First Failure Data Capture (FFDC) for Linux
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the products or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
The FFDC Concept Proposal
Action Plan
Motivation

- **A symptom was detected**
  - Which problem lead to the symptom?
  - In which component did the problem occur?
  - What was the root cause for the problem?
  - Which sequence of operations triggered to the problem?

- **Current approach**
  - Live debugging
  - Increase debug levels
  - Do additional tests with the system
  - Reproduce problem

- **What if this is not possible? (enterprise server, embedded...)**
  - *Automatically record* the required information
  - Always *collect enough information* to reconstruct the history of the error
  - FFDC is a concept to achieve this
What is FFDC? 65500?

- FFDC = First Failure Data Capture

- In case of a problem
  - The problem is detected (automatically)
  - All data needed to analyze the problem is available
  - Data gets collected immediately after problem symptom detection

- Data for the first occurrence of a problem must be preserved

- FFDC data collection executes in rare cases

- Working FFDC means, no need to ...
  - replay a problem situation
  - configure any FFDC setting

- FFDC must be available for all relevant components of your product
  - Relevant kernel components
  - Relevant user space processes
Examples for available FFDC

- **Android:**
  - Crash reports
  - Stack backtrace

- **KDE bug reports**

- **Windows blue screen:**
  - IBM bluescreen capturing

- **Linux manual FFDC:**
  - Redhat: sosreport

- **System z firmware:**
  - IQYYLOG and SE
  - Call home
Constraints

- FFDC is only possible by adding overhead
- FFDC overhead must not cost too many resources
  - limited time to recovery
  - limited CPU usage
  - limited memory usage
  - limited disk storage usage
  - limited network bandwidth
- 2% overhead for all resources might be acceptable?
What do we need for FFDC?

- **Perfect:** (?)
  - Get *all* data
    - Record every *state change*
    - Dump *complete state*
  - Detect *all possible errors*

- **Doable:**
  - Get *relevant data*
    - History: *Trace* component entry/exit points with relevant parameters
    - State: *Dump* component control structures
    - Partitioning/Relationship: Collect only data for *affected components*
  - Detect *relevant errors*
    - Define *error classes & actions*
    - Define what has to be collected for which error class

- **Advantages of doable approach:**
  - Reduced runtime overhead, downtime & disruptiveness
  - Less data to analyze
What we have?
Types of debugging data

- **Logs:**
  Short messages written by a running system to *non-volatile storage* cover whole history of the system (or at least a *long period*). Includes events known to have *long term relevance* (e.g. configuration changes). Log messages are targeted primarily at *system administrators*.

- **Traces:**
  A trace provides a means to create a component-local sequence of timestamped short entries related to events that may be *relevant for debugging purposes*. A trace often is *not persistent* (wrap-around buffers). *Single trace entries* may have *no relevance*. Traces can have a *high frequency*.

- **Dumps:**
  Point-in-time *copy of the state* (memory, registers) of a process or operating system which is created *without assistance of the component* being dumped. Examples for dumps: core dump (user space), kdump (operating system dump)
What is missing?
Types of debugging data

- **Component state-save (snapshot/dump):**
  Component-assisted *point-in-time copy* of *selected component-internal state* data, annotated with meta-data.

- **FFDC Log:**
  The FFDC Log provides a means to create a global sequence of timestamped messages related to *events* that may be *relevant for debugging purposes*. The FFDC log is *persistent*. FFDC log messages are *targeted at developers* or service personal, not primarily system administrators.

- **FFDC Statistics:**
  Aggregated counters or counter rate. *Used to reduce trace data* amount. Not the same as performance statistics.
What is missing?
FFDC transport and repository mechanism

- **FFDC snapshots**
  - Persistent collection of debugging data including descriptive meta-data
  - Snapshots can contain all kinds of debugging data (logs, traces, state save, dumps).

- **FFDC repository**
  - Persistent data store for FFDC snapshots
  - Interface to manage (list, delete, report...) snapshots

- **FFCD snapshot API**
  - For transport of debugging data to FFDC repository
  - Kernel API
  - User space API (library, CLI, different language bindings)
What is FFDC?
Proposal
Action Plan
FFDC snapshot runtime

Kernel

Component

Snapshot API

Push snapshots

Snapshot daemon

Store

FFDC snapshot repository
FFDC snapshot runtime

Kernel

Component

Snapshot API

Push snapshots

Snapshot daemon

Store

Push snapshots

Snapshot API

Component

Userspace

FFDC snapshot repository
FFDC snapshot runtime

- Request snapshot (trigger/pull)
  - Userspace
    - Monitor
    - Snapshot API
    - Request snapshot (trigger/pull)
  - FFDC snapshot repository
    - Store
  - Kernel
    - Component
      - Snapshot API
        - Push snapshots
    - Snapshot daemon
      - Store
      - Push snapshots
      - Snapshot API
        - Component
    - Userspace
FFDC snapshot runtime

- **Userspace**
  - Monitor
  - Snapshot API
  - Request snapshot (trigger/pull)
  - Reporting/Housekeeping

- **Kernel**
  - Component
  - Snapshot API
  - Push snapshots

- **FFDC snapshot repository**
  - Store
  - Manage snapshots
    - List
    - Delete
    - Extract
    - Report
  - Push snapshots

- **Snapshot daemon**
  - Snapshot API
  - Component
FFDC snapshot/state-save API

- Register FFDC component (struct ffdc_info)
  ```c
  struct ffdc_info
  struct ffdc_info *ffdc_register(const char *id, ...);
  ```

- Create snapshot (struct ffdc_snap)
  ```c
  struct ffdc_snap;
  struct ffdc_snap *ffdc_snap_begin(struct ffdc_info *ffdc_info, const char *reason, ...);
  void ffdc_snap_add_meta(struct ffdc_snap *ffdc_snap, const char *key, const char *value, ...);
  void ffdc_snap_add_blob(struct ffdc_snap *ffdc_snap, const char *type, void *buf, size_t len);
  void ffdc_snap_end(struct ffdc_snap *ffdc_snap);
  ```

- FFDC snapshot callback (ffdc_snap_cb)
  ```c
typedef void (*ffdc_snap_cb)(struct ffdc_snap *snap, void *data);
  int ffdc_snap_register(struct ffdc_info *ffdc_info, ffdc_snap_cb snap_cb, void *data);
  ```
Kernel component snapshot/state-save

- Saves relevant component state
- Consistent data view (uses component locking)
- Debugfs: /sys/kernel/debug
  - Kernel component initiated
    - ffdc/snapshot_stream
    - Read by snapshot daemon
  - User space initiated (e.g. by a monitor)
    - ffdc/<component>/snapshot
    - Uses snapshot callback
- Key/value ASCII meta data + binary data
- CPIO archive

```
<component>/<timestamp>/meta
  0.blob
  0.meta
  1.blob
  1...
```
What is FFDC?
Proposal
Action Plan
What to do for better Linux FFDC?

- **Define FFDC recommendations**
  - Which tracepoints should be enabled?
  - How much overhead is acceptable?
  - Which error classes?

- **Enable traces for FFDC**
  - Define initial trace settings
  - Allow access for snapshots

- **Define component relationship and namespace**

- **Define relevant component state (state-save)**

- **Define snapshot transport**
  - snapshot API, runtime and repository
Thank you!