**1. Introduction**

ITP 30002-02 Operating System, 2023 Spring

**Homework 2**

Seongbin Kim 22100113, Gunmin Yoo 21900458 {22100113@handong.ac.kr, 21900458@handong.ac.kr}

Our minimizing delta debugging algorithm *ddmin* is fed with a failing test case, which it simplifies by successive testing. It stops when a minimal test case is reached, where removing any single input entity would cause the failure to disappear. (original paper) This is surprising, because re-testing a program under changed circumstances is a common debugging approach and the only way to prove that the circumstances actually cause the failure. Eventually, we expect that several debugging tasks can in fact be stated as search and minimization problems, based on automated testing and thus be solved automatically.

**2. Approach**

In this section, we first introduce the necessary requirements (Section 2.1). Then, we introduce the detailed approach of cimin (Crash Input Minimizer) (Section 2.2).

**2.1 Requirements**

The requirements for cimin, our Crash Input Minimizer program, are based on two criteria, system functions and production of the minimized input. The goal is to produce an input that preserves the crash status while communicating through system functions to provide functionality (e.g., fork(), pipe()).

**2.2 Structure of CIMIN**

From the given requirements above, cimin is implemented by the following flow, processing input options, minimizing the crash input, running the program, communicating between processes, and ending the program.

**2.2.1 Process of input options**

The program receives input options via command-line arguments. Using the C library[[1]](#footnote-1), the function getopt() parses the input options. The additional arguments required when executing the target program are listed after the binary execution file of the target program. The program reads the list of additional arguments required for the target program when executing it and assigns them to a separate array. This is to send the list to the target program as \*argv[] through execv().

**2.2.2 Minimization of the crash input**

Our algorithm of minimizing the crash input is based on the *delta debugging[[2]](#footnote-2)* algorithm. Based on the output of the execution of the target program with the modified input, the algorithm explores the subsequences. To compare the crashing messages, strstr() is used.

**2.2.3 Execution of the target program**

cimin executes the program to obtain the crashing message, and compare to the desired message (*-m* of our input option). Through the child process, created via fork(), execv() is executed.

**2.2.4 Inter-communication between processes**

As execv() replaces the image of the current process with the target program’s process image[[3]](#footnote-3), fork() is required to safely execute the target program. The main components used for the communication to work properly were, pipe(), signal, and dup2().

Two pipes were used for redirecting I/O, and dup2() to send and read via standard input or error. For the pipe from the parent to the child, the input is sent via standard input. In the case of the child to the parent, standard error outputs are redirected for the parent to read as a crash message. Exceptional cases such as, an early end to the program or an infinite loop, were delt with signal communication. A signal alarm is reset for 3 seconds every execution, and every output is saved right away for preemption toward sudden termination signals.

**2.2.5 Termination of the program**

For the termination of cimin, the requirements are clear and concise. Our approach prints specification of the result of the program, whether it was ended predetermined (via Ctrl+C), or not. The size of the final minimized input, and an output file with the input. Last but not least, the process identification number is saved for termination. This is to prevent a continuous program executing whilst the parent is ended.

**3. Evaluation**

In this section we provide a demonstration of cimin using a few test cases. Then, our evaluation of our program based on our implementation and results with follow.

**3.1 Demonstration of CIMIN with test cases**

The criteria for correction using test cases are defined as the following: using the output file, which contains the derived reduced input, if the test case program works with the output file sent through standard input it is considered valid. Figure 1. shows the result of producing the desired crash in the original test case is as follows:

$cd libxml2

$./xmllint -–recover -–postvalid - < testcases/crash.xml

…

==ERROR: AddressSanitizer: SEGV on unknown address…

…

<Figure 1. Crash result of provided test case>

Figure 2. is an actual implementation of reproducing the crash using the output file *reduced* and is as follows:

$cd libxml2

$./xmllint -–recover -–postvalid - < ../cimin/reduced

…

==ERROR: AddressSanitizer: SEGV on unknown address…

…

<Figure 2. Crash result of reduced crash input>

**3.2 Dealing with *balance* using a global data structure**

The *balance* test case provides a special condition to solve. The failing input requires the program to timeout the process when it passes the time limit (e.g., 3 seconds). When the program is signaled to end, it should provide a saved output of the crash-causing input. Preserving and updating the current data (reduced input) every iteration of the minimization process to a global data provided this functionality. Figure 3. describes the global data structure.

struct handler\_args {

int kill\_pid;

int length;

char\* output\_string;

char\* output\_filename;

} global\_handler;

<Figure 3. Global Data Structure>

Through this global data structure it is possible to solve both passing and failing test cases of *balance*. Saving the data prior and throughout the execution of cimin, when the program is suddenly ended or given a non-crashing input, cimin provides the initial input.

**3.3 Finding the minimum crash input for *jsmn[[4]](#footnote-4)***

Our approach for *jsmn* provided the clearest view of the process of reducing the crash input. Figure 4. illustrates a snippet of the input being minimized to the minimal.

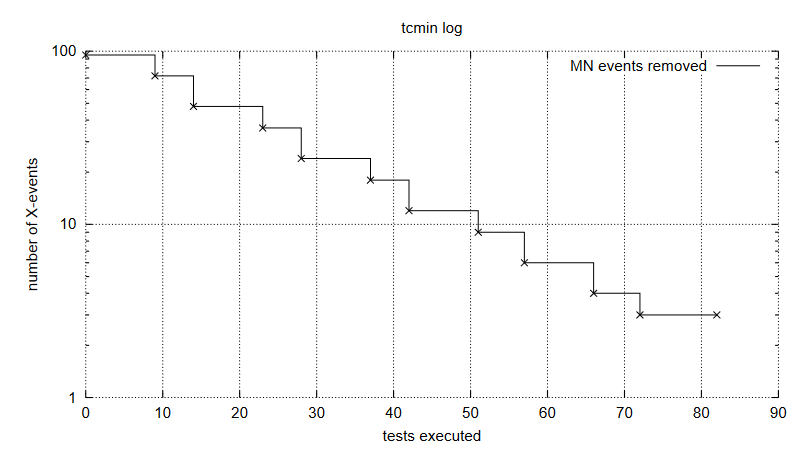
1 <SELECT NAME="priority" MULTIPLE SIZE=7> 8  
2 <SELECT NAME="priority" MULTIPLE SIZE=7> 4  
3 <SELECT NAME="priority" MULTIPLE SIZE=7> 4  
4 <SELECT NAME="priority" MULTIPLE SIZE=7> 4  
5 <SELECT NAME="priority" MULTIPLE SIZE=7> 8  
6 <SELECT NAME="priority" MULTIPLE SIZE=7> 8  
7 <SELECT NAME="priority" MULTIPLE SIZE=7> 4  
8 <SELECT NAME="priority" MULTIPLE SIZE=7> 4  
9 <SELECT NAME="priority" MULTIPLE SIZE=7> 4

<Figure 4. Reducing crash input of *jsmn*>

Starting off with the initial *crash.json* file, the *minimize input* algorithm reduces the message string character-by-character.

**3.4 Reducing *libxml2[[5]](#footnote-5)*’s crash input**

*libxml2* had additional arguments to process. We had to deal with tighter constraints such as, having to add a NULL value at the end of the 2D-array of arguments. The execution of the program results with a reduced input. Figure 5. provides a preview of an example of the length of the input reducing.



<Figure 5. Reduction of *libxml2*’s crash input>

The results show after … tests, … characters are reduced. The test demonstrated in Section 3.1 shows the resulting crash input is valid.

**4. Discussion**

One of the definite signs of having an effective knowledge is to apply it to the topics beyond the given material. This section is to present your ideas, thoughts, lessons learned, suggestions, opinions that you have conceived while working on the given tasks.

**5. Conclusion**

This section summarizes what you had presented in the report so far. It is a good idea to recap key points of your homework, which can tell readers that you had effectively learned the topic by accomplishing all required tasks.

Convert your report to a PDF and submit the PDF version.

1. unistd.h [↑](#footnote-ref-1)
2. Andreas Zeller and Ralf Hildebrandt, Simplifying and Isolating Failure-Inducing Input, IEEE Transactions on Software Engineering 28(2), February 2002, pp. 183-200. [↑](#footnote-ref-2)
3. https://linux.die.net/man/3/execv [↑](#footnote-ref-3)
4. https://zserge.com/jsmn/ [↑](#footnote-ref-4)
5. https://gitlab.gnome.org/GNOME/libxml2/-/wikis/home [↑](#footnote-ref-5)