DPDK

DATA PLANE DEVELOPMENT KIT

Supporting Cloud Native with DPDK and containers

KEITH WILES @ INTEL CORPORATION

Making Applications Cloud Native Friendly

- How can we make DPDK Cloud Native Friendly?
  - Reduce startup resources for quicker startup
  - Make it simpler to startup and run a DPDK application
  - Make it easy to monitor the application
  - Make it easy to configure the application during runtime
  - Make it easy to attach/detach hardware
  - Make it easy to create new virtual interfaces
  - Make it easy for non-DPDK applications to connect to DPDK owned hardware

What can DPDK do?

- Needs to have less command line options
- Needs to be configurable at runtime
- Easy to configure during runtime
- Easy for orchestration to change and monitor DPDK apps
- Simpler set of APIs for non-DPDK experts to use in applications
- Improve performance for data movement to/from containers

Simplify DPDK Startup

- Command lines are great for developers not so much for others
- Simplify required command line options
- Needs to startup quickly with minimum resources
  - lcores, memory/hugepages, devices, threads, …
  - Can we make DPDK startup with just a thread
  - Then add the resources to DPDK as needed via a runtime configuration
- What does DPDK need to make this happen

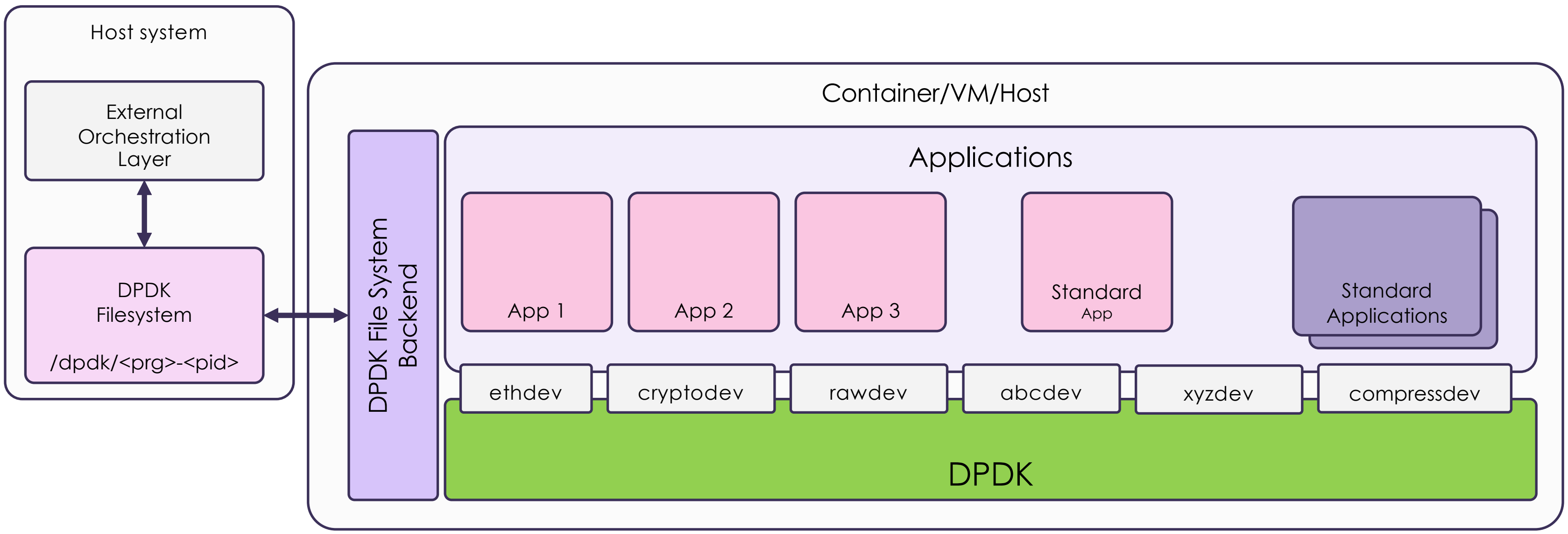
Dynamic DPDK Resources

- If DPDK started with minimum resources we need to have ways to add these resources at runtime
- DPDK needs to have dynamic lcore support
  - We need to add support to add/remove lcores at runtime
  - We have a PoC that is able to add/remove lcores
- Memory resources in DPDK is coming along nicely and Anatoly has done a great job in reworking DPDK memory system to be much more dynamic
- DPDK dynamic hardware support, will hotplug work here
- Need dynamic virtual interfaces like virtio, tap, …

DPDK File System

A FILE SYSTEM FOR DPDK TO CONFIGURE AND MONITOR DPDK

DPDK File System (DFS)



- DPDK File System backend provides the connection between the FUSE filesystem to app
- Each DPDK instance has it own filesystem path and configuration/information files
- The external or orchestration agents interact with the FUSE filesystem to Get/Set information/configuration via files
- Also provides an API for applications to modify the FUSE file system dynamically

FUSE information

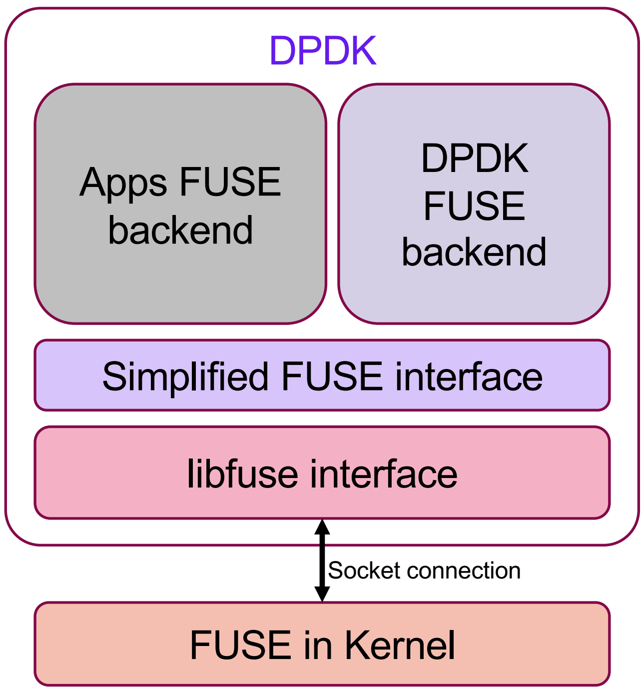
- Create a FUSE or User space file system similar to /proc or /sys in the kernel
- The DFS is backed by application code to handle read/write requests
- The read or write request is then handled by that application to supply the data

- From the Wiki https://en.wikipedia.org/wiki/Filesystem_in_Userspace

The FUSE system was originally part of *AVFS* (*A Virtual Filesystem*), a filesystem implementation heavily influenced by the translator concept of the GNU Hurd.[3]

FUSE was originally released under the terms of the GNU General Public License and the GNU Lesser General Public License, later also reimplemented as part of the FreeBSD base system[4] and released under the terms of Simplified BSD license. An ISC-licensed re-implementation by Sylvestre Gallon was released in March 2013,[5] and incorporated into OpenBSD in June 2013.[6]
FUSE was merged into the mainstream Linux kernel tree in kernel version 2.6.14.[7]

Simplified Interface to FUSE



- Simple API to create new files and directories
- Callbacks from libfuse is a simple set of events open, release, read, write and init
- Creating files is a simple structure with optional API to create files/directories

Example directory layout

```
/dpdk/
└── dfs-12345
    ├── copyright
    ├── debug
    │   ├── dump_fs
    │   ├── hash
    │   ├── scratch
    │   └── sizes
    │       ├── dfs
    │       └── dfs_node
    ├── eal
    │   ├── bus
    │   │   ├── buses
    │   │   ├── dpaa
    │   │   ├── fslmc
    │   │   ├── ifpga
    │   │   ├── pci
    │   │   └── vdev
    │   ├── config
    │   ├── lcore-cnt
    │   ├── lcore-list
    │   ├── roles
    │   └── socket-cnt
    ├── ethdev
    │   ├── avail_count
    │   └── total_count
    ├── fuse-version
    ├── mempool
    │   ├── dump
    │   └── info
    ├── pid
    ├── rawdev
    │   └── count
    ├── ring
    │   └── info
    ├── timer
    │   └── dump
    ├── type
    └── version
```

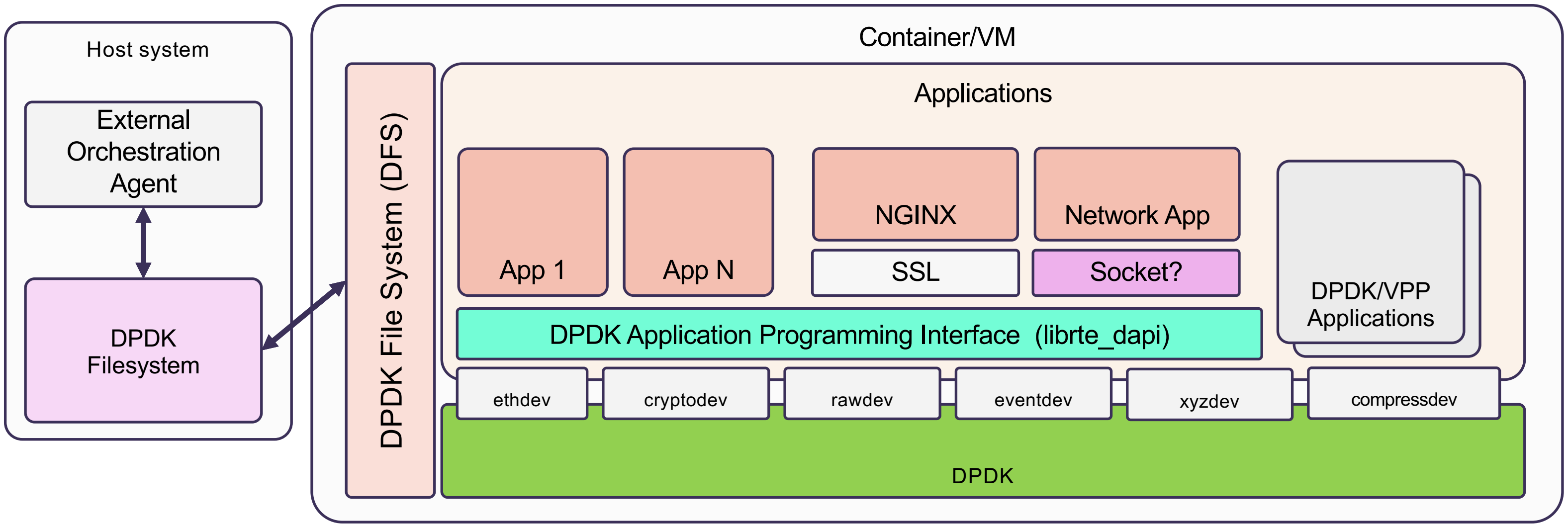
- Most data or complex information is formatted as JSON
- Simple data output e.g. lcore-cnt is just a simple ASCII number
- Developer only needs to define the files/directories and what type of access Read/Write
- The 'libfuse3' library provides the connection to the kernel fuse code and file system handling opcodes
- The libfuse3 code gets messages from the kernel and handles the request in a layer hidden from the developer
- The layer the developer deals with is a simplified set of function callbacks to inform the developer about a few actions, but most of the data movement and files system actions are handled in the fuse layer
- Files and/or directories can be added or removed dynamically
- Applications can also add to DFS by creating /dpdk/<appName>/…

DPDK API (DAPI)

A Work In Progress

SIMPLIFIED
PROGRAMMING
INTERFACE FOR
NON-DPDK EXPERTS

DAPI



- New DPDK library librte_dapi (optional for applications)
- Providing a higher layer abstraction for applications using standard DPDK APIs
- Giving the application developer a simpler set of APIs, which helps hide some of the more complexed APIs in DPDK and/or structures, but still able to use DPDK APIs
- Hiding the nature of the hardware or software under the hood allowing the DAPI layer to decide which type to use

DAPI Goals

- Simplify/combine DPDK APIs into higher level API (Not a one for one substitution)
- Allows the application to still call DPDK standard APIs
- APIs for configuration are combined into a single API with attributes
- MBUFs are now abstracted objects, to discourage direct access
- Add a ‘file descriptor’ like index system instead of port/queue IDs (open/close)
- New Rx/Tx APIs do not effect performance in current testing
- Hide the polling loop inside the DAPI layer away from the application
- Have these APIs hide the DPDK performance specific details
- Data access is done by providing functions to set/get the data
- Utilize Macros and inline functions to create the new API

DAPI Goals

- Applications can still use DPDK standard APIs if needed
- Hides the internals of DPDK with opaque objects and structures
- Uses default values in its APIs to eliminate complexed
- Data path, must be light weight and very high performance (no real impact)
- Must abstract the internals of DPDK like mbufs from the application
- Provide a simple set of APIs to access the mbufs (some already exist)
- Standard DPDK utility libraries e.g. Hash, Ring, cmdline, … should not have new APIs as they are normally easy to use, except cmdline :-)

Example DAPI Prototypes

- int dapi_eal_init(struct dapi **ret_dapi, int argc, char *argv[]);
  - Wrapper around rte_eal_init(), dapi_create(), dfs_create(), …
- int dapi_open(const char *devname, int flags);
  - Returns the ‘dd’ index into the device descriptor table
  - The devname is a simple string with the port ID and Queue ID encoded into the string or add your own set of strings
    - e.g. “/ethdev/dev-<pid>:<qid>” the ‘/ethdev/’ (prefix maybe optional)
    - Use dapi_register_devname() for different device naming strings
- int dapi_close(int dd);

Example of DAPI

- int dapi_pktbuf_pool_create(int dd, unsigned int nb_bufs, unsigned int cache_size, uint16_t data_size);
  - Similar to rte_pktmbuf_pool_create() but reduce to basic needed arguments.
- int dapi_default_port_configs(struct dapi *dapi, portlist_t portlist, struct port_cfg *cfg)
  - Setup the port_cfg structure for each port in the portlist as a default value
- int dapi_eth_port_setup(int dd, struct port_cfg *p, uint32_t flags);
  - Single line to setup and configure a port based on port_cfg or defaults if NULL
  - The above API sets up the configuration defaults if needed

Example of DAPI

- The single pktbuf_t allocation/free routines

  int dapi_pktbuf_alloc(int dd, pktbuf_t *pkt)

  - The pktbuf_t is just a void* to hide the mbuf pointer

  int dapi_pktbuf_free(pktbuf_t *pkt);

- The pktbuf_t allocation/free routines for multiple packets

  int dapi_pktbuf_alloc_bulk(int dd, pktbuf_t **pkts, unsigned nb_bufs);

  - Allocate or free multiple pktbuf_t pointers (these are the mbuf pointers)

  int dapi_pktbuf_free_bulk(pktbuf_t **pkts, unsigned nb_bufs);

Example of DAPI

- Pktio APIs
- Single pktbuf_t read/write routines

```
int dapi_pktio_read(int dd, pktbuf_t **pkt);
int dapi_pktio_write(int dd, pktbuf_t *pkt);
```

- Multiple pktbuf_t read/write routines

```
int dapi_pktio_read_multi(int dd, pktbuf_t **pkts, int nb_pkts);
int dapi_pktio_write_multi(int dd, pktbuf_t **pkts, int nb_pkts);
```

- The pktbuf_t writes are buffered and sent when flushed or the array is filled

```
int dapi_pktio_flush(int dd);
```

PKTIO DAPI APIs

- Pktio APIs
- int dapi_pktio_set_len(pktbuf_t *pkt, uint16_t len);
- int dapi_pktio_get_len(pktbuf_t *pkt);
- int dapi_pktio_get_buflen(pktbuf_t *pkt);
- void *dapi_pktio_ptod(pktbuf_t *pkt);

- Some of these are already in rte_mbufs, trying to not create one to one APIs
- I have not listed all of the APIs here

- The API is a Work in Progress and any help would be great

PKTIO DAPI APIs

- Using the dapi_open() routine you can define the files to port mapping
- int dapi_register_devnames(struct dapi *dapi, struct dapi_devname *dn);
  - struct dapi_devname dn[] = { {.name = “/ethdev/eth0”, .pid = 1, .qid = 2},
    { .name = “/ethdev/eth1”, .pid = 4, .qid = 0},
    { .name = “/crypto/crypto0”, .pid = 3, .qid = 0},
    { .name = “/ethdev/40g-0”, .pid = 5, .qid = 0}, { .name = NULL } };
  - The .name contains dev-<pid>:<qid> or can be any string, could be dynamic as well
  - If the register call is not done then dapi_open() will expect the string to have a ‘dev-<pid>:<qid>’ string segment ‘/ethdev’ or ‘/crypto’ strings maybe something we may need to identify a class of devices
- int dapi_remote_launch(struct dapi *dapi, lcore_function_t func, void *arg, unsigned lcore_id);
  - Used to launch the function from user and set the this_dapi core local variable that hides details

"

Questions?

"