E'R'R'R'R'(P_1 cannot enter C' even if Critical Section is empty).

The beauty of the model is that just by realizing a real life situation, one can prove the incompleteness of a given algorithm. One need not work out the sequence, which may be tougher for more complicated algorithm. Laurel and Hardy model encourages the reader to think in terms of real life situations than worrying about keeping track of variables and context switches. Once the story is weaved with Laurel and Hardy model, which proves that a critical section condition is not met, getting a sequence is trivial. For the remaining algorithms, we only give the 'story'. It is a trivial exercise for the reader to enumerate the sequence from the 'story'.

5.2 Flag algorithm

```
repeat
    flag[0] = true;
    while (flag[1]) do no-op;
        Critical Section
    flag[0] = false;
    Remainder Section;
until false;
```

5.2.1 Explanation of the algorithm using the model

Laurel starts waving the flag. If he sees Hardy too waving the flag, he does nothing. As soon as Hardy stops waving the flag, he enters the room, still waving the flag [Now the transparent walls come into picture. Hardy who is outside can still see what Laurel is doing inside]. Laurel comes out and stops waving the flag.

5.2.2 Flag algorithm is NOT d_f

Both Laurel and Hardy start waving the flag outside the room simultaneously and they start cluelessly hoping for the other one to stop. Each of them freeze and unfreeze forever. However, freezing won't deter these determined comedians from forever waving the flag! Since NOT d_f => NOT p, flag algorithm does not satisfy Strict Progress too.

5.3 Modified Flag algorithm

```
repeat
    while (flag[1]) do no-op;
    flag[0] = true;
        Critical Section;
    flag[0]=false;
    Remainder Section
until false;
```

5.3.1 Explanation of the algorithm

Laurel checks to see if Hardy is waving the flag before he himself starts waving. If Hardy is not waving, he starts waving and then enters the room. He eventually leaves the room and stops waving.

5.3.2 Modified Flag algorithm does not satisfy m

Laurel goes in and before he can start waving the flag, swift Hardy too goes inside the unlocked door and starts waving. Both find themselves inside the glass room.

5.3.3 Modified Flag algorithm does not satisfy u_f

Hardy is freezed. Laurel goes in, starts waving the flag, comes out, stops waving and before Hardy can unfreeze, Laurel again goes in and freezes. Hardy unfreezes and sees freezed Laurel still waving the flag [walls are transparent and freezed persons can wave the flag]. This happens all the time and poor Hardy never enters! Since NOT b_f => NOT b, Modified FLAG algorithm does not satisfy bounded waiting.

5.4 Combination Algorithm

By combining the ideas of Turn and Flag algorithm, we get the following algorithm

```
repeat
    flag[0] = true;
    turn = 1;
    while (flag[1] and turn) do no-op;
        Critical Section
    flag[0] = false;
    Remainder Section
until false;
```

5.4.1 Explanation of the Combination algorithm

Laurel starts waving the flag and gives the key to Hardy. As long as Hardy is waving the flag and Hardy has the keys, Laurel does nothing. Else, he enters still waving the flag and stops waving the flag as soon as he exits.

5.4.2 Combination algorithm is NOT p

Laurel comes out and starts waving the flag and freezes. Hardy too starts waving the flag and foolishly passes on the keys to freezed Laurel. Now Hardy realizes that he cannot enter the room since freezed Laurel has the key and is still waving the flag outside the room.

5.5 Test-and-Set algorithm

```
boolean lock = false; //shared variable

repeat
    while Test-and-Set(lock) do no-op;
        Critical Section
    lock = false;
    remainder section
until false;

boolean Test-and-Set(boolean target)
    begin
        if (target == false);
            target = true;
        return target;
    end
```

The Test-and-Set instruction is executed atomically, that is as one uninterruptible unit. Thus, if two Test-and-Set instructions are executed simultaneously each on a different CPU), they will be executed sequentially in some arbitrary order.

5.5.1 Explanation of Test-and-Set algorithm
Laurel checks to see if the door is locked. If it is unlocked, he goes in and locks the door. All these happen before Laurel can freeze the next time (which means Test-and-Set is atomic). When Laurel goes out, he unlocks the door.

5.5.2 Test-and-Set is NOT u_f
Laurel comes out. Hardy is freezed. Laurel gets in and freezes again before Hardy can unfreeze. When Hardy unfreezes he sees freezed Laurel inside the room and waits. This happens forever. Clearly NOT u_f => NOT b

6. Conclusion
The unsuccessful software solutions to the Critical Section problem ultimately led to the concept of Semaphores, Mutexes, Monitors and other synchronization primitives which are aided by the hardware. We can easily use the Laurel and Hardy model to explain any synchronization algorithms that involve these primitives. The model is powerful when it comes to pointing out the flaw in a synchronization algorithm. It may not be effectively used to prove that the algorithm satisfies all the Critical Section algorithms. Nevertheless, when it comes to understanding and pointing out the flaws of any synchronization solution, Laurel and Hardy model comes in handy. It frees us from keeping track of myriad of variable instances in a multiprocess environment. Above all, it can be used as a teaching aid in the beginner's Operating System course.

References
[1] Silberschatz. A., Galvin. P. B., Gagne. G.,"Operating System Concepts", Sixth Edition, John Wiley and Sons Inc., ISBN 0-471-41743-2, June 2001
[2] Funderlic. R., "Operating Systems [CSC 501] Course Pack", Department of Computer Science, North Carolina State University, August 2001.

SIGCSE 2004 Doctoral Consortium

The SIGCSE Doctoral Consortium is an annual event held in conjunction with the SIGCSE Technical Symposium each spring. It is designed primarily for Ph.D. students in computing-related areas who are planning a career in academia. Students whose research focus is related to CS education are a primary focus, although doctoral students in any computing-related area are welcome to apply.

Link from the website

<http://www.csc.vill.edu/sigcse2004/>