Preliminary Assessment of indeterminism in the GNU/Linux Kernel

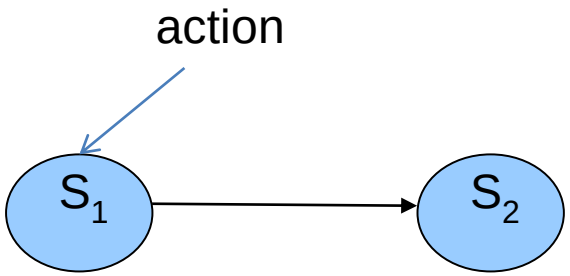
Nicholas Mc Guire
Peter Okech*
Quinggou Zhou

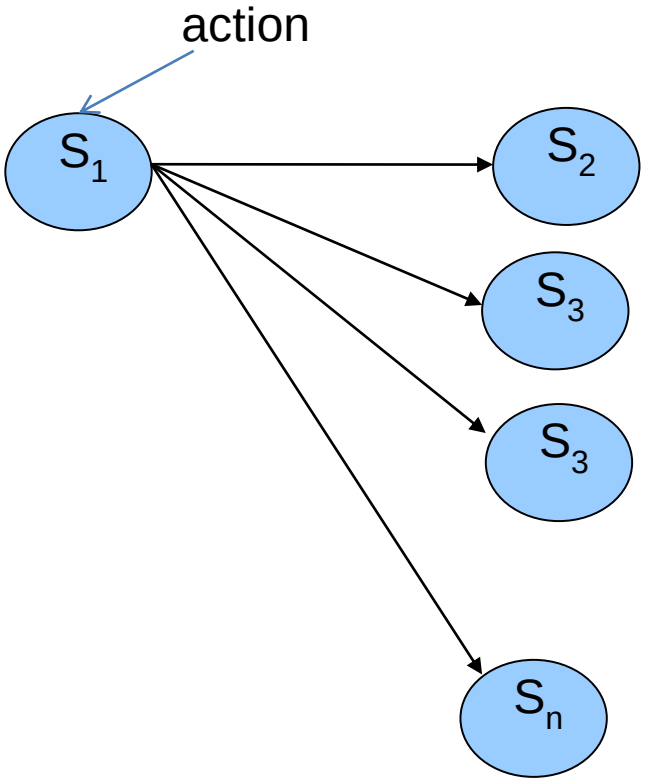
DSLab, Lanzhou University, P.R. Of China
****Faculty of Information Technology, Strathmore University***

Introduction

- There is a common assumption that applications behave the same way each time they are executed
  - i.e. deterministic execution

What do we mean by determinism?



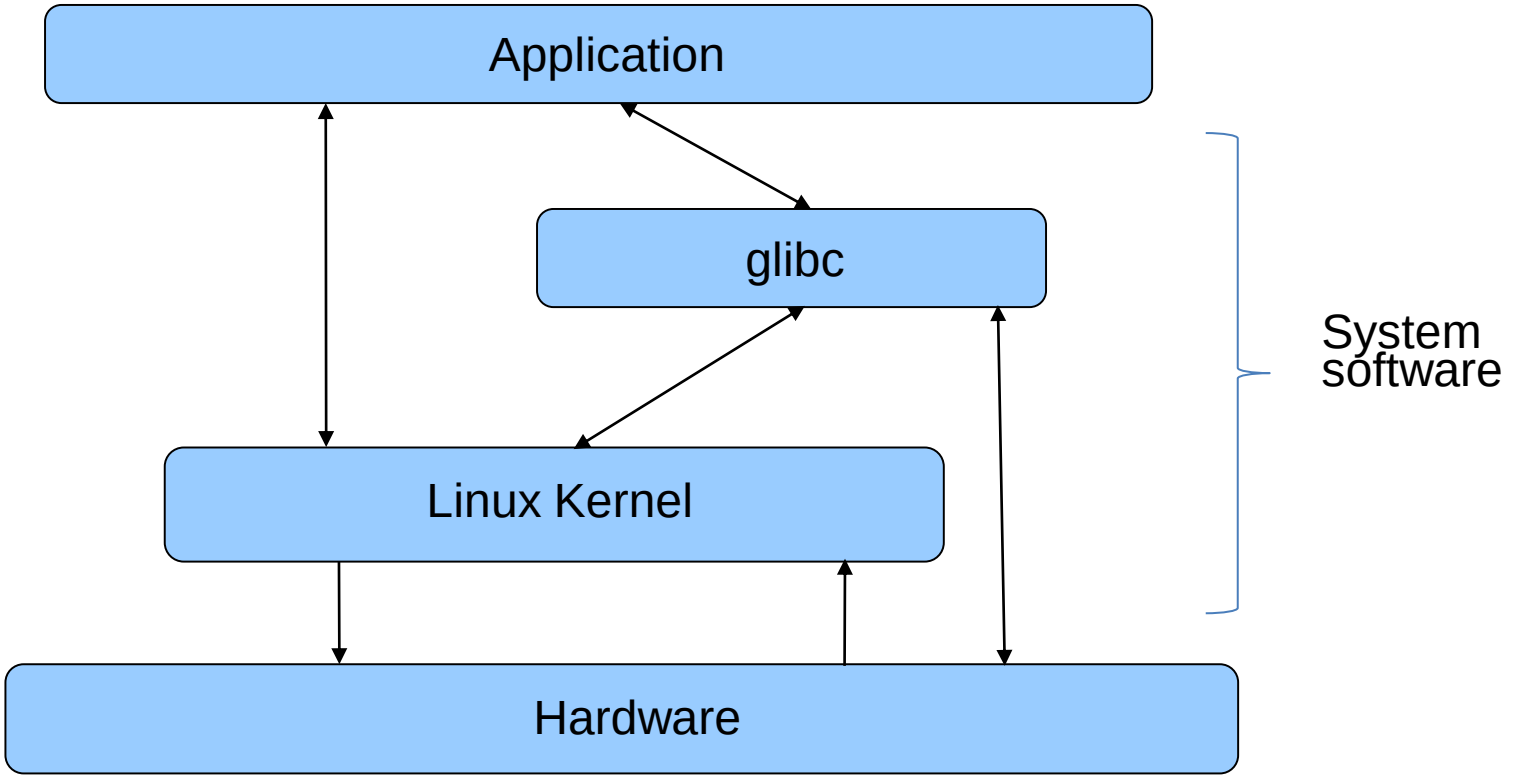


Deterministic system:- there is always a transition from S_1 to S_2 when an action is applied state S_1

Non-deterministic system:- when an action is applied to sate S_1, the system will transition unpredictably to any of the possible states S_2, S_3, ... S_n

System Software Layer

- Most applications interacts with the hardware through a system software layer



Effect of System Software Layer

- The system software layer acts as randomizing agent, introducing temporal non-determinism

Hypothesis

If software execution is deterministic,

it should not be possible to write a random number generator in software that produces output which exhibit truely random properties.

Approach

- To test the hypothesis and quantitatively verify our view that software execution is non-deterministic, we have implemented a trivial random number generator

Testing

- We then measured the quality of randomness of the result by carrying out two tests

1. Comparison of the output of the our RNG with the output of the systems /dev/random
   - Completed

2. Apply standard tests from the TESTU01 test suite (L'Ecuyer & Simard, 2002) to confirm that the resulting sequence of generated from our software are truely independent
   - **Tests not yet complete**

Implementation Platform

- Our implementation platform is a standard desktop system
  - Hardware : Intel Core 2 Duo CPU
  - Operating system: Debian GNU/Linux distribution, running the mainline Linux kernel version 2.6.24

Implementation

- Our code is a simple loop that reads the least significant bit (LSB) of the time stamp counter (TSC). The value returned is shifted into a 32 bit block.

Implementation - 2

RNG code segment

```
__inline__ unsigned long long int hwtime (int shift)
{
unsigned long long int x,
         res = 0;
int i;
int bit = 1;

bit <<= shift;
for (i=0; i<sizeof(res); i++)
{
   ___asm__ __volatile__ ("rdtsc\n\t":"=A" (x));
   res |= (((x & bit) >> shift) << i);
   usleep (RANDOMIZE_INTERVAL);
}
return res;
}
```

Results

Table 1: Comparison of our RNG with the systems /dev/random for 512 bytes of data

	Our RNG	/dev/random
Time to execute	0.052 secs	169.138 secs
Size after compression with bzip2	679 bytes	672 bytes

Discussion

- Both the CPU and our code are deterministic (thier periods are fixed)
  - The TSC is incremented every CPU tick (1/hz)
  - The code sleeps for some time – RANDOMIZE_INTERVAL before reading the TSC
- The ratio of the two periods should ideally produce a pattern, thus a non-random output
- The results we have obtained show otherwise
- This verifies our model that software execution is non-deterministic

Potential Usage

- The random number string would be useful in initializing the operating systems entropy pool
  - Especially useful for systems that do not have sufficient asynchronous event sources

Conclusion

- The results indicates that the quality of random numbers generated by our code and /dev/random are comparable.
- The concept of deriving randomness from the inherent determinism of a complex software system is usable.
- Further assessment will help in understanding the indeterminism in such systems

Further work

- A more thorough assessment needs to be done to check the dependency on
  - Systems loads
  - Scheduling policy
  - Hardware architecture
- We need to complete the standard randomness tests, and analyze the results
- Identify the underlying artifacts (hardware and software) that causes the randomness

Thank You!
Questions?

Results

```
aetsch:~/rand/data# time ../trng -s 0 -l 512 -p 0
-d 1 > trng512
real    0m0.052s
user    0m0.000s
sys     0m0.016s

aetsch:~/rand/data# ls -l trng512
-rw-r--r-- 1 root root 512 2009-06-08 21:09
trng512
aetsch:~/rand/data# bzip2 trng512
aetsch:~/rand/data# ls -l trng512.bz2
-rw-r--r-- 1 root root 672 2009-06-08 21:08
trng512.bz2
```

Results - 2

```
aetsch:~/rand/data# dd if=/dev/random of=random512
bs=1 count=512
512+0 records in
512+0 records out
512 bytes (512 B) copied, 169.138 seconds, 0.0
kB/s

aetsch:~/rand/data# ls -l random512
-rw-r--r-- 1 root root 512 2009-06-08 21:03
random512
aetsch:~/rand/data# bzip2 -9 random512
aetsch:~/rand/data# ls -l random512.bz2
-rw-r--r-- 1 root root 676 2009-06-08 21:03
random512.bz2
```