

## Homework 7, ECE 590 & CS320 Software Reliability.

1. In the lecture on fundamentals of software aging, we saw that an AR-fault (ARB) activation occurs under the presence of aging factors. Describe in your own words, what does it mean for a computer program execution to have the presence of aging factors? Give one real or hypothetical (but realistic) example.

It means during our coding, we might use some resources without freeing them, it would cause some problems such as memory leak. When this piece of code ran many times, it would occupy more and more resources. Finally, it would cause a crash. For example, when we learn the “memory leak” concept, my teacher Drew told us “Dukehub” has a memory leak, leading it should reboot every several months. It is an example of ARB.

2. The C language code listed below contains a fault that is classified as an AR-fault (ARB). Based on this code describe the

0. a) AR-fault.

1. b) AR-error caused by the activation of the AR-fault identified in (a).

2. c) Aging effect caused by the causality chain composed of (a) and (b).

- a) AR-fault: parent process does not call wait/waitpid to free the child’s pid resource

- b) AR-error: will continuously occupy the pid without freeing them in the while loop.

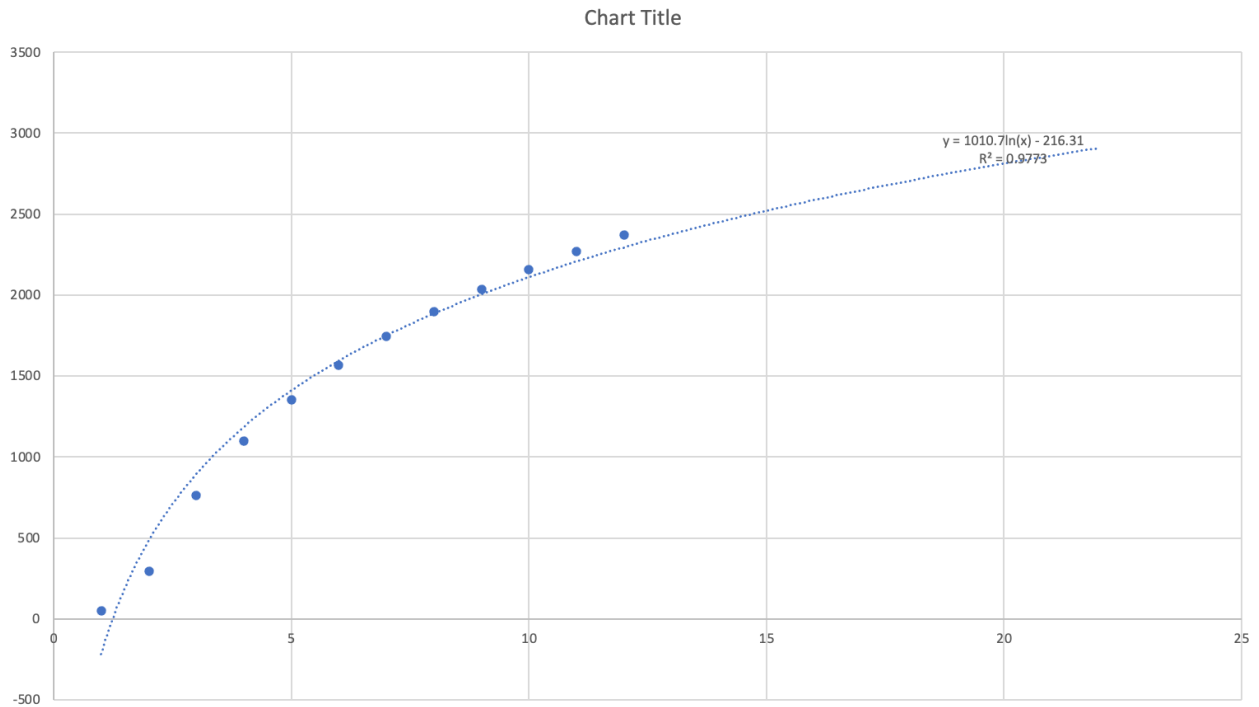
- c) Aging effect: The number of processes available to the system is limited, if in the while loop, no children’s pids are freed, the system will not be able to create new processes because there are no process numbers available. It means during our coding, we might use some resources without freeing them, it would cause some problems such as memory leak. When this piece of code ran many times, it would occupy more and more resources. Finally, it would cause a crash. For example, when we learn the “memory leak” concept, my teacher Drew told us “Dukehub” has a memory leak, leading it should reboot every several months. It is an example of ARB.

3. The embedded software that controls a military satellite suffers from software aging. This satellite’s availability must be no less than six nines (99.9999%), which means a yearly downtime no longer than 31.56 seconds. Because of the accumulation of the aging effects, the size of the satellite’s main application process grows progressively. The memory size of this process has been monitored monthly, for one year, and the result follows (in megabyte): 50, 293, 763, 1097, 1355, 1567, 1745, 1900, 2037, 2159, 2269, 2370. It is known that once

this process grows beyond 2.8 gigabytes, it will be abruptly forced to terminate by the satellite operating system. Taking into consideration the above-described scenario, answer the following questions:

1. a) Based on the monitored data, is the aging effect accumulation rate linear or non-linear? Justify your answer.
2. b) What is the expected time to AR-failure (in months)? Justify your answer. (Show how you derive the answer)

a) it is non-linear. I plot it in matlab:



- b) I assume it as a log function, it would reach 2.8 gigabytes (2867 megabyte) in month 10 later (22 months), which is 2907. In 9 months later, it is 2860. Therefore, the AR-failure would happen in 10 months later.

According to the specification for the C code listed below, its correct output is expected to be “Counter Value: 100”. Based on this expected value and the taxonomy we saw in the lecture, clearly keeping in mind the concepts of fault, error, and failure, answer the following questions:

1. a) Based only on the analysis of the source code, is this program fault-free? Justify your answer.

No, it does not have the mutex lock, could lead to race condition.

2. b) If the answer to (a) is No, is there an AR-fault (ARB)? Justify your answer.

It is an AR-fault because it causes the accumulation of numerical errors(More and more plus 1 operations are being ignored).

3. c) Assume you are responsible for the acceptance testing of this program. Implement the acceptance test, define the number of test executions for this program, and perform it. For each execution, mark if the test “passed or failed”.

Attach your program and a screenshot of your test results in your submission.

Test.sh:

```
#!/bin/bash
for a in `seq 100`
do
./test
done
```

Makefile:

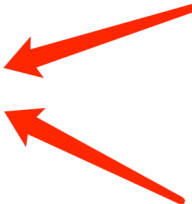
```
test: test.cpp
    g++ -o test -Wall -Werror -std=gnu++11 -pedantic -g -gdb test.cpp -lpthread
```

Test.cpp

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <pthread.h>
5  #define N_THREADS 100
6  int counter;
7  void *thread_code (void *arg) { counter = counter + 1; pthread_exit(0);
8  }
9  void start_threads() { int i;
10 pthread_t threads[N_THREADS]; counter = 0;
11 for (i = 0; i < N_THREADS; i++) {
12 pthread_create(&threads[i], NULL, thread_code, NULL); }
13 for (i = 0; i < N_THREADS; i++) { pthread_join(threads[i], NULL);
14 }
15 if(counter==100){
16     printf("100 passed\n");
17 }else{
18     printf("%d failed\n", counter);
19 }
20 // printf("Counter value: %d\n", counter);
21 }
22 int main() { start_threads(); return EXIT_SUCCESS;
23 }
```

Output:

```
p1189@vcvm-24744:~/ece590reli/proj7$ sh test.sh  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 failed  
100 passed  
98 failed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed  
100 passed
```

Two large red arrows are drawn over the terminal output. The first arrow starts near the top right and points left towards the line "100 failed". The second arrow starts further down on the right and points left towards the line "98 failed".

We can see two ‘failed’ in the output.

4. d) Based on the results obtained in (c), do you recommend this program to be released and deployed, or to return to the test team?  
Justify your answer.

I recommend this program to return to the test team because it has race condition. Test team should fix this bug.