OSPERT’15
A New Configurable and Parallel Embedded Real-time Micro-Kernel for Multi-core platforms

Antonio Paolillo
GOALS

Produce safe and reliable embedded software systems
GOALS

Produce safe and reliable embedded software systems

MEANS

R&D: experimental platform
  ➔ new kernel architecture
GOALS

Produce safe and reliable embedded software systems

MEANS

R&D: experimental platform
    → new kernel architecture

Research: validate good results experimentally
CONSTRAINTS
CONSTRAINTS

⏰ Real-time
CONSTRAINTS

⏰ Real-time

🚀 Embedded
CONSTRAINTS

- Real-time
- Embedded
- Certifiable
Ready list
The symmetric approach
Our approach is asymmetric
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes

$K = \text{Kernel mode}$
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
**Initialisation time**
The master core dispatches all ready processes
**Initialisation time**
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
Initialisation time
The master core dispatches all ready processes
**Initialisation time**
The master core dispatches all ready processes.
Initialisation time
The master core dispatches all ready processes
Expected benefits

Easier design

Less contention $\rightarrow$ improved scalability

Private code and data $\rightarrow$ less cache issues
Remote system calls: `exit()`
System calls are remote
Every process shares a mutex with the master kernel to enqueue system call requests
Exit system call
example of remote system call, the task is done, we must call the scheduler

= System call arguments
Exit system call
example of remote system call, the task is done, we must call the scheduler

✉ = System call arguments
Exit system call

Example of remote system call, the task is done, we must call the scheduler

= System call arguments
Exit system call
example of remote system call, the task is done, we must call the scheduler

= System call arguments
Exit system call
example of remote system call, the task is done, we must call the scheduler

-envelope = System call arguments
Exit system call
example of remote system call, the task is done, we must call the scheduler

✉️ = System call arguments
Exit system call
example of remote system call, the task is done, we must call the scheduler

= System call arguments
Exit system call
example of remote system call, the task is done, we must call the scheduler

✉️ = System call arguments
⏳ = Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler

= System call arguments
= Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler

- System call arguments
- Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler

✉ = System call arguments
☒ = Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler

= System call arguments
= Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler

✉️ = System call arguments
❌ = Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler

Envelope = System call arguments
Clock = Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler

= System call arguments

= Core in user-mode busy loop
Exit system call
example of remote system call, the task is done,
we must call the scheduler

✉ = System call arguments

嗉 = Core in user-mode busy loop
Exit system call
example of remote system call, the task is done, we must call the scheduler
Exit system call
example of remote system call, the task is done, we must call the scheduler
Exit system call
example of remote system call, the task is done, we must call the scheduler
Exit system call
example of remote system call, the task is done, we must call the scheduler
Exit system call
example of remote system call,
the task is done,
we must call the scheduler
Remote system calls: sleep(10)
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends
for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Sleep system call
the task self-suspends for 10 ms
Inter Process Communication mechanism: `send()` and `recv()`
IPC: message passing between processes

Implemented UDP-like API
(send(), recv(), port-based)

It affects process states
→ must be based on remote system calls
IPC protocol: performance issues

Micro-kernel → IPC requires high efficiency

Avoid to overload the master core

The challenge is many-to-one IPC in multicore
IPC with system calls
If a process state transition occurs, call the master
**IPC with system calls**
If a process state transition occurs, call the master

- $\text{= System call arguments}$
- $\text{= Message content}$
Receiver calls `recv()`
It can block the process

- System call arguments
- Message content
Receiver calls `recv()`
It can block the process

- `= System call arguments`
- `= Message content`
Receiver calls `recv()`
It can block the process

- System call arguments
- Message content

Sender

Shared buffer

K

Receiver
Receiver calls `recv()`
It can block the process

- 💌 = System call arguments
- 💌 = Message content

Sender

Shared buffer

K

Receiver
Receiver calls `recv()`
It can block the process

- = System call arguments
- = Message content
Receiver calls `recv()`
It can block the process

-envelope = System call arguments
-envelope = Message content
Receiver calls `recv()`
It can block the process

- `= System call arguments`
- `= Message content`

(Sender) -- Shared buffer -- (Receiver)
Receiver calls `recv()`
It can block the process

- = System call arguments
- = Message content

Sender

Shared buffer

Receiver
Receiver calls `recv()`
It can block the process

- = System call arguments
- = Message content

Sender

Shared buffer

Receiver
Receiver calls \texttt{recv()}.
It can block the process.

$\rightarrow$ = System call arguments
$\rightarrow$ = Message content

Sender

\hspace{1cm} K \hspace{1cm}

\hspace{1cm} Shared buffer \hspace{1cm}

Receiver
Receiver calls `recv()`
It can block the process

- 💌 = System call arguments
- 💌 = Message content
Receiver calls `recv()`
It can block the process

- = System call arguments
- = Message content
Receiver calls `recv()`
It can block the process

- $\text{Message content}$
- $\text{System call arguments}$
Receiver calls `recv()`
It can block the process

- System call arguments
- Message content

Sender

Shared buffer

Receiver

K

BLOCKED

BLOCKED
Receiver calls `recv()`
It can block the process

- `recv()` = System call arguments
- `=` Message content

Sender

Shared buffer

Receiver

BLOCKED

BLOCKED
Sender calls `send()`
It can release some processes
The copy is done locally

- **= System call arguments**
- **= Message content**
**Sender calls send()**

It can release some processes
The copy is done locally

- \[\text{System call arguments}\]
- \[\text{Message content}\]

Sender

\[\text{K}\]

Sender

Shared buffer

Receiver
**Sender calls send()**
It can release some processes
The copy is done locally

- = System call arguments
- = Message content
Sender calls `send()`

It can release some processes

The copy is done locally

- `= System call arguments`
- `= Message content`
Sender calls `send()`

It can release some processes

The copy is done locally

- System call arguments
- Message content
Sender calls `send()`
It can release some processes
The copy is done locally

- = System call arguments
- = Message content
**Sender calls send()**
It can release some processes
The copy is done locally

- = System call arguments
- = Message content

**Diagram:**
- Sender
- Receiver
- Shared buffer
- Blocked

Diagram: Sender calls `send()` to release processes. The copy is done locally. Symbols: ⌨️ = System call arguments, ⌨️ = Message content.
**Sender calls `send()`**

It can release some processes
The copy is done locally

- System call arguments
- Message content

K

Sender

**Shared buffer**

Receiver

BLOCKED
Sender calls `send()`

It can release some processes

The copy is done locally

- System call arguments
- Message content

Sender

Shared buffer

Receiver
Sender calls `send()`
It can release some processes
The copy is done locally

= System call arguments
= Message content
Sender calls `send()`
It can release some processes
The copy is done locally

- = System call arguments
- = Message content
Sender calls `send()`
It can release some processes
The copy is done locally

- 💌 = System call arguments
- 💌 = Message content
Sender calls `send()`
- It can release some processes
- The copy is done locally

= System call arguments
= Message content
**Sender calls `send()`**

It can release some processes

The copy is done locally

- = System call arguments
- = Message content
Sender calls `send()`
It can release some processes
The copy is done locally

- 💌 = System call arguments
- 💌 = Message content

(Sender)

Shared buffer

(K)

(BLOCKED)

(Receiver)
Sender calls \texttt{send} ()

It can release some processes
The copy is done locally

\begin{itemize}
  \item \texttt{Message content} = System call arguments
  \item \texttt{Message content} = Message content
\end{itemize}
The master releases the receiver
The receiver can re-execute the `recv()`

- 💌 = System call arguments
- 💌 = Message content

Sender

Sender

Shared buffer

Receiver

Receiver
The master releases the receiver
The receiver can re-execute the recv()

= System call arguments
= Message content
The master releases the receiver

The receiver can re-execute the `recv()`

- 💌 = System call arguments
- 💌 = Message content
The master releases the receiver
The receiver can re-execute the `recv()`

- = System call arguments
- = Message content
The master releases the receiver
The receiver can re-execute the `recv()`

- [ ] = System call arguments
- [ ] = Message content
The master releases the receiver
The receiver can re-execute the `recv()`

\[\text{System call arguments} = \text{Message content}\]
The master releases the receiver
The receiver can re-execute the `recv()`

- ✉️ = System call arguments
- ✉️ = Message content
Access to the buffer requires synchronisation.
The implementation is hybrid w.r.t. symmetric/asymmetric approaches.

- = System call arguments
- = Message content
FUTURE WORK
FUTURE WORK

Benchmark system calls and IPC scheme
FUTURE WORK

Benchmark system calls and IPC scheme

Analytically bound the protocol
FUTURE WORK

Benchmark system calls and IPC scheme

Analytically bound the protocol

Evaluate real-time schedulers
HIPPEROS = spin-off company of ULB
= family of RTOS

→ New kernel for Real Time Systems