To Predict ▸ To Design ▸ To Perform

College of Engineering
Department of Mechanical & Industrial Engineering

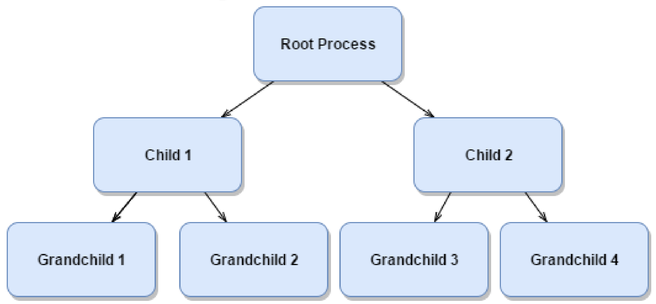
ME, ECE, BE Capstone Design Programs

Multithreaded Fault Injector

Maxim Bankston, Travis LeCompte
Sponsors: Dr. Lu Peng

Objectives
- Dynamically simulate soft errors in multithreaded program execution
- Provide documentation and user interface for system
- Goal is to use system to analyze Big Data applications

Multithreading



- Split large task into smaller, parallel tasks
- Extremely common in Big Data applications for efficiency

Soft errors

1	0	1	1

Original Data (value = 13)

Bit Flip

1	0	1	0

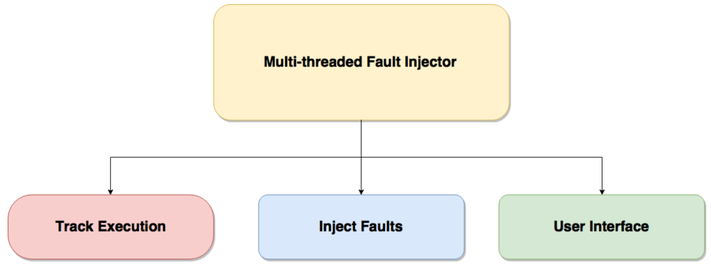
Corrupt Data (value = 12)

- Fault in data during execution
- Caused by radiation (internal and external to machine)

Specifications

System	Build for Ubuntu 16 and C/C++
Threads	Handle four threads minimum
Injection method	Dynamic fault injection
User interface	Automate trial execution and result aggregation

Conceptual Design



Code Samples

```
bool corruptIntData_1bit(int fault_index, int inject_once, int ef, int tf, int byte_val, char inst_data) {
        if(!is_kulfi_enabled) return (bool)inst_data;
        unsigned int bPos;
        incrementFaultSiteHit(fault_index);
        fault_site_count++;
        fault_site_intData1bit++;
        if(inject_once == 1)
                ijo_flag_data = 1;
        if(ijo_flag_data == 1 && fault_injection_count>0)
                return inst_data;
        if(!shouldInject(ef, tf)) return inst_data;
        if(bit_position == 0) {
                fault_injection_count++;
                printFaultInfo("1-bit Int Data Error", bPos, fault_index, ef, tf);
                if(inst_data) return false;
                else return true;
        } else {
                printf("Fault not injected because the set bit position is > 0");
                return inst_data;
        }
}
```

- Inject into single bit data type
- One for each data type (1, 4, 8, 16, 32, 64 bits)

```
INTERCEPTOR(int, pthread_create, pthread_t* thread, const pthread_attr_t* attr, void* (*start_routine) (void *), void *arg){
        pthread_mutex_lock(&mux);
        REAL(pthread_create)(thread, attr, start_routine, arg);
        log(pthread_self(), *thread);
        pthread_mutex_unlock(&mux);
        printf("Intercepted thread creation: thread %lu spawned thread %lu\n", pthread_self(), *thread);
}

INTERCEPTOR(int, pthread_join, pthread_t thread, void ** retval){
        int x = REAL(pthread_join)(thread, retval);
        printf("Intercepted thread (%lu) joining with status %d\n", thread, x);
}

void InitializeTracking(){
        INTERCEPT_FUNCTION(pthread_create);
        INTERCEPT_FUNCTION(pthread_join);
        pthreadNode me = pthreadNode(pthread_self(), 0);
        struct timeval now;
        gettimeofday(&now, NULL);
        g_creation_time[me] = now;
}
```

- Intercept and log information
- Stores thread information in tree

Example output

```
[Fault Injection Campaign details]
   Max interval: 10000000
Reading configuration from environment variables.
   Next fault CountDown = -1
   Should initialize randseed = 0
   Bit position for faults=-1
   Dump BB Trace=0
Intercepted thread creation: thread 140367842170688 spawned thread 140367817524992
Intercepted thread creation: thread 140367842170688 spawned thread 140367809132288
Intercepted thread (140367817524992) joining with status 0
Intercepted thread (140367809132288) joining with status 0
Result: 5100
140367842170688 (0) 1492740495 662696
        140367809132288 (0) 1492740495 662949
        140367817524992 (0) 1492740495 662863

/********************Fault Injection Statistics*************************/
Total # fault sites enumerated : 1058
Categorization of fault sites individually enumerated:
Total # 8-bit  Int Data fault sites enumerated : 0
Total # 16-bit Int Data fault sites enumerated : 0
Total # 32-bit Int Data fault sites enumerated : 0
Total # 64-bit Int Data fault sites enumerated : 0
Total # 32-bit IEEE Float Data fault sites enumerated : 0
Total # 64-bit IEEE Float Data fault sites enumerated : 0
Total # Ptr fault sites enumerated : 0
/*********************************End**********************************/
```

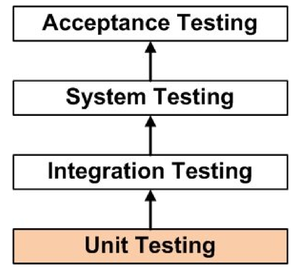
Interceptor Output

Thread Tree Output

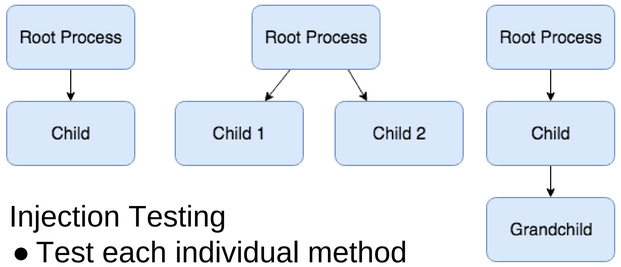
Current Injection Statistics

Trial Automation
- Analysis of programs requires many trial runs (millions or billions, each taking substantial time)
- Impractical to execute each by hand
- Solution: Python wrapper file to automate execution, aggregate output into database files for analysis
- Acts as friendly user interface

Testing
- Test individual units
- Move forward when all tests succeed



Thread Testing
- Start simple, increase complexity
- More children, greater depth



Injection Testing
- Test each individual method
- Test dynamic injection

Safety Concerns
- Protect user data
- Protect user hardware
- Protect other processes

Conclusion
- All original specs satisfied
- Works with sample Big Data applications
- Ready for full scale application

Adviser: Dr. Gabriel de Souza