Presentation of
“An Empirical Study of Operating System Errors”

Lucas Panjer

September 21, 2006

Summary

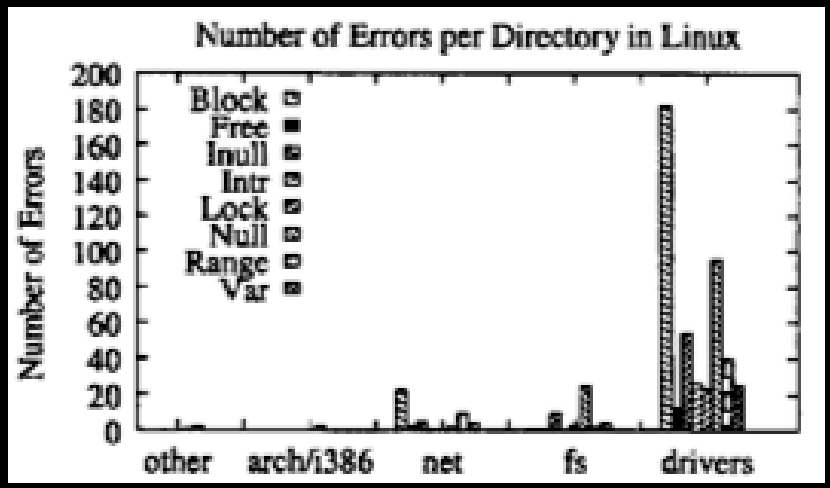
- Study of collected coding “errors” generated by automatic checkers
- Errors are predefined common coding errors
- 12 checkers
- Purposely ignored implementation details of checkers
- Validity of errors is unknown

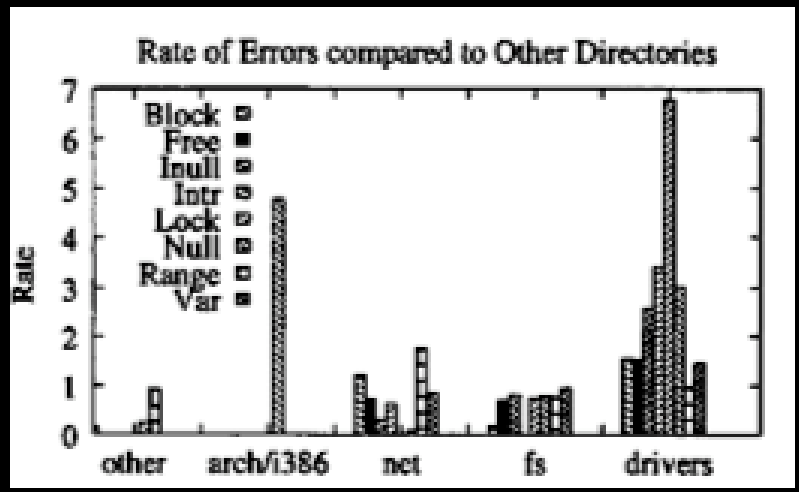
Check	Nbugs	Rule checked
Block	206 + 87	To avoid deadlock, do not call blocking functions with interrupts disabled or a spinlock held.
Null	124 + 267	Check potentially NULL pointers returned from routines.
Var	33 + 69	Do not allocate large stack variables ($> 1K$) on the fixed-size kernel stack.
Inull	69	Do not make inconsistent assumptions about whether a pointer is NULL.
Range	54	Always check bounds of array indices and loop bounds derived from user data.
Lock	26	Release acquired locks; do not double-acquire locks.
Intr	27	Restore disabled interrupts.
Free	17	Do not use freed memory.
Float	10 + 15	Do not use floating point in the kernel.
Real	10 + 1	Do not leak memory by updating pointers with potentially NULL realloc return values.
Param	7	Do not dereference user pointers.
Size	3	Allocate enough memory to hold the type for which you are allocating.

Topics of Analysis

- Where are the bugs?
- How are bugs distributed?
- How long do bugs live?
- How do bugs cluster?

Where are the bugs?

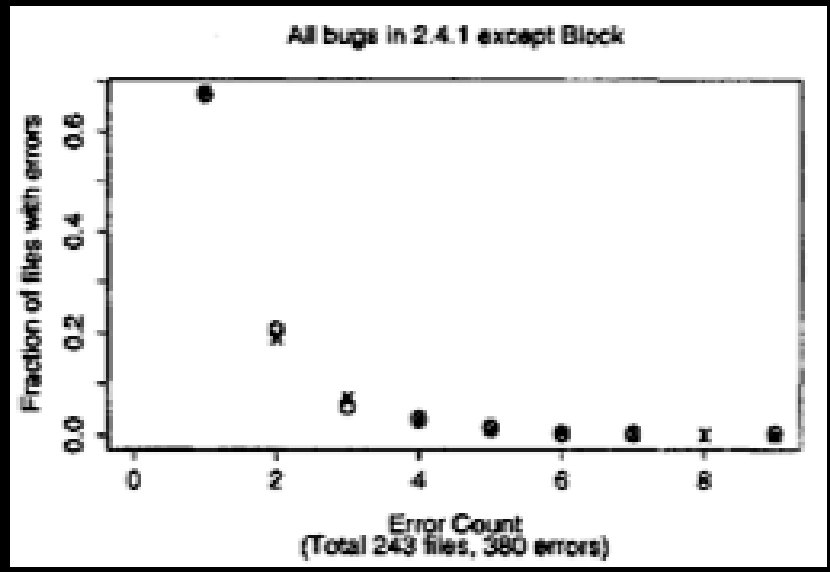




- Majority of code is drivers (~50-70%)
- Most errors are in drivers code
- Detected errors correlate with large functions

How are bugs distributed?

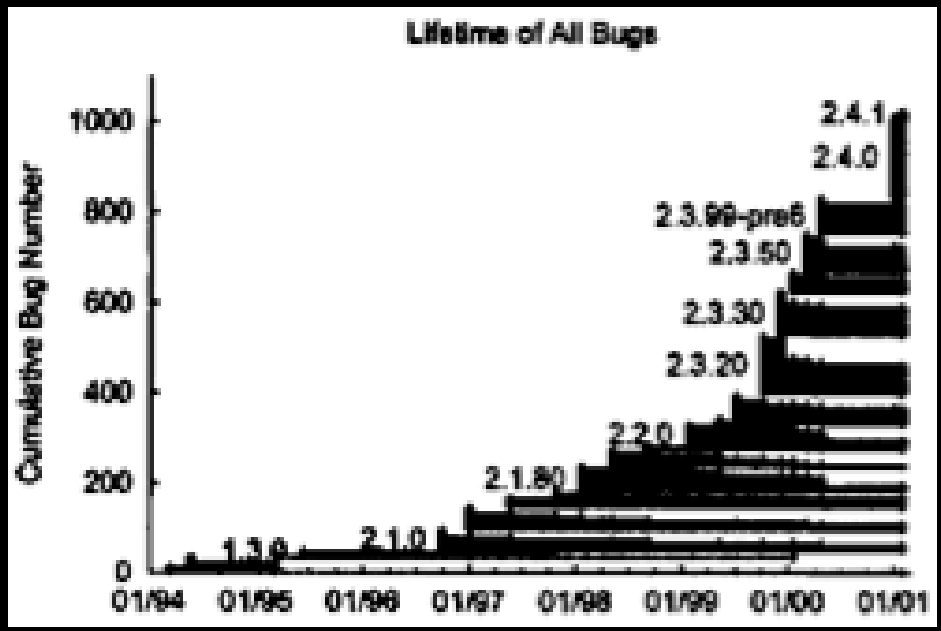
- Long tail distribution
- Log series distribution for all checkers except *Block*
- Yule distribution for *Block* (longer tail/higher clustering)

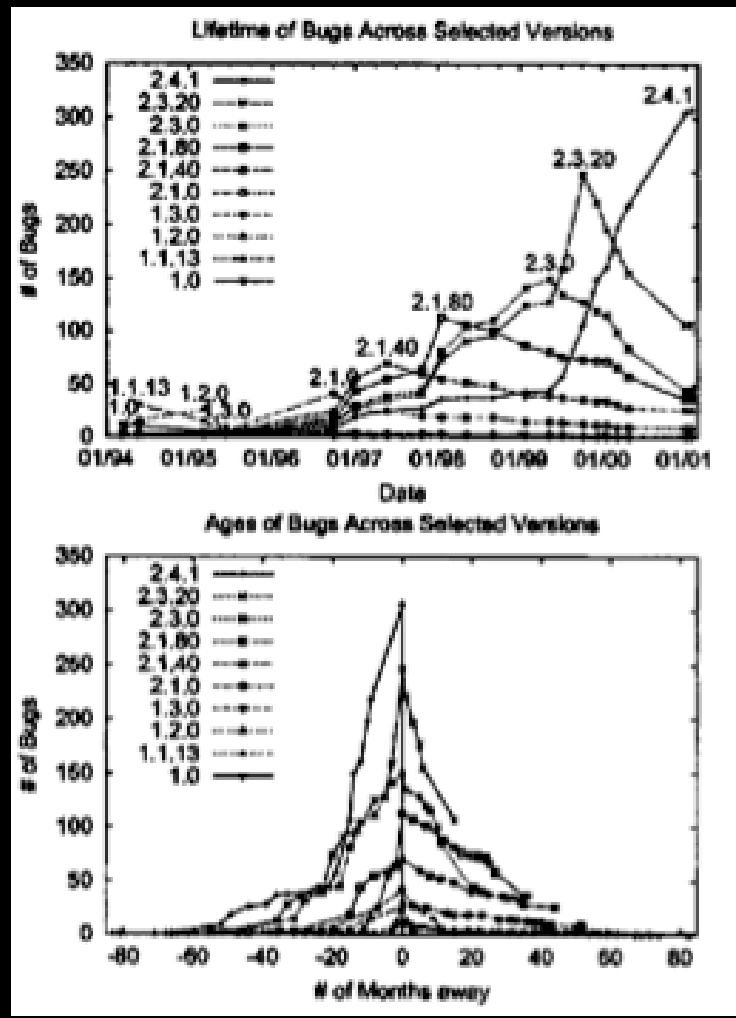


- Uses file as the unit for aggregating error count
- Also tried using a standardized chunking method
- More work necessary, seems like significant clustering could exist

How long do bugs live?

- Analysis of distribution through time





How long do bugs live?

- 40-60% of bugs introduced in current year, remaining are older
- Interesting facts can be gleaned easily
  - How many bugs shared between versions
  - Number of bugs introduced in a version

How long do bugs live?

- Bugs detected as born at version in which they appear, death occurs at first version in which they are no longer present
- Misses bugs that live only between versions (Overestimates)
- Misses bugs that start between versions or end well after versions (Underestimates)

How long do bugs live?

- Kaplan-Meier method used to estimate censored data
- Expectancy calculated at 1.8yrs, median 1.25yrs
- Code does harden
  - Older files have lower error rate
  - Newest quartile is 2x older quartile

How do bugs cluster?

- Metric: Variance to the mean

Arrangement				Clustering
e1	e2	e3	e4	c
x	x	x	x	0
xx	x	x		0.5
xx	xx			1
xxx	x			1.5
xxxx				3

How do bugs cluster?

- Many errors found to be isolated mistakes
- Minimal to no clustering on most checkers
- *Null* and *Inull* show minimal clustering
  - Seems to imply incomplete knowledge of APIs or “cut and paste”
- *Block* shows more clustering
  - Seems to imply that programmers are unaware of this mistake

How do bugs cluster?

- Lots of hypotheses
  - Inexperienced programmers
  - Ignorance to rules or conventions
  - Working code seen as correct code

Comparison with OpenBSD

	Percentage			Bugs		Notes	
Checker	Linux	OpenBSD	Ratio	Linux	OpenBSD	Linux	OpenBSD
Null	1.786%	2.148%	1.203	120	27	6718	1257
Intr	0.465%	0.617%	1.328	27	22	5810	3566
Free	0.297%	0.596%	2.006	14	13	4716	2183
Param	0.183%	1.094%	5.964	9	18	4905	1645

- Higher rate of errors OpenBSD
- Fewer occurrences of errors OpenBSD
- Very preliminary results

Contributions

- Many errors/bugs fixed in Linux
- Unusual to analyze only the bug detection tool output, many studies focus on the checkers
- Surprised by lack of clustering results
- Lots of questions and conjecture

Issues

- If a tree falls in the forest… ?
- If a bug is squished before it bites… ?
- Are “errors” bugs… ?

- Only covers low level operations
- Interference, control, validation?
- Selection of code base?
- Accessibility?