POK Operating System

Julien Delange <julien@gunnm.org>
Forewords

• The POK project
  • Design and implement safe and secure system
  • Complete development process with model-based engineering

• Now, focus on the underlying operating system
  • Main guidelines
  • Architecture, services
  • Go into code organization
Outline

• Introduction

• Overall architecture

• Kernel layer

• Partition Layer

• Conclusion
Outline

• Introduction

• Overall architecture

• Kernel layer

• Partition Layer

• Conclusion
Introduction

• Partitioning functionalities
  • Time isolation across partitions
  • Space isolation – segments and communication control

• Interoperability
  • Ada and C programming interfaces
  • ARINC653 compliance
  • POSIX compliance

• Embedded architectures support
  • X86/QEMU
  • PPC/QEMU
  • SPARC/Leon
One OS, two layers

• Kernel layer
  • Critical functions
  • Few services, prone to verification/certification
  • Actually < 6000 SLOC

• libpok
  • Non-critical functions
  • All remaining services
  • Actually ~ 20000 SLOC
Project guidelines

- **Real-Time compliance**
  - $O(1)$ algorithms

- **High-integrity compliance**
  - Static allocation of kernel/partitions resources
  - Avoid dead code, useless functionalities

- **Embedded systems compliance**
  - Low complexity
  - Avoid memory overhead
Development guidelines

- Reduce critical code
  - Few services in kernel
  - Remaining services in libpok
- Each snapshot must work
  - Compilation of all examples on all architectures
  - Prevent functionalities breakage
- Enforce coding style
  - Use rules inspired by best-practices
  - See MISRA-C for example
  - Look at doc/CODING_GUIDELINES
Naming guidelines

- Resources dimensioning
  - POK_CONFIG_NB_* macros
  - Ex: POK_CONFIG_NB_THREADS

- Service configuration, services inclusion
  - POK_NEEDS_FUNCTIONS
  - Ex: POK_NEEDS_TIME, POK_NEEDS_SCHED, ...

- Headers
  - #ifndef __POK_SERVICE_NAME_H__
  - Ex: #ifndef __POK_SCHED_H__
Naming guidelines – cont'd

• Types
  • pok_typename_t
  • See include/types.h for types definitions
  • Ex: pok_partition_t, type that contains partition attributes

• Functions
  • pok_servicename_functionname ()
  • Ex: pok_partition_load, function that loads a partition
Outline

- Introduction
- Overall architecture
- Kernel layer
- Partition Layer
- Conclusion
Main architecture

Partitioning Kernel

Partition 1

Partition 2

- Intra-partition comm.
- ARINC653 & POSIX layers
- Libc & libm support
- Ada layer
- Device drivers
- Ciphers algorithms
- Time & space partitioning
- I/O interface
- Scheduling

Located in libpok/ directory

Located in kernel/ directory

Julien Delange <julien@gunnm.org> - http://pok.gunnm.org
Kernel architecture

Partitions

Interface with partitions

Wrapper to architecture functions
(see core/thread.c)

Schedule threads & partitions
(see core/sched.c)

Wrapper to low-level layer
(unified time representation)
(see core/time.c)

Thread

Scheduling

Time

Architecture support

Communication service

Partitioning

Memory

Errors

Queuing and sampling ports manager
(see middleware/directory and core/lockobj.c)

Load partitions, separate resources
(see core/partition.c)

Propagate errors to partitions
(see core/error.c)

Context-switch functions, access to clock,
exception handlers, memory segments
manager (see arch/) directory

Julien Delange <julien@gunnm.org> - http://pok.gunnm.org
We love globvars ...

• Globvars = hell !
  • Don't use them, it introduces too many bugs !

• ... but useful when programming a kernel
  • pok_thread_t threads[POK_CONFIG_NB_THREADS]
  • pok_partition_t pok_partitions[POK_CONFIG_NB_PARTITIONS]
  • pok_port_t pok_ports[POK_CONFIG_NB_PORTS]
  • Used very carefully inside the kernel !
### Libpok architecture

**DES/Blowfish/ceasar cipher algorithms, OpenSSL backport** (see `protocols/` directory)

**Math functions**, backport from NetBSD (see `libm/` directory)

**Restricted libc** (see `libc/` directory)

**Deterministic memory allocator** see (`core/allocator.c`)

**Access to kernel functionalities:**
- thread creation
- lockobj management
- inter-partitions comm.
- see `include/syscall.h`

**Partial support for** ARINC653 APEX (see `ada/` and `arinc653/` directories)

**Cipher algorithms**

**Libc**

**Device drivers**

**Lib math**

**POSIX layer**

**ARINC653 layer**

**Lib math**

**Errors handling**

**Kernel interface**

**Intra-partition comm.**

**Errors handling**

**Architecture support**

**Blackboards, buffers, events and semaphores handling** (see `middleware/` directory)

**Architecture-specific types** (see `arch/` directory)

**Ethernet network in partition space, polling mode** (see `drivers/` directory)
Services separation

- No globvars
  - Too many potential interactions

- Easily add/remove services
  - Rely on POK_NEEDS_* macros

- Services configuration
  - POK_CONFIG_* macros
Error Handling

● Kernel level
  • Kernel layer function
  • Discriminant: raised error

● Partition level
  • Kernel layer function
  • Discriminants: raised error, faulty partition

● Process level
  • Dedicated thread in each partition
  • Discriminants: raised error, faulty partition
  • Kernel receives exception and activates the thread
Process level error handling

Identify faulty partition and its associated faulty process

Detects error (ex: divide by 0)

Hardware

Julien Delange <julien@gunnm.org> - http://pok.gunnm.org
Outline

• Introduction

• Overall architecture

• Kernel layer

• Partition Layer

• Conclusion
Sources organization

• arch/
  • Architecture/BSP-dependent files
  • Ex: arch/x86/x86-qemu

• core/
  • Mainly wrappers to architecture-dependent services
  • Maintain isolation across partitions
  • Partitions loading

• middleware/
  • Inter-partitions communication
Resources organization

• Statically defined
  • All resources are statically defined
  • Massive use of arrays

• No memory allocation in kernel layer
  • Analysis purpose
  • Ease further certification/verification
typedef struct
{
    uint32_t              base_addr;
    uint32_t              base_vaddr;
    uint32_t              size;
    *name;
    uint32_t              nthreads;
    uint8_t               priority;
    uint32_t              period;
    pok_sched_t           sched;
    uint32_t (*sched_func)(uint32_t low, uint32_t high);
    uint64_t              activation;
    uint32_t              current_thread;
    uint32_t              thread_index_low;
    uint32_t              thread_index_high;
    uint32_t              thread_index;
}
    pok_partition_t;

Indicates where the threads of a partition reside.

Bound partition accesses in the lockobjects array.

Critical functions must absolutly CHECK these bound to enforce resources isolation.
Resources organization

- pok_partition_t* pok_partitions
  - Partition 1
  - Partition 2
- pok_port_t* pok_ports
  - Port 1
  - Port 2

- pok_partition_t* pok_partitions
  - Thread1
  - Thread2
  - Thread3
  - Thread4

- pok_partition_t* pok_partitions
  - Task1
  - Task2

Kernel
Kernel startup
(core/boot.c)

```c
void pok_boot ()
{
    pok_arch_init();
pok_bsp_init();

#if defined (POK_NEEDS_TIME) || defined (POK_NEEDS_SCHED) || defined (POK_NEEDS_THREADS)
pok_time_init();
#endif

#if defined POK_NEEDS_PARTITIONS
    pok_partition_init ();
#endif

#if defined POK_NEEDS_THREADS
    pok_thread_init ();
#endif

#if defined (POK_NEEDS_SCHED) || defined (POK_NEEDS_THREADS)
pok_sched_init ();
#endif

#if (defined POK_NEEDS_LOCKOBJ) || defined (POK_NEEDS_PORTS_QUEUEING) || defined (POK_NEEDS_PORTS_SAMPLING)
pok_lockobj_init ();
#endif
#if defined (POK_NEEDS_PORTS_QUEUEING) || defined (POK_NEEDS_PORTS_SAMPLING)
pok_port_init ();
pok_queue_init ();
#endif

#if defined (POK_NEEDS_DEBUG) || defined (POK_NEEDS_CONSOLE)
pok_cons_write("POK kernel initialized\n", 23);
#endif

    pok_arch_preempt_enable();
}
```

Activate services according to system requirements (POK_NEEDS*)

Initialize kernel services
Sources organization

- **arch/**
  - Architecture/BSP-dependent files
  - Ex: `arch/x86/x86-qemu`

- **core/**
  - Mainly wrappers to architecture-dependent services
  - Maintain isolation across partitions
  - Partitions loading

- **middleware/**
  - Inter-partitions communication
**Important functions**

- **pok_sched()**
  - Called every tick
  - Reschedule the system, enforce time isolation
  - Flush partitions ports when major time frame is reached

- **pok_core_syscall()**
  - Interface with partitions
  - Highly critical
  - Check space isolation of arguments
Important functions – cont'd

- **pok_port_flushall ()**
  - Called at each major time frame
  - Flush partitions ports

- **pok_error_declare ()**
  - Wakeup the error process in the partition
  - Complete necessary information to handle the error (error type, faulty thread)
Important functions – cont'd

- `pok_partition_error()`
  - Raised an error at the partition level
  - Error handler differentiates each partition

- `pok_kernel_error()`
  - Raise an error at kernel level
Important variables

- `pok_threadt_t* threads`
  - Contain informations about all threads of all partitions
  - Include IDLE and KERNEL threads
  - Access to current thread: `POK_CURRENT_THREAD`

- `pok_partitiont_t* pok_partitions`
  - Array of all partitions
  - Statically defined
  - Used everywhere in the sources
  - Access to the current partition: `POK_CURRENT_PARTITION`
Important variables – cont'd

• **pok_port_t** *pok_ports*
  - Information about ALL ports of ALL partitions
  - Used in the middleware layer (sampling & queuing ports)

• **pok_queue_t** pok_queue
  - Big array
  - Contain data of ALL ports of the current node
  - Statically bound
Important variables – cont'd

- `uint64_t current_time`
  - Amount of elapsed ticks
  - Clock granularity in POK: 1ms
Outline

- Introduction
- Overall architecture
- Kernel layer
- Partition Layer
- Conclusion
Architecture-dependent layer

- Syscall handling ...
  - Marshall syscall args
- … but mainly types interfacing
  - Do not handle many low-level functions
  - Use unprivileged rights (ring 3 on x86)
- Other functions on x86
  - PCI management
  - Input/Output ports
Kernel interfacing

• Access to kernel functionalities
  • Threads
  • Mutexes
  • Partition management
  • Inter-partitions ports
  • Time
  • ...

• Use software interrupts (aka syscalls)
  • Syscalls functions for many potential arguments
  • See. include/core/syscall.h
Intra-partition comm.

• Four communications functionalities
  • Blackboard
  • Buffer
  • Semaphore
  • Event

• Resources statically allocated
  • cf. POK_CONFIG_NB_BLACKBOARDS, POK_CONFIG,NB_BUFFERS, …
  • Same mechanisms as inter-partitions functionalities in kernel

• Rely on kernel interfacing functions
  • Require mutex handling
Errors handling

• Error process handler creation
  • cf. `pok_error_handler_create ()`
  • Create a task, entrypoint=`pok_error_handler_worker`

• Error handler internals
  • Declare as ready (`pok_error_handler_set_ready ()`)
  • Catch an error (`pok_error_get ()`)
  • Handle error (written by the developer)

• Execution of error process
  • Executed as soon as an error is raised
Errors handling – con't

• Raise application error
  • cf. *pok_raise_application_error* ()
  • Report errors to the error handler process
Error worker example

```c
void pok_error_handler_worker ()
{
   pok_error_status_t error_status;
   while (1)
   {
      pok_thread_stop_self ();
      pok_error_get (&(error_status));
      switch (error_status.failed_thread)
      {
         case 1:
            { case POK_ERROR_KIND_APPLICATION_ERROR:
               { pok_thread_restart (1);
                 break;
               }
            }
            case POK_ERROR_KIND_NUMERIC_ERROR:
            { pok_partition_set_mode (POK_PARTITION_MODE_INIT_WARM);
              break;
            }
            break;
      }
   break;
   }
}
```
libc

• Memory allocation (stdlib.h)
  • Rely on the deterministic memory allocation

• String handling (string.h)
  • memcmp(), strcpy(), ...

• Basic input/output (stdio.h)
  • Basic printf()

• Partial implementation
  • Some functions are missing (see include/libc/)
  • Easily extendable with code reuse from NetBSD

Julien Delange <julien@gunnm.org> - http://pok.gunnm.org
Device drivers

- Requires access to low-level concerns
  - Reservation of low-level access at initialization time
  - No reservation allowed at runtime
- Polling mode
- Experiment with one device
  - Realtek 8029, network device of QEMU
  - See drivers/ directory
Device drivers – cont'd

- Constrained to partition restrictions
  - Time & space isolation
  - Communication with other partitions using inter-partitions mechanisms

- Analyze end to end latency
  - Time isolation increases response time
  - See impact of the major frame

- Incoming work on this topic
  - Preliminary work with implementation of HFPPS scheduling algorithm (cf. Burns and Nolte work)
ARINC653 layer

- Implementation of ARINC653 APEX
  - Definition of APEX in include/arinc653

- Wrapper to POK legacy API
  - Use kernel interface
  - Use on intra-partition communication

- Complete implementation
  - Almost all functions are implemented
  - Need to synchronize with newer version of the APEX
Lib math

• Access to mathematical functions
  • Required by some application code (Simulink, Lustre)

• Complete implementation
  • Successful usage with Lustre and OpenSSL algorithms

• Port of NetBSD libm
  • No licence conflicts
Cipher algorithms

- Cipher data before sending
  - Prevent data sniffing over ethernet networks
- Implementation of symmetric algorithms
  - Data Encryption Standard (DES)
  - Blowfish
  - Ceasar
- Configuration with dedicated macros
  - `POK_NEEDS_PROTOCOLS`
  - `POK_BLOWFISH_KEY`, `POK_DES_KEY`, ...
- Port of OpenSSL algorithms
Ada ARINC653 layer

- Compliant with standardize ARINC653 APEX
- Wrapper to the C version
  - Keep consistency between types
  - Massive use of `with Interfaces.C`
- Disable Ada runtime
  - Lose benefits of Ada runtime (Task, Protected objects ...)
  - Lightweight implementation
  - `pragma No_Run_Time`
Outline

- Introduction
- Overall architecture
- Kernel layer
- Partition Layer
- Conclusion
Conclusion

- **First *libre* partitioned operating system !**
  - Really *libre* (not a GPL-like software !)

- **Remaining technical challenges**
  - Integrate device drivers with respect to T&S isolation
  - Improve system analysis

- **Improve POK !**
  - Better standard support
  - Feel free to join the POK community
Thanks to ...

- François Goudal
  - Initial project (Gunther)
- Julian Pidancet
  - First version of space isolation
- Laurent Lec
  - Device drivers & Ada/ARINC653 layer
- Fabien Chouteau
  - SPARC/LEON port & Ada/ARINC653 layer
- Tristan Gingol
  - PowerPC/QEMU port
Questions ?