Semantics for the OSEK model

Martin Schwarz

Lehrstuhl für Informatik II, Technische Universität München
Boltzmannstraße 3, D-85748 Garching b. München, Germany
{schwmart}@in.tum.de

October 14, 2009

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

Introduction

- Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
- embedded systems setting
- interrupt driven concurrency
- unified execution framework
  - tasks
  - interrupts
  - priorities
  - pre-emption

OSEK OS

- queue of ready tasks
- scheduler
- Priority Ceiling Protocol (PCP)
  - raise priority when acquiring resources
  - minimize priority inversion
  - rule out dead locks

OSEK OS

- queue of ready tasks
- scheduler
- Priority Ceiling Protocol (PCP)
  - raise priority when acquiring resources
  - minimize priority inversion
  - rule out dead locks

OSEK OS

- queue of ready tasks
- scheduler
- Priority Ceiling Protocol (PCP)
  - raise priority when acquiring resources
  - minimize priority inversion
  - rule out dead locks

OSEK OS

- queue of ready tasks
- scheduler
- Priority Ceiling Protocol (PCP)
  - raise priority when acquiring resources
  - minimize priority inversion
  - rule out dead locks

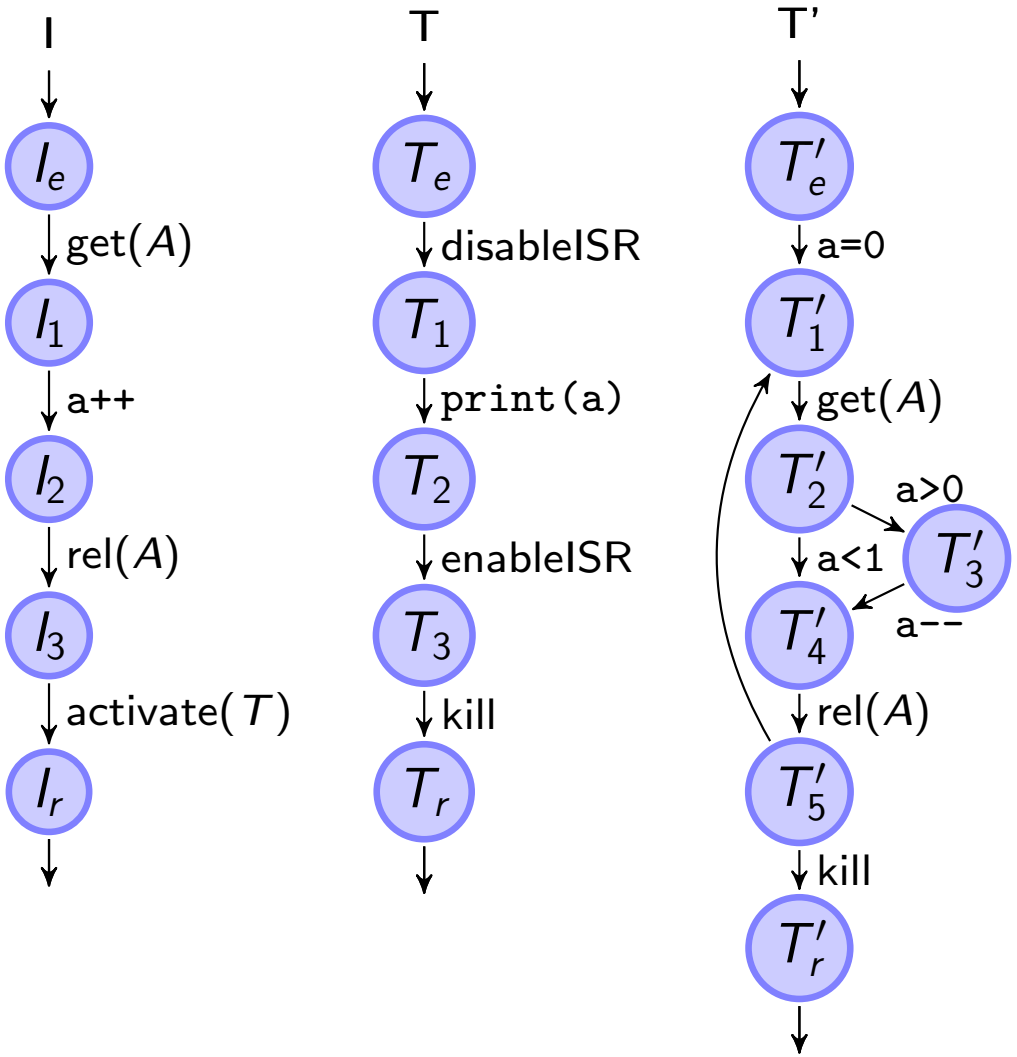
OSEK OS

- queue of ready tasks
- scheduler
- Priority Ceiling Protocol (PCP)
  - raise priority when acquiring resources
  - minimize priority inversion
  - rule out dead locks

OSEK OS

- queue of ready tasks
- scheduler
- Priority Ceiling Protocol (PCP)
  - raise priority when acquiring resources
  - minimize priority inversion
  - rule out dead locks

Example




```
[...]
  TASK T' {
    PRIORITY = 1;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
    RESOURCE = A;
  };
  TASK T {
    PRIORITY = 3;
    AUTOSTART = FALSE;
    ACTIVATION = 2;
  };
  ISR I{
    PRIORITY = 10;
    RESOURCE = A;
  };
[...]
```

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get`(r_{ISR})
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get`(r_{ISR})
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get(`r_{ISR}`)`
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get`(r_{ISR})
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get`(r_{ISR})
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get`(r_{ISR})
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get(`r_{ISR}`)`
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Ideas

- encode all scheduling restrictions in resources
  - `disableISR` $\rightsquigarrow$ `get(`r_{ISR}`)`
- make scheduler calls explicit
  - `rel(A)` $\rightsquigarrow$ `rel(A); schedule`
  - add '`kill`' to interrupts
- simplify the queue
  - keep pre-empted tasks on a stack
  - use the queue only to store *fresh* tasks

Basic edges

$$\frac{(u, c, v) \in E \qquad c \in \text{BASIC} \qquad \langle R', \mathrm{Q}' \rangle = [\![c]\!]\langle R, \mathrm{Q} \rangle}{\langle \Gamma \,;\, \langle u, R \rangle, \mathrm{Q} \rangle \rightarrow \langle \Gamma \,;\, \langle v, R' \rangle, \mathrm{Q}' \rangle}\ \text{BASIC}$$

- stack of pre-empted/interrupted tasks (Γ)
- current node and set of held resources $\langle u, R \rangle$
- priority queue of waiting tasks (Q)
- $[\![\mathrm{get}(r)]\!]\langle R, \mathrm{Q} \rangle = \langle R \cup \{r\}, \mathrm{Q} \rangle$
- $[\![\mathrm{activate}(q)]\!]\langle R, \mathrm{Q} \rangle = \langle R, \mathit{add}(q, \mathrm{Q}) \rangle$

Basic edges

$$\frac{(u, c, v) \in E \qquad c \in \text{BASIC} \qquad \langle R', \mathrm{Q}' \rangle = [\![c]\!]\langle R, \mathrm{Q} \rangle}{\langle \Gamma \,;\, \langle u, R \rangle, \mathrm{Q} \rangle \rightarrow \langle \Gamma \,;\, \langle v, R' \rangle, \mathrm{Q}' \rangle}\ \text{BASIC}$$

- stack of pre-empted/interrupted tasks (Γ)
- current node and set of held resources $\langle u, R \rangle$
- priority queue of waiting tasks (Q)
- $[\![\text{get}(r)]\!]\langle R, \mathrm{Q} \rangle = \langle R \cup \{r\}, \mathrm{Q} \rangle$
- $[\![\text{activate}(q)]\!]\langle R, \mathrm{Q} \rangle = \langle R, \mathit{add}(q, \mathrm{Q}) \rangle$

Basic edges

$$\frac{(u, c, v) \in E \qquad c \in \text{BASIC} \qquad \langle R', \mathrm{Q}' \rangle = [\![c]\!]\langle R, \mathrm{Q} \rangle}{\langle \Gamma \,;\, \langle u, R \rangle, \mathrm{Q} \rangle \rightarrow \langle \Gamma \,;\, \langle v, R' \rangle, \mathrm{Q}' \rangle}\ \text{BASIC}$$

- stack of pre-empted/interrupted tasks (Γ)
- current node and set of held resources $\langle u, R \rangle$
- priority queue of waiting tasks (Q)
- $[\![\text{get}(r)]\!]\langle R, \mathrm{Q} \rangle = \langle R \cup \{r\}, \mathrm{Q} \rangle$
- $[\![\text{activate}(q)]\!]\langle R, \mathrm{Q} \rangle = \langle R, \mathit{add}(q, \mathrm{Q}) \rangle$

Interrupts

$$\frac{q \in \mathsf{Irpt} \qquad \mathcal{P}(\{r_q\}) > \mathcal{P}(R)}{\langle \Gamma \; ; \langle u, R \rangle, \mathsf{Q} \rangle \longrightarrow \langle \Gamma \; ; \langle u, R \rangle \; ; \langle q_e, \{r_q\} \rangle, \mathsf{Q} \rangle} \; \text{IRPT}$$

- (lots of) non-deterministic branching
- termination as tasks via '`kill`'

Interrupts

$$\frac{q \in \mathsf{Irpt} \qquad \mathcal{P}(\{r_q\}) > \mathcal{P}(R)}{\langle \Gamma \,;\langle u, R\rangle, \mathsf{Q}\rangle \longrightarrow \langle \Gamma \,;\langle u, R\rangle \,;\langle q_e, \{r_q\}\rangle, \mathsf{Q}\rangle}\ \text{IRPT}$$

- (lots of) non-deterministic branching
- termination as tasks via '`kill`'

Interrupts

$$\frac{q \in \mathsf{Irpt} \qquad \mathcal{P}(\{r_q\}) > \mathcal{P}(R)}{\langle \Gamma \; ; \langle u, R \rangle, \mathsf{Q} \rangle \longrightarrow \langle \Gamma \; ; \langle u, R \rangle \; ; \langle q_e, \{r_q\} \rangle, \mathsf{Q} \rangle} \; \text{IRPT}$$

- (lots of) non-deterministic branching
- termination as tasks via '`kill`'

Scheduling

$$\frac{(u, \text{schedule}, v) \in E \quad R' = \{r_{hd(\mathrm{Q})}\} \quad \mathrm{Q}' = tl(\mathrm{Q}) \quad \mathcal{P}(R) < \mathcal{P}(R')}{\langle \Gamma \,;\, \langle u, R\rangle, \mathrm{Q}\rangle \rightarrow \langle \Gamma \,;\, \langle v, R\rangle \,;\, \langle hd(\mathrm{Q})_e, R'\rangle, \mathrm{Q}'\rangle} \text{ SWITCH}$$

$$\frac{(u, \text{schedule}, v) \in E \qquad R' = \{r_{hd(\mathrm{Q})}\} \qquad \mathcal{P}(R) \geq \mathcal{P}(R')}{\langle \Gamma \,;\, \langle u, R\rangle, \mathrm{Q}\rangle \rightarrow \langle \Gamma \,;\, \langle v, R\rangle, \mathrm{Q}\rangle} \text{ STAY}$$

- entry state reconstruction
- pre-emption check

Scheduling

$$\frac{(u, \text{schedule}, v) \in E \quad R' = \{r_{hd(\mathrm{Q})}\} \quad \mathrm{Q}' = tl(\mathrm{Q}) \quad \mathcal{P}(R) < \mathcal{P}(R')}{\langle \Gamma \,;\, \langle u, R\rangle, \mathrm{Q}\rangle \rightarrow \langle \Gamma \,;\, \langle v, R\rangle \,;\, \langle hd(\mathrm{Q})_e, R'\rangle, \mathrm{Q}'\rangle} \text{ SWITCH}$$

$$\frac{(u, \text{schedule}, v) \in E \qquad R' = \{r_{hd(\mathrm{Q})}\} \qquad \mathcal{P}(R) \geq \mathcal{P}(R')}{\langle \Gamma \,;\, \langle u, R\rangle, \mathrm{Q}\rangle \rightarrow \langle \Gamma \,;\, \langle v, R\rangle, \mathrm{Q}\rangle} \text{ STAY}$$

- entry state reconstruction
- pre-emption check

Scheduling

$$\frac{(u, \text{schedule}, v) \in E \quad R' = \{r_{hd(\mathrm{Q})}\} \quad \mathrm{Q}' = tl(\mathrm{Q}) \quad \mathcal{P}(R) < \mathcal{P}(R')}{\langle \Gamma \,;\, \langle u, R\rangle, \mathrm{Q}\rangle \rightarrow \langle \Gamma \,;\, \langle v, R\rangle \,;\, \langle hd(\mathrm{Q})_e, R'\rangle, \mathrm{Q}'\rangle} \text{ SWITCH}$$

$$\frac{(u, \text{schedule}, v) \in E \qquad R' = \{r_{hd(\mathrm{Q})}\} \qquad \mathcal{P}(R) \geq \mathcal{P}(R')}{\langle \Gamma \,;\, \langle u, R\rangle, \mathrm{Q}\rangle \rightarrow \langle \Gamma \,;\, \langle v, R\rangle, \mathrm{Q}\rangle} \text{ STAY}$$

- entry state reconstruction
- pre-emption check

Termination

$$\frac{(u, \mathsf{kill}, v) \in E \quad R' = \{r_{hd(\mathsf{Q})}\} \quad \mathsf{Q}' = tl(\mathsf{Q}) \quad \mathcal{P}(\tilde{R}) < \mathcal{P}(R')}{\langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle \; ; \langle u, R \rangle, \mathsf{Q} \rangle \rightarrow \langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle \; ; \langle hd(\mathsf{Q})_e, R' \rangle, \mathsf{Q}' \rangle} \; \text{NEXT}$$

$$\frac{(u, \mathsf{kill}, v) \in E \qquad R' = \{r_{hd(\mathsf{Q})}\} \qquad \mathcal{P}(\tilde{R}) \geq \mathcal{P}(R')}{\langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle \; ; \langle u, R \rangle, \mathsf{Q} \rangle \rightarrow \langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle, \mathsf{Q} \rangle} \; \text{RESUME}$$

- NEXT: indirect pre-emption by $hd(\mathsf{Q})$
- RESUME: return to last pre-empted task

Termination

$$\frac{(u, \mathsf{kill}, v) \in E \quad R' = \{r_{hd(\mathsf{Q})}\} \quad \mathsf{Q}' = tl(\mathsf{Q}) \quad \mathcal{P}(\tilde{R}) < \mathcal{P}(R')}{\langle \Gamma \,;\, \langle \tilde{u}, \tilde{R} \rangle \,;\, \langle u, R \rangle, \mathsf{Q} \rangle \rightarrow \langle \Gamma \,;\, \langle \tilde{u}, \tilde{R} \rangle \,;\, \langle hd(\mathsf{Q})_e, R' \rangle, \mathsf{Q}' \rangle} \text{ NEXT}$$

$$\frac{(u, \mathsf{kill}, v) \in E \qquad R' = \{r_{hd(\mathsf{Q})}\} \qquad \mathcal{P}(\tilde{R}) \geq \mathcal{P}(R')}{\langle \Gamma \,;\, \langle \tilde{u}, \tilde{R} \rangle \,;\, \langle u, R \rangle, \mathsf{Q} \rangle \rightarrow \langle \Gamma \,;\, \langle \tilde{u}, \tilde{R} \rangle, \mathsf{Q} \rangle} \text{ RESUME}$$

- NEXT: indirect pre-emption by $hd(\mathsf{Q})$
- RESUME: return to last pre-empted task

Termination

$$\frac{(u, \mathsf{kill}, v) \in E \quad R' = \{r_{hd(\mathsf{Q})}\} \quad \mathsf{Q}' = tl(\mathsf{Q}) \quad \mathcal{P}(\tilde{R}) < \mathcal{P}(R')}{\langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle \; ; \langle u, R \rangle, \mathsf{Q} \rangle \rightarrow \langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle \; ; \langle hd(\mathsf{Q})_e, R' \rangle, \mathsf{Q}' \rangle} \; \text{NEXT}$$

$$\frac{(u, \mathsf{kill}, v) \in E \qquad R' = \{r_{hd(\mathsf{Q})}\} \qquad \mathcal{P}(\tilde{R}) \geq \mathcal{P}(R')}{\langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle \; ; \langle u, R \rangle, \mathsf{Q} \rangle \rightarrow \langle \Gamma \; ; \langle \tilde{u}, \tilde{R} \rangle, \mathsf{Q} \rangle} \; \text{RESUME}$$

- NEXT: indirect pre-emption by $hd(\mathsf{Q})$
- RESUME: return to last pre-empted task

Conclusion

Theorem

OSEK programs can be modeled as push-down systems

- push-down symbols: $\langle u, R \rangle \in N \times 2^{\text{Res}}$
- states: $Q \in \text{Queue}(\text{Task})$

Conclusion

Theorem

OSEK programs can be modeled as push-down systems

- push-down symbols: $\langle u, R \rangle \in N \times 2^{\text{Res}}$
- states: $Q \in \text{Queue}(\text{Task})$

Conclusion

Theorem

OSEK programs can be modeled as push-down systems

- push-down symbols: $\langle u, R \rangle \in N \times 2^{\text{Res}}$
- states: $\mathrm{Q} \in \mathrm{Queue}(\mathrm{Task})$

Thank you for your attention!

Thank you for your attention!

Questions?