Automatyczne testowanie jądra Linuksa

Krzysztof Skrzypczyński

Marcin Mieteń

Automatyzacja testów

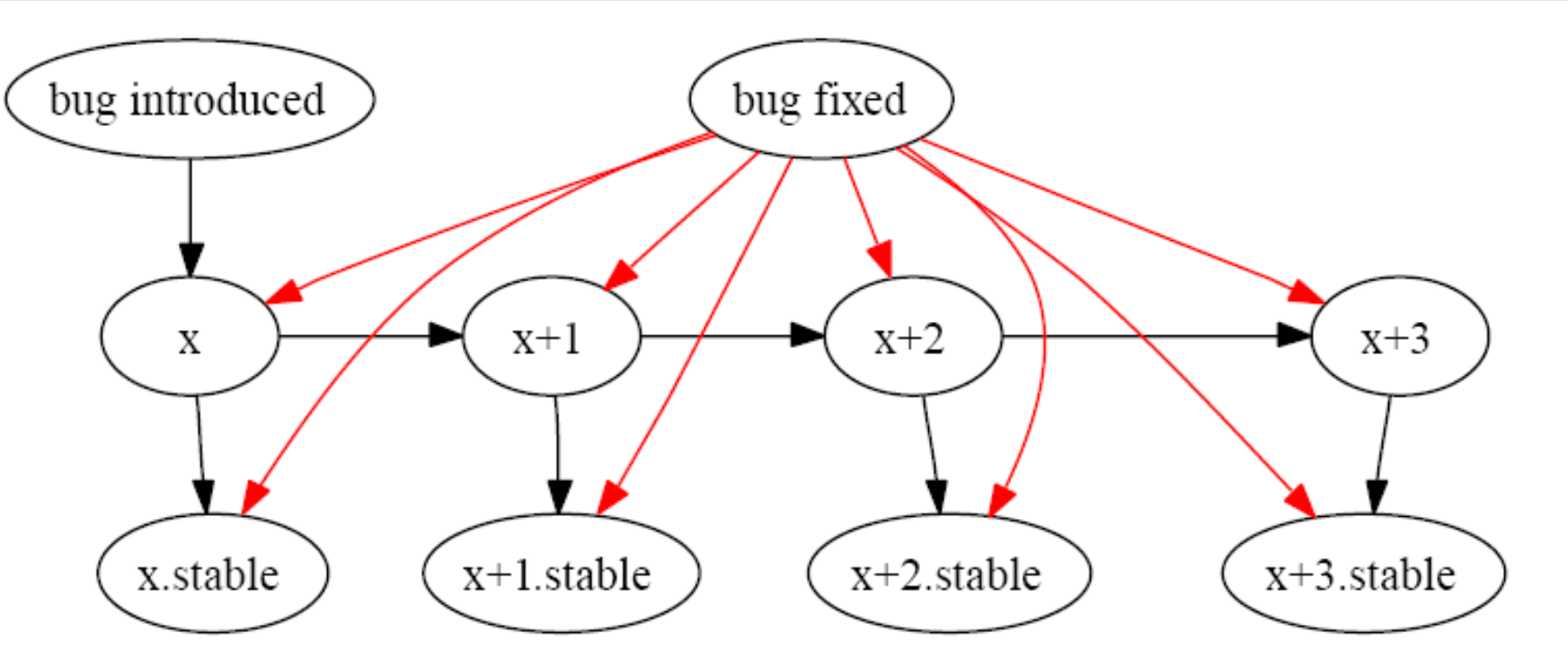
- W kontekście testowania jądra linuxa tworzy się skrypty i programy sprawdzające działanie odpowiednich części systemu pod kątem poprawność, wydajności i stabilności.
- Takie testy też powinny być testowane.

Zalety automatyzacji

- Możliwość ponawiania testów.
- Testy zawsze wykonywane są w dokładnie ten sam sposób.
- Zmniejsza ryzyko błędu ludzkiego przy testowaniu.
- Możliwość uruchamiania wielu testów jednocześnie
- Szybkość przeprowadzania testu

Wykryć jak najszybciej

- Czym szybciej wykryjemy błąd tym mniejsze koszty naprawy w przyszłości.



Charakterysytka testów Jądra

- Jądro linuxa jest dystrybuowane jako wysoko konfigurowalny kod. Więc musi być testowane przy wielu różnych ustawiniach tak aby wykryć wszystkie kombinacje powodujące błędy.
  300 tys lini kodu w zainstalowanym jądrze.
  W całym jądrze jest 3 miliony l. kodu
- Wsparcie dla wielu architektur, różnych procesorów a nawet dla komputerów dużej mocy (mainframe processors)
- Wsparcie dla dużej liczby różnych konfiguracji sprzętowych (hardware)

Charakterystyka testów jądra

- Jądro jest interfejsem miedzy programem użytkownika a sprzętem.
- Błąd w systemie ma dużo gorsze skutki niż błąd w programie użytkownika.
- Nie ma możliwoście sprawdzani wycieków pamięci.
- Testy mogą zniszczyć sprzęt
- Nie można uruchamiać na maszynie roboczej
- Jądro testowane jest, także w sytuacjach nietypowach takich jak brak zasobów systemowych czy ekstremalnie długi czas pracy.

Testy Poprawnościowe - Metodologia

- Testy funkcjonalne.
  - Sprawdzają zgodność ze specyfikacją.
  - Jakość tych testów zależy głównie od sposobu wyboru danych testowych
  - Losowe dane testu nie gwarantują poprawnego działania programu dla wszystkich danych
  - Testy powinny być powtarzane wielokrotnie – im więcej tym lepiej.
  - Testy powinny być różnorodne
  - W celu zapewnienia lepszej jakości testów przeprowadza się je między innymi dla granicznych wartości danych.

Nawet sprawdzenie wszystkich możliwych danych wejściowych nie gwarantuje poprawności – WSPÓŁBIEŻNOŚĆ !

Testy Poprawnościowe

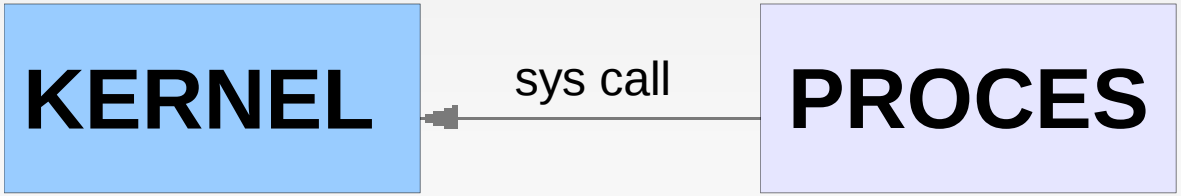
- Testy strukturalne.
  badanie poprawności kodu tak, żeby:
  - każda instrukcja była wykonana przynajmniej raz - *(pokrycie wszystkich instrukcji).*
  - każdy warunek logiczny był przynajmniej raz spełniony i niespełniony- *(pokrycie instrukcji warunkowych).*

Testy poprawnościowe - co testować?

- **Wywołanie systemowe** (ang. *system call*)
  interfejs między wykonywanym programem a jądrem systemu operacyjnego.
  open, read, write, close, wait, exec, fork, exit, kill ...

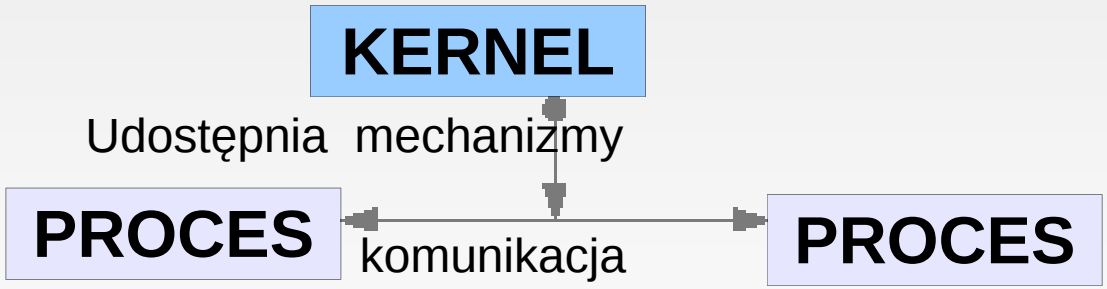
  Linux posiada około 320 różnych wywołań systemowych

  narzędzie: strace
  Śledzi wywołania systemowe oraz sygnały w procesie. Może też zliczać i mierzyć czas poszczególnych wywołań



Testy poprawnościowe - co testować?

- IPC(ang. *Inter-Process Communication* — **IPC**)
  Komunikacja międzyprocesowa – umowna nazwa zbioru sposobów komunikacji pomiędzy procesami systemu operacyjnego.
  System operacyjny udostępnia mechnizmy które programista wykorzystuje do komunikacji międzyprocesorowej
  pliki i blokady ,sygnały, semafory, łącza nienazwane, kolejki komunikatów, pamięć dzielona ...

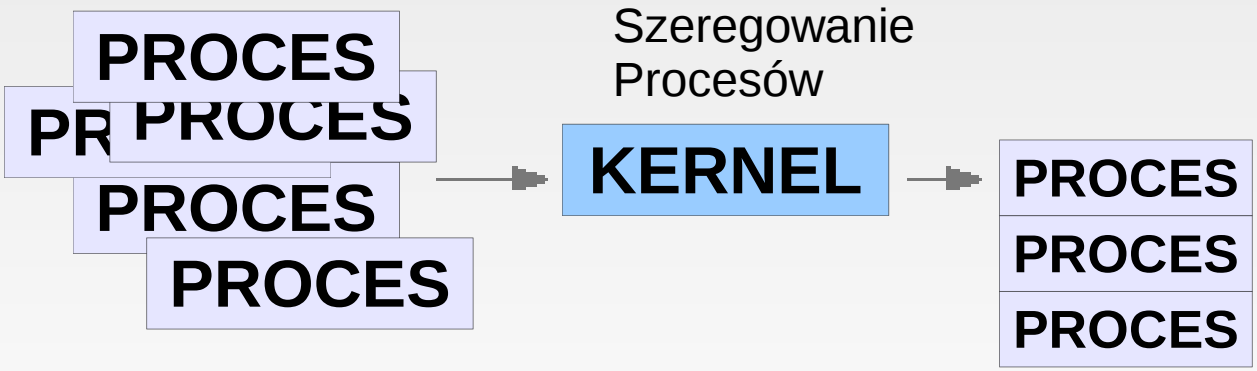


- Z uwagi na implementację w jądrze cała komunikacja wykonywana jest w kontekście jądra

Testy poprawnościowe - co testować?

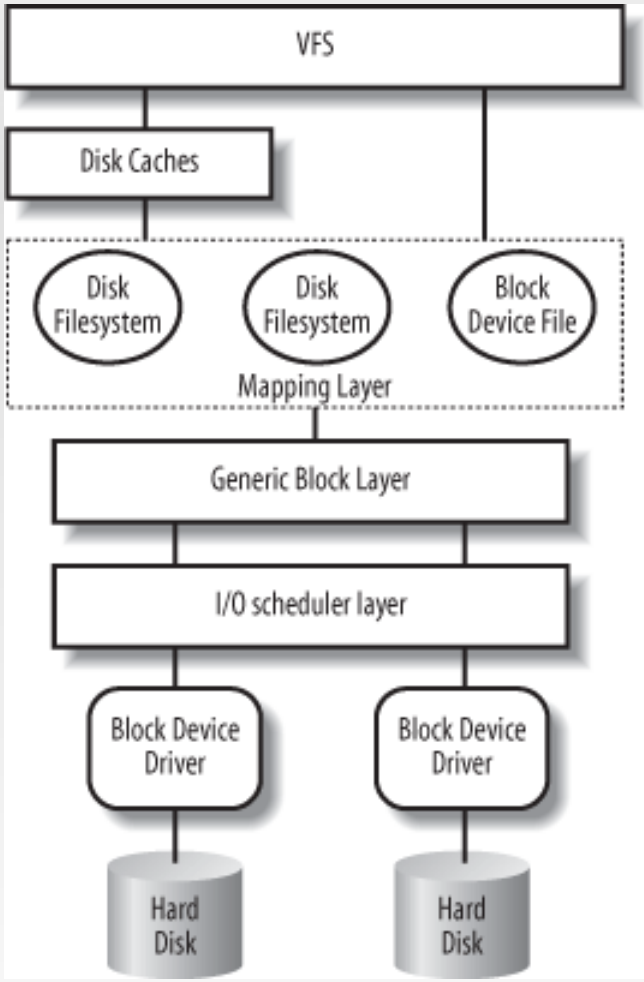
- Scheduler

  Testowanie zachowania schedulera przy dużym obciążeniu (zagłodzenie i inne problemy programowania współbieżnego)



Testy poprawnościowe - co testować?

- Wszystkie struktury związane z:
  - disk I/O
  - memory management
  - file systems
  - real time features
  - POSIX semantics
  - networking
  - security



Testy Poprawnościowe:

Stress test

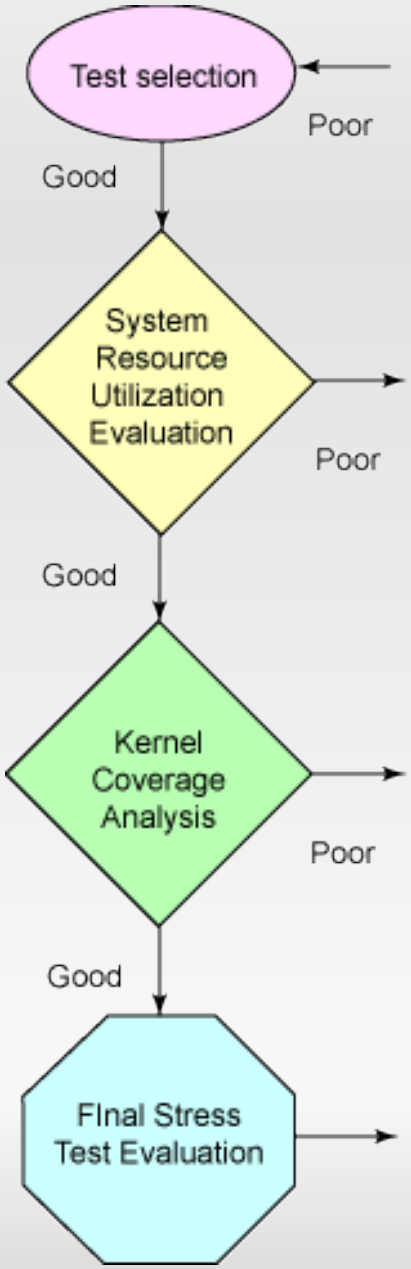
- Wykonywany jest raczej przy wysokim obciążeniu systemu, a nie przy standardowym wykorzystaniu zasobów systemu.
- Kładzie nacisk na testowanie pod kątem :
  - Stabilności
  - Dostepności
  - Wyłapywania błędów
- Cel:Zachowanie systemu w wyjątkowych sytuacjach (brak zasobów, duża konkurencja o zasoby)

Testy Poprawnościowe:

Stress test

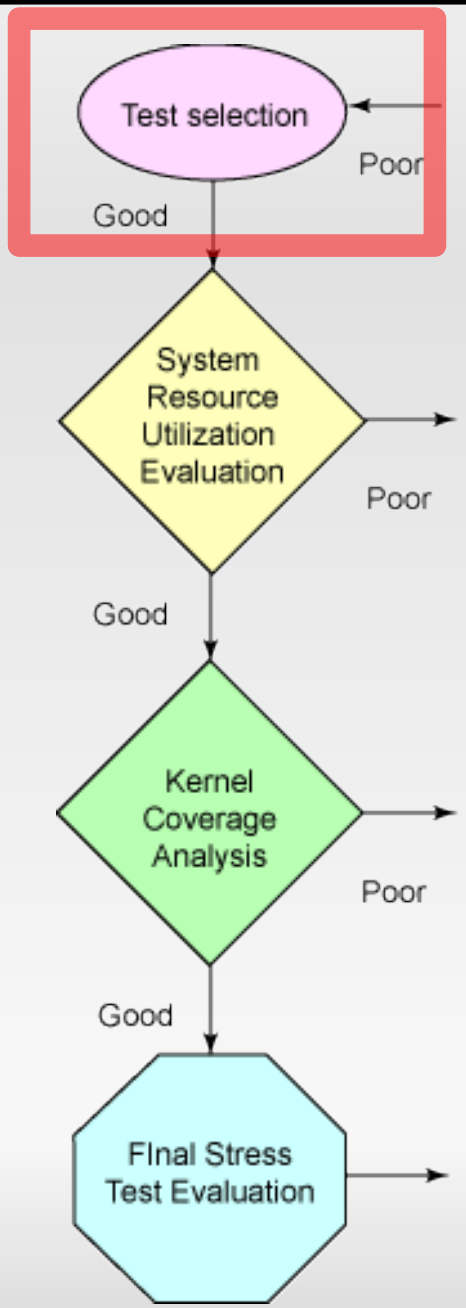
- Proces przeprowadzania stress testów.
  - Wybór testów funkcjonalnych
  - Ocena wykorzystania zasobów
  - Analiza pokrycia kodu
  - Stress testing

Brak akceptacji któregoś etapu powoduje powrót do punktu pierwszego



Testy Poprawnościowe

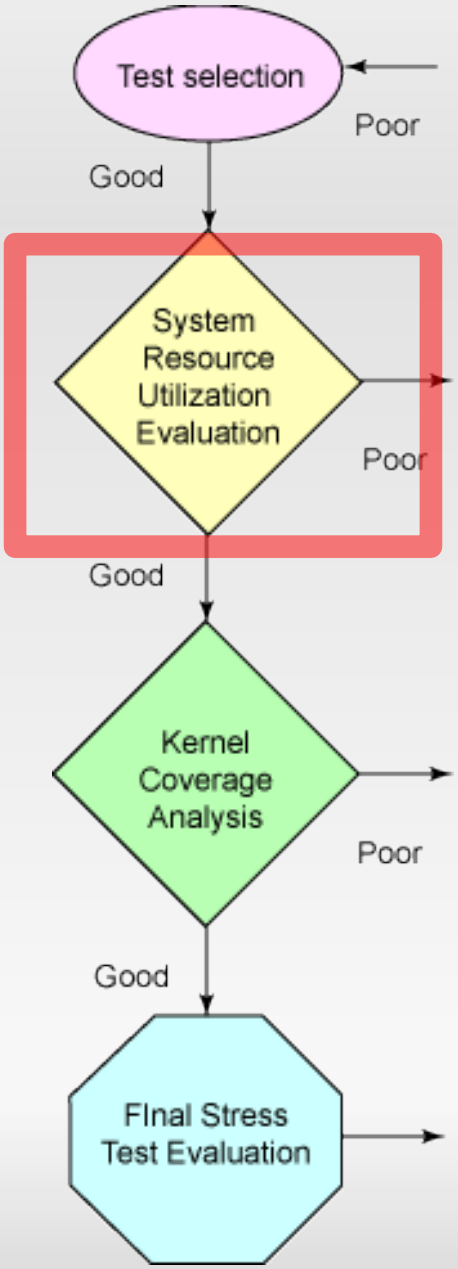
- Wybór testów
  - Testy powinny pozwolić na wysokie wykorzystanie głównych przestrzeni jądra takich jak : CPU(s), pamięc operacyjna, I/O
  - Testy muszą pokryć dostatecznie dużą częśc kodu jądra.
  - Wybieramy testy zautomatyzowane pozwalające na szybkie i wielokrotne wykonywanie.



Testy Poprawnościowe:

Stress test

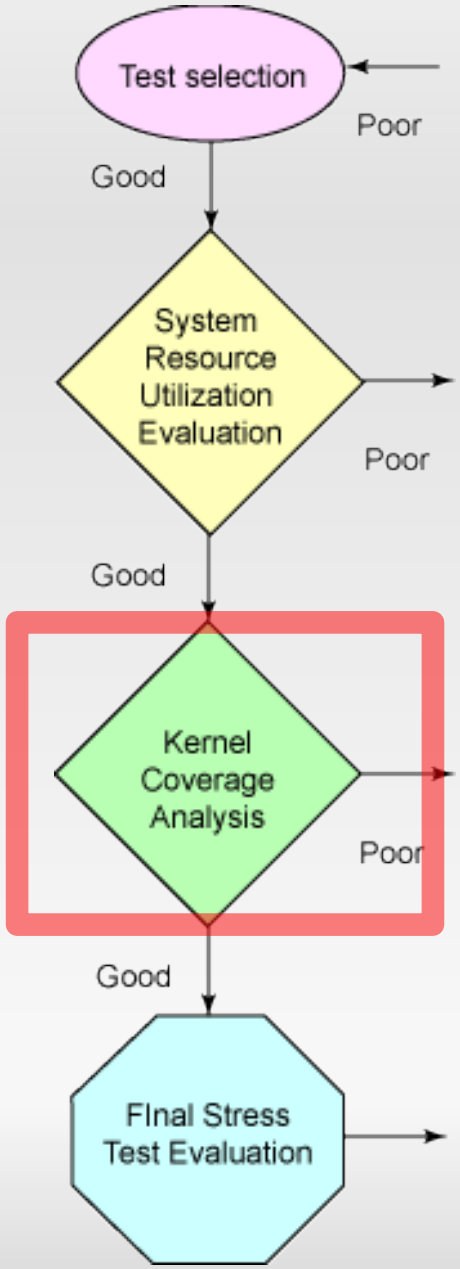
- Ocena wykorzystania zasobów
  - CPU
  - Memory
  - I/O
- Metodą prób i błędów szukamy odpowiedniej kombinacji testów.
- Narzędzia do monitorowania wykorzystania zasobów:
  - Top - monitorowanie w czasie rzeczywistym
  - Sar – zbiera statystyki co określony czas
- Po wybraniu ostatecznej kombinacji testów wykonujemy je jednocześnie w celu pełnej analizy wykorzystania zasobów.
  (krótsze testy powinny być wykonywane wielokrotnie tak aby trwały tyle co najdłuższy test)



Testy Poprawnościowe:

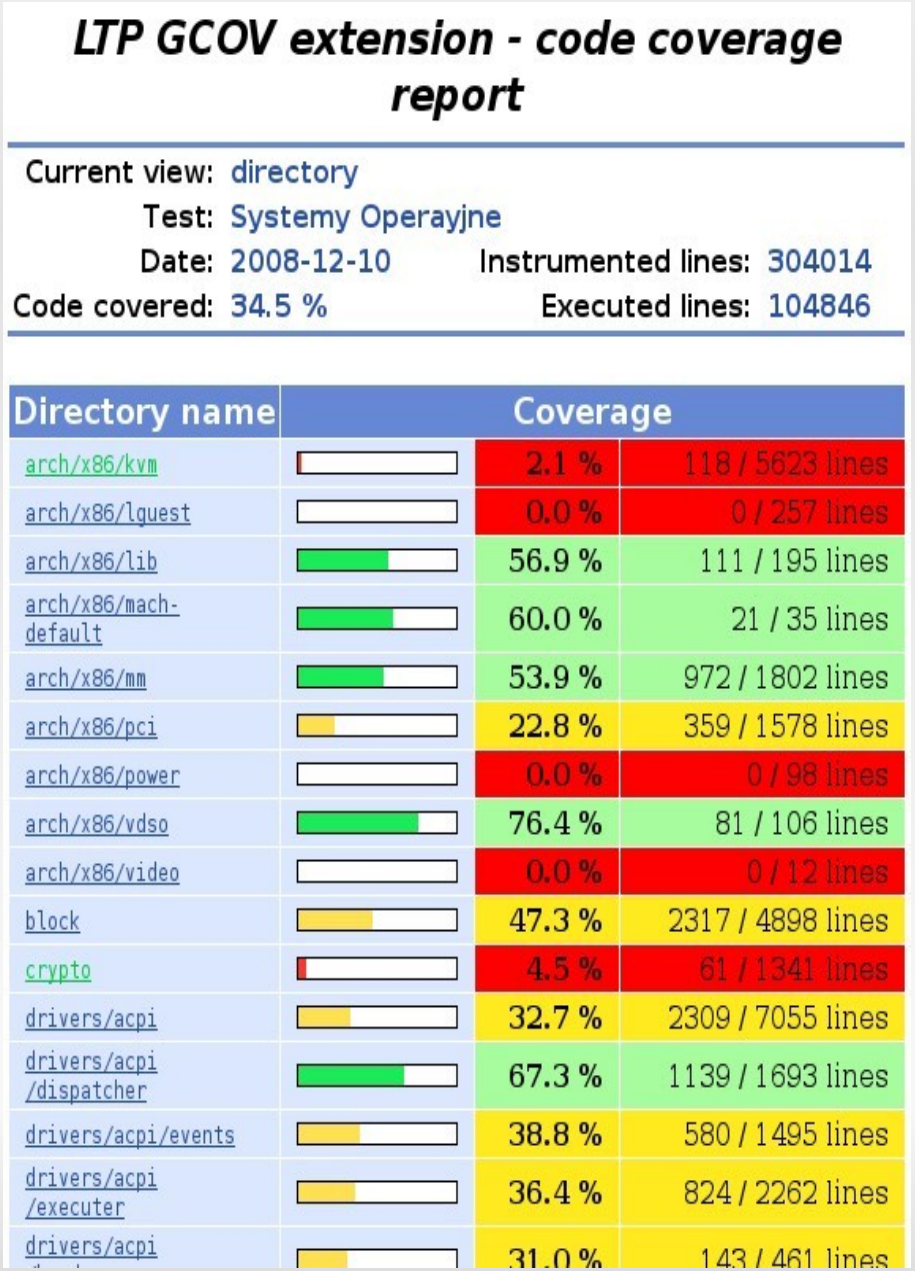
Stress test

- Analiza pokrycia kodu
- Zlicza ile razy została wykonana każda linia kodu
- Możemy sprawdzić jaka część kodu jest faktycznie wykonywana
- Statystyki każdego pliku w /proc/gcov/
- Instaluje się jako patch do jądra



LCOV

- Skrypt przedstawiający dane z GCOV w postaci plików html
- Wyraźnie widać które części jądra w jakim stopniu zostały pokryte
- Statystyki z całego jądra generują się w kilka minut



LTP GCOV extension - code coverage report

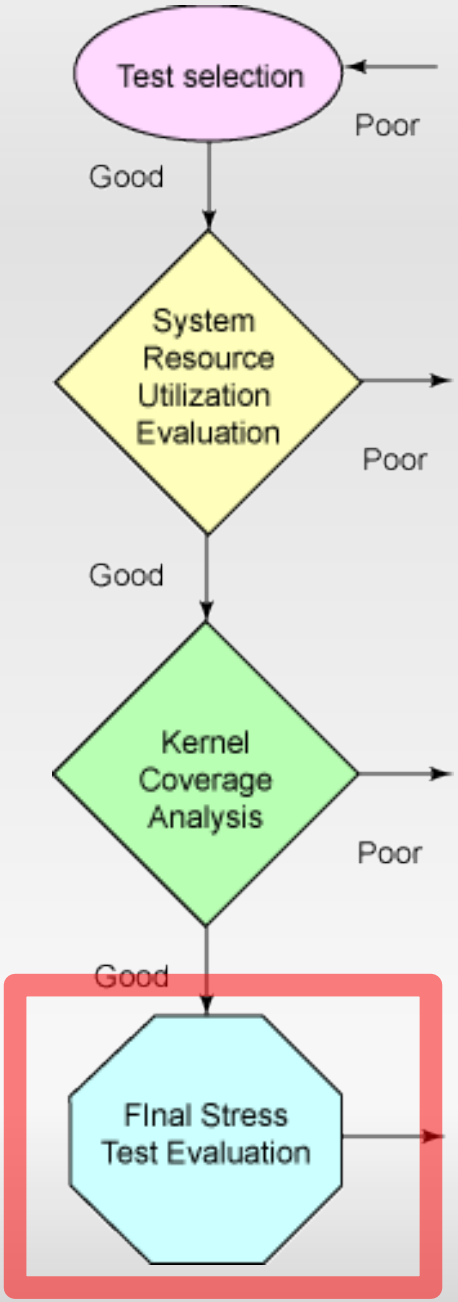
Current view: directory
Test: Systemy Operayjne
Date: 2008-12-10 Instrumented lines: 304014
Code covered: 34.5 % Executed lines: 104846

Directory name	Coverage		
arch/x86/kvm		2.1 %	118 / 5623 lines
arch/x86/lguest		0.0 %	0 / 257 lines
arch/x86/lib		56.9 %	111 / 195 lines
arch/x86/mach-default		60.0 %	21 / 35 lines
arch/x86/mm		53.9 %	972 / 1802 lines
arch/x86/pci		22.8 %	359 / 1578 lines
arch/x86/power		0.0 %	0 / 98 lines
arch/x86/vdso		76.4 %	81 / 106 lines
arch/x86/video		0.0 %	0 / 12 lines
block		47.3 %	2317 / 4898 lines
crypto		4.5 %	61 / 1341 lines
drivers/acpi		32.7 %	2309 / 7055 lines
drivers/acpi/dispatcher		67.3 %	1139 / 1693 lines
drivers/acpi/events		38.8 %	580 / 1495 lines
drivers/acpi/executer		36.4 %	824 / 2262 lines
drivers/acpi		31.0 %	143 / 461 lines

Testy Poprawnościowe:

Stress test

- Finalne uruchomienie stress testu
  - Uruchamiamy przez dłuższy okres czasu (min. 24 h).
  - Test pozwala to na wykrycie błędów nie pojawiających się przy testach krótkich.
  - Zbieramy statystyki za pomocą narzędzia sar. Dane posłużą do porównywania z przyszłymi testami jądra.



Sar

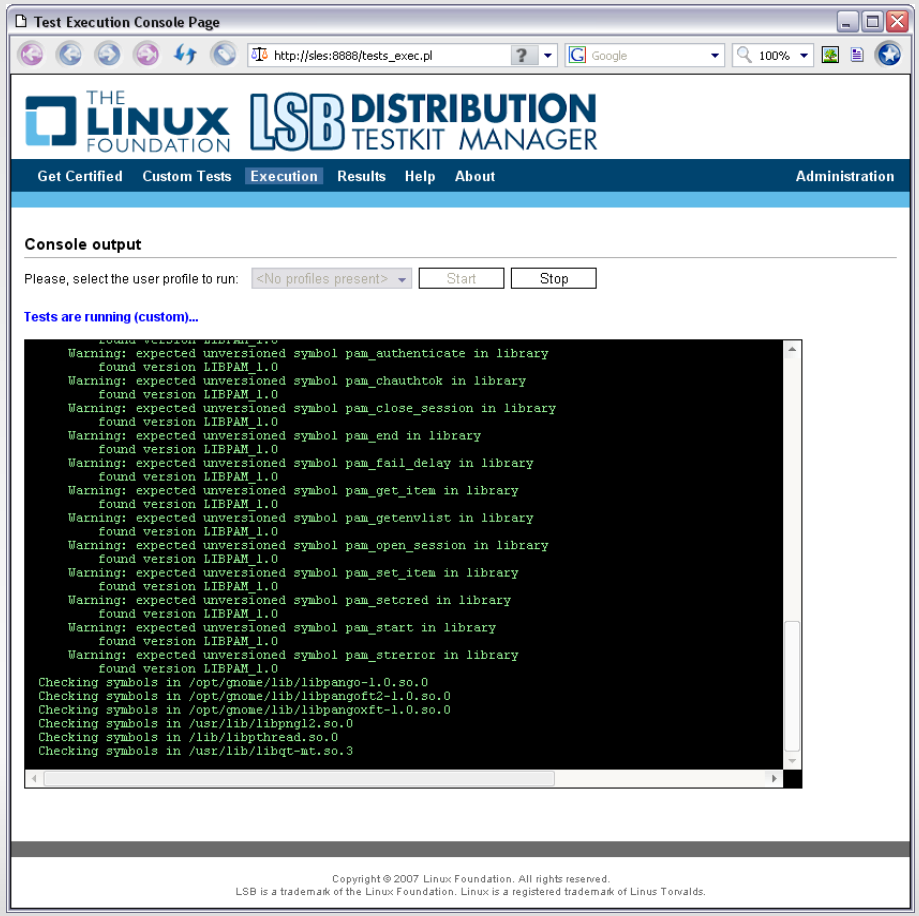
- Pozwala na mierzenie:
  - I/O Transfer (opcja -b)
    - **tps, rtps, wtps** - ilość żądań I/O do urządzenia/s
    - **bwrtn/s, bread/s** - ilość bloków(512B) / s
  - Paging statistics (opcja -B)
    - **pgpgin/s, pgpgout/s** (ilość kb które system pobiera/zapisuje z/do dysku)/ s
    - **fault/s** – ilość błędów braku strony/s (ale nie wszystkie muszą generowć I/O)
    - **majflt/s** – ilość błędów braku strony/s (każdy błąd generuj I/O)
  - Ilość Procesów utworzonych /s (opcja -c)
  - Obciążenie procesorów (opca -P)
  - Zużycie pamięci opracyjnej(opcja -r)
    - Kbmemfree, kbmemused – wolna/używana przestrzeń w kb

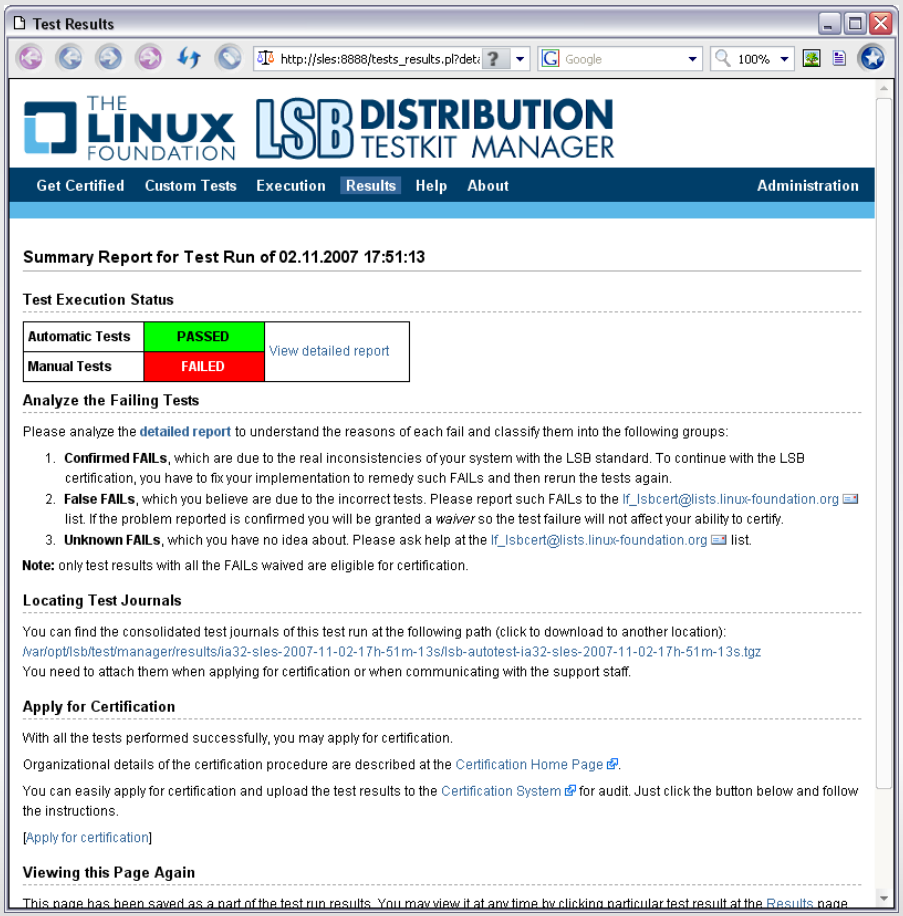
Statystyki Błędów

- W komercyjnym oprogramowaniu występuje od jednego do siedmiu błędów na 1000 linii kodu!!!
- W jądrze 2.6 odkryto 0,127 błędu na 1000 linii. Jądro to składa się z 3 milionów 639 tysięcy 322 linii.

Linux Standard Base (LSB)

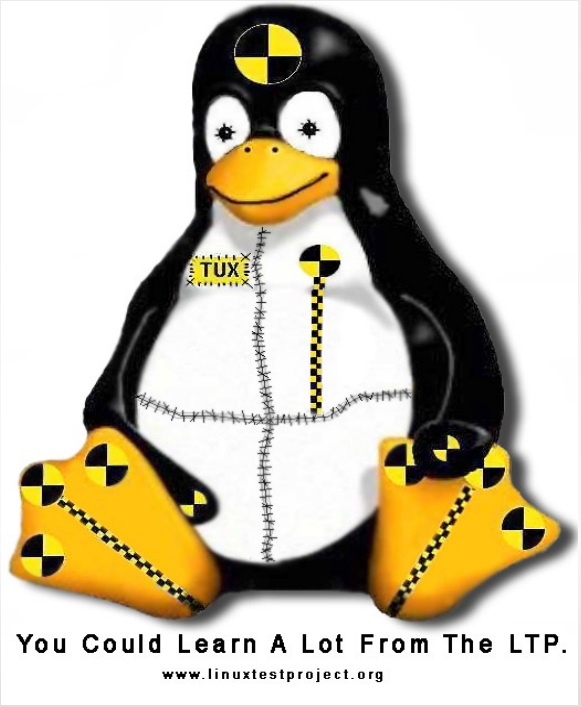
- Zestaw testów pod kątem zgodności z standaradami.
- Zwiększenie przenośności aplikacji





Linux Test Project

- Zestaw ponad 3000 testów dla Linux OS
- Wspierany przez IBM i wiele innych firm.
- LTP udostępnia testy Funkcjonalne, Systemowe i Stress testy.



sgi IBM SDL BULL WIPRO Applying Thought neumont UNIVERSITY

Pokrycie kodu linuxa przez ltp

- Rozwój ltp nie jest prosty gdyż rozmiar systemu ciągle rośnie.
- Mimo to zespół deweloperów nadal zwiększa procentowe pokrycie kodu przez testy z ltp

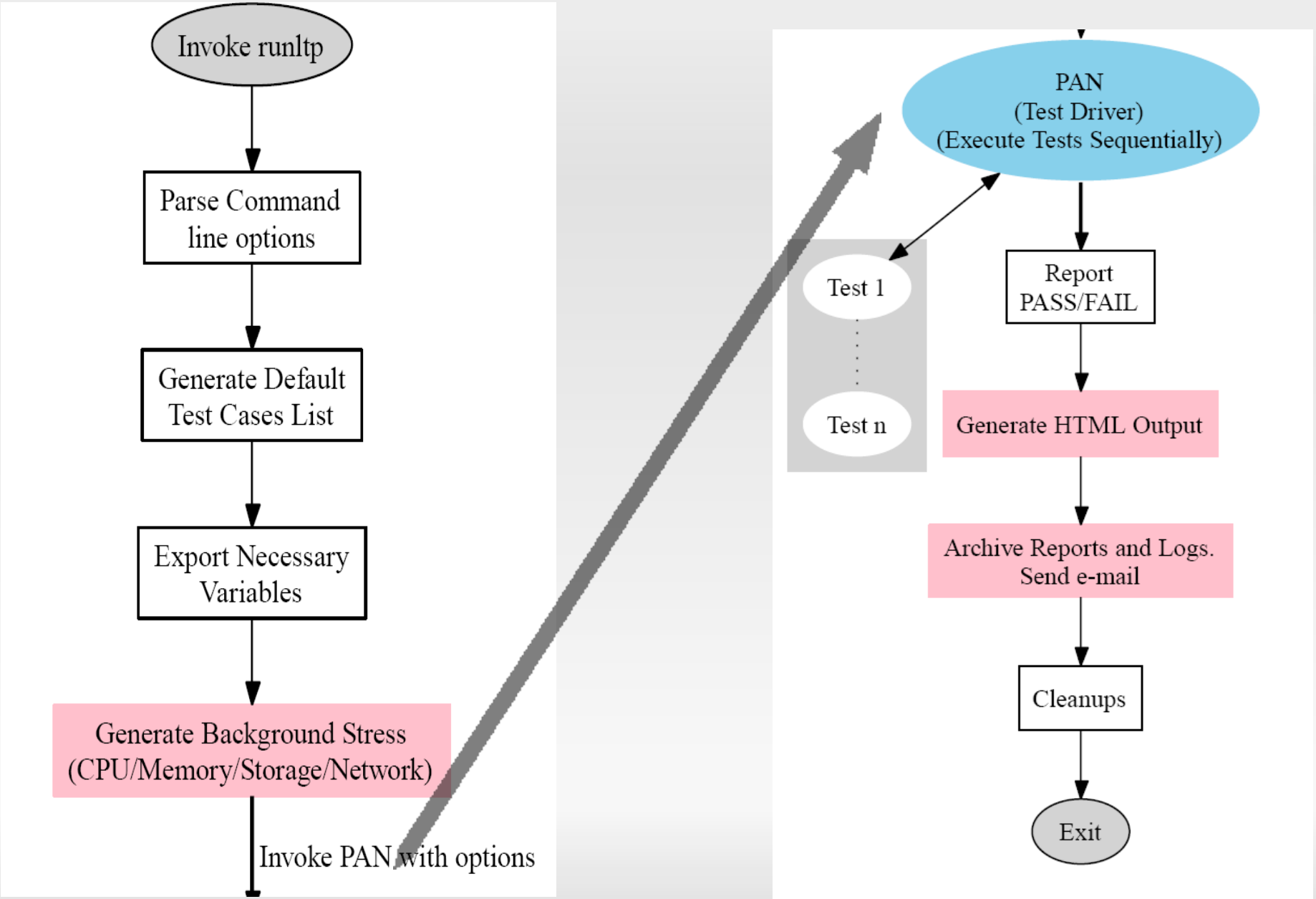
2006

Directory	Coverage	
fs	49.8%	10135/20367 lines
include/asm	49.4%	595/1204 lines
include/linux	58.7%	2239/3812 lines
include/net	56.2%	990/1762 lines
ipc	52.8%	1442/2729 lines
kernel	38.2%	9880/25837 lines
lib	42.2%	2105/4992 lines
mm	51.5%	6899/13396 lines
net	65.4%	630/964 lines
security	51.9%	666/1283 lines

2008

Directory	Coverage	
fs	52.9%	10778/20367 lines
include/asm	50.9%	613/1204 lines
include/linux	60.0%	2283/3812 lines
include/net	57.6%	1015/1762 lines
ipc	56.4%	1539/2729 lines
kernel	39.1%	10097/25837 lines
lib	43.2%	2159/4992 lines
mm	52.7%	7066/13396 lines
net	65.7%	633/964 lines
security	51.9%	666/1283 lines

Wykonywanie automatycznych testów w ltp



Testy wydajnościowe

- Popularnie nazywane benchmarkami
- Mierzą wydajność systemu, w szczególności elementy odpowiadające za dostęp do zasobów:
- CPU
- Pamięć
- I/O
- sieć

Benchmarki – podział

- Dzielą się na
  - Syntetyczne
  - Aplikacyjne

Benchmarki syntetyczne

- Obejmują jeden specyficzny fragment funkcjonalności systemu
- Wyniki są dość dokładne
- Wiemy dokładnie co ma wpływ na wynik tesu
- Nie wiemy jak dokładnie dany fragment wpływa na całą funkcjonalność
- Przykłady: szybkość przełączania kontekstu, przepustowość sieci, opóźnienia w dostępie do pamięci

Benchmarki aplikacyjne

- Obejmują działanie całej aplikacji
- Pozwalają na przetestowanie dużej części funkcjonalności jądra
- Symulują oczekiwane użycie zasobów systemowych
- Nie pozwalają na szczegółową analizę poszczególnych częsci jądra

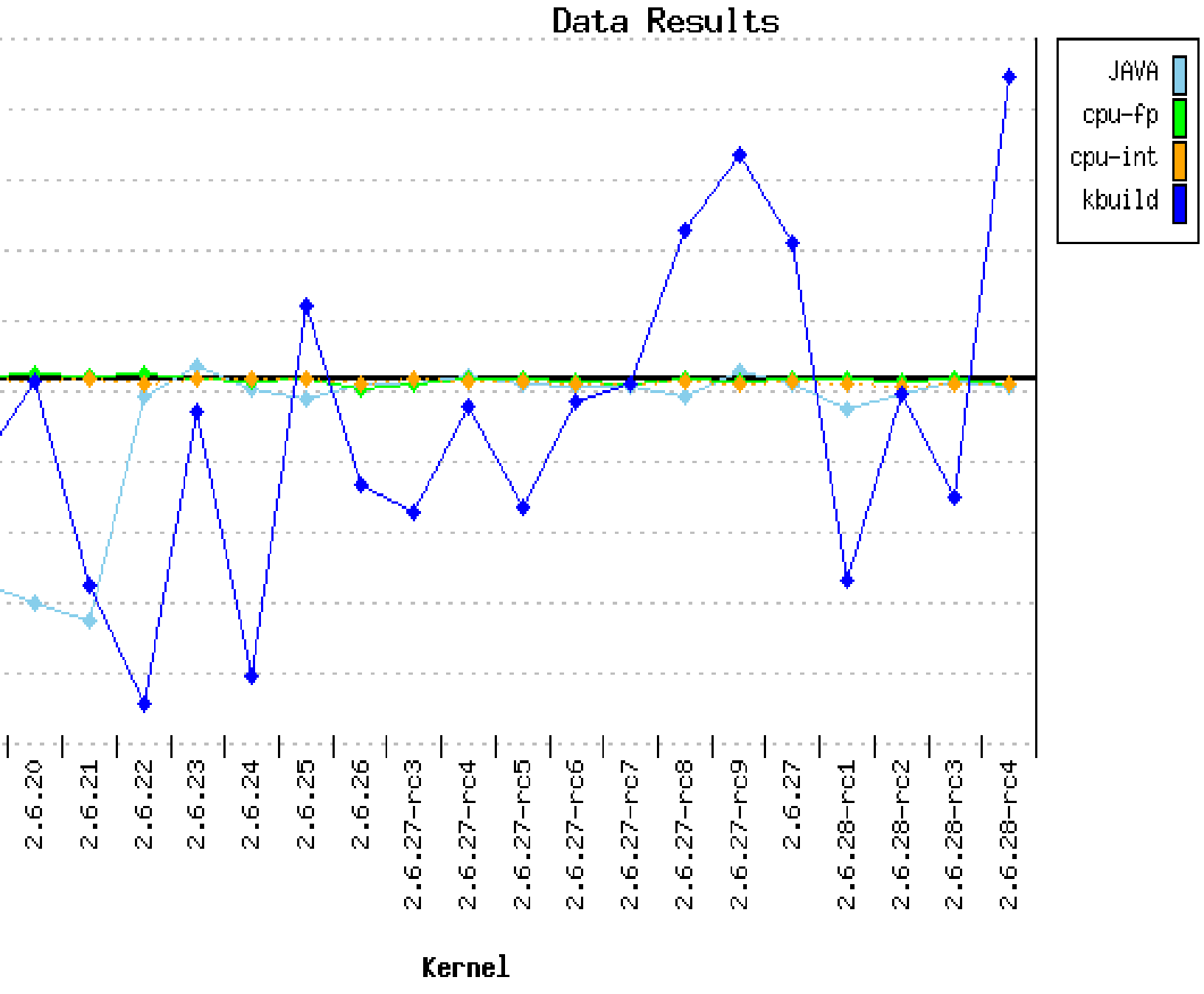
Metody pomiaru

- Bardzo duży wpływ na wyniki benchmarków, zwłaszcza syntetycznych, ma użyty sprzęt
- Porównujemy tylko testy wykonywane na tej samej maszynie
- Wyniki zawsze porównujemy ze stabilną wersją jądra.
- Wnioski wyciągamy dopiero po dostatecznie dużej liczbie testów

W jakich jednostkach mierzymy?

- Wydajność można mierzyć w jednostkach czasu – przydaje się polecenie time
- Przykłady: przełączanie kontekstu, czas dostępu do urządzenia I/O, szybkość wywołania systemowego
- Można również mierzyć przepustowość w B/s (KB/s, MB/s)
- Przykłady: Przepustowość sieci, czytanie/pisanie do pamięci

Zmiany wydajności w kolejnych jadrach



Bootchart

- Mierzy czas uruchomienia systemu
- Monitoruje wszystkie wszystkie procesy wykonujące się w trakcie uruchamiania
- Łatwo zauważyć błędy w skryptach uruchomieniowych

Bootchart

- Nakładka graficzna