Documentation

**Crash Analysis Tool for Profiling Multi-Tasking Applications**

by

Faizan Ullah

Supervisor

Prof. Anjum Naveed (ext.)

Dr. Adnan Iqbal

Department of Computer Science

Namal College, Mianwali (An Associate Institute of University of Bradford, UK)

2018-2019

Iteration 2.0

 Date of Revision: November 14, 2018

# Introduction

## Abstract

A software utility capable of analyzing crashes of multitasking application. With its help, developers can reassure that the application has the ability to handle anomalies like power off, sudden kill, accidental resources snatching, race-condition, resources unavailability, permanent locking and etc. In general, utility’s focus will be on the unhandled stray locks that might be of an effect in any case of the anomaly stated above. The common effect could be no restarting of the application, not able to score a particular recourse, crashing several times and many more. The purpose of this software is to point out what could go wrong in case of any irregularity.

## About the document

In this write-up, I’ll be going to include basic user level understanding of what this tool might be, what is the motivation behind this project and what will be the steps and procedure I’ll be taking to make this idea into a working application.

## Motivation

In multitasking application, threads/processes use locks to communicate with other tasks about the resource they are currently occupying. Sometimes the lock remains unhandled as the task had to stop abruptly or it was being killed by the kernel. These stray locks compel the software to behave oddly as in many cases the utility might not even start or needs a complete reboot or sometimes the lock is permanent which needs extensive manual debugging before the user is able to use it again. MYSQL Database is one example of it that use file lock to acquire a socket which has beside of some plus points more adverse effects as in case of system failure the DBMS won’t get the chance to leave the resource and in future the file is already there, and system consider that socket already being locked and not allowed to acquire.

This is one example and there are countless to make oneself familiar with the complication an incautious behavior of a programmer arises.

Keeping in mind the problems and the efforts end user might be needing to put in, I took this challenge to devise a tool that’ll make rigorous disruption in the locking mechanism of a software and show the developer exact spot where the problem is arising.

This is a challenging task and very few people try to work at this level. I wanted to go deep into the kernel, deep enough to cherish the things many developers take for granted.

## Implementation Methodology

Implementation methodology requires the following steps to accomplish for completing this software. Before getting to these steps we are required to get yourself acquainted with some related software like strace, ltrace, lsof, and some system calls like ptrace, execvp, and techniques of Inter process communication (IPC) like signals and system locks etc. These prequels will help us get our main part done with ease.

### Detection

This step requires us to build individual detection system for different types of locks. Every type of lock mutex, semaphore, spinlock etc. requires a sperate identification method.

### Back Trace

Printing the Back Trace of a specific call. This may include steps that lead to this call. The stack can be a great help in this case.

### Which resource is lock

To generate a log having all the description of the resource that has been locked.

### Which process is lock

To generate a log having all the description of the process that has been locked.

### What type of lock

To generate a log having all the description of the types of locks on the process and resource.

### Testing some of well-known software

In the end for validation purposes, some well-known application and some pre-written fake application must be tested in order to get the best results from the utility.