A PROMELA MODEL FOR CONTIKI’S SCHEDULER

Hassan Mousavi, Elham Mahmoudzadeh, Ali Ebnenasir
A PROMELA MODEL FOR CONTIKI’S SCHEDULER

Hassan Mousavi

M.Sc. student in software engineering
Department of Electrical and Computer Engineering
Isfahan University of Technology
OUTLINE

- Motivation
- Challenges
- Research Objectives
- Contributions
MOTIVATION

IoT & CPS in every aspect of our lives
MOTIVATION

Memory-constrained devices
CHALLENGES

- Executing a process
- Interrupt / Timer

Contiki’s scheduler

- Non-preemptive scheduler
- Interrupt-driven
CHALLENGES

- Executing a process
- Posting a synch event
- Context switch
- Posting an Asynchronous event
- Event queue
- Interrupt / Timer

Contiki’s scheduler

- Non-preemptive scheduler
- Interrupt-driven
- Event-driven
CHALLENGES

- Executing a process
- Posting a synch event
- Context switch
- Posting an Asynchronous event
- Event queue
- Interrupt / Timer

Contiki’s scheduler

- Non-preemptive scheduler
- Interrupt-driven
- Event-driven
- Stackless multithreading mechanism
RESEARCH OBJECTIVES

- Extract a formal model that captures concurrency-related behaviors of the process scheduler
- Specify the critical properties of the scheduler
- Verify the properties with respect to the extracted model
- Validate the detected errors/design flaws with respect to the source code
- Repair the flaws of the scheduler at the formal model level
- Repair the flaws at the source code level
PROCESS STATES

- Process activation
  - Adding process to the process list

Diagram:
- Node: NONE
  - Start
  - Termination of process thread or process being killed
- Node: RUNNING
  - Process thread is not assigned
  - Blocking on an event to occur
  - Process waiting
  - Being in Running state
  - Sending an event to the process
  - Polling the process
  - Process execution
  - Adding process to the process list
- Node: CALLED
  - Process thread has already been assigned
  - Process being called
  - Process deactivation

7/22
PROCESS CONTROL BLOCK (PCB)

- Process Control Block (PCB)
- Code of process (process thread)
**EVENT STRUCTURE**

- Event identifier (ev)
- Receiver of event (p)
STARTING POINT OF CONTIKI’S SCHEDULER

process_run()

- processing an event or poll requests to make process/processes execute

```c
1 main()
2 {
3   //
4   //System initial
5   //
6   while(true)
7     process_run()
8 }
```

```c
process.c: 301-313
process.c: 305-307
process.c: 306
process.c: 310
process.c: 312
```
CONTROL FLOW OF CONTIKI’S SCHEDULER

- **main()**
  - Taking the control of the program after system boot-up
  - Transferring the program control to the scheduler in an non-terminating loop after some system initialization

- **do_event()**
  - Removing an event from the event queue if it is not empty
  - Sending the event to the its receiver
  - Two execution path based on the receiver
### C TO PROMELA TRANSFORMATION RULES

#### Rules 1-4

- **Syntactic transformation**

<table>
<thead>
<tr>
<th>#</th>
<th>C</th>
<th>Promela</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic data type</td>
<td>Basic data type</td>
</tr>
<tr>
<td>2</td>
<td>User-defined structure</td>
<td>typedef</td>
</tr>
<tr>
<td>3</td>
<td>Constant or macro</td>
<td>C-style macro</td>
</tr>
<tr>
<td>4</td>
<td>enum</td>
<td>mtype</td>
</tr>
<tr>
<td>5</td>
<td>Function definition</td>
<td>Proctype definition</td>
</tr>
<tr>
<td>6</td>
<td>Function parameter list</td>
<td>Message channel</td>
</tr>
</tbody>
</table>
| 7 | Function call         | An instantiation of the proctype of the callee function +  
                          | A communication channel                          |
| 8 | Pointer to data type  | int                                               |
| 9 | Linked list (nodes + links) | Array + int                                   |
MODELING THE PROCESS LIST/QUEUE

Using rule 9 to transform linked list structure to an array with fix size

Using rules 5,6 to transform process thread to proctype
PROMELA MODEL OF PROCESS_RUN FUNCTION

Rules 5,7
- Function definition to proctype definition
- Using a communication channel for function return value and function call

C code

Promela

Model: 1-13 (process.c: 301-313)
Model: 4-9 (process.c: 305-307)
Model: 6-7 (process.c: 306)
Model: 10-11 (process.c: 310)
Model: 12 (process.c: 312)
Rule 7

- An instantiation of the proctype of the callee function
- Using a communication channel for function return value

Promela

Body of proctype
MODELING THE OCCURRENCE OF INTERRUPTS

process_run(void)

poll_requested ?

Yes

do_poll(void)

No

do_event(void)

return nevents + poll_requested

process_poll(p)

return

Challenge

- Non-deterministic interruption

Interrupt Service Routine (ISR)
ISR MODELING

Model

- Using an active proctype
- Non-terminating loop

ISR() active

end

No

True

Yes

select a random process

process_poll(p)
If the state of a process is **RUNNING** then the process is in process list.

```c
1 #define isInRunningState(p) (processes[p].state==PROCESS_STATE_RUNNING)
2 #define isInProcessList(p) (processes_valid[p] == 1)
3 #define p1(p) (isInRunningState(p) -> isInProcessList(p))
4 ltl prop1 {[]((p1(0) && p1(1) && p1(2) && p1(3) && p1(4))}
```

If the state of a process is **RUNNING** and it receives kill/exit event, then it must eventually exit and transition to the **NONE** state.

```c
1 #define isInRunningState(p) (processes[p].state==PROCESS_STATE_RUNNING)
2 #define lastReceivedEvent(p,ev) (calledProcess_id == p && sent_ev == ev )
3 #define isInNoneState(p) (processes[p].state == PROCESS_STATE_NONE)
4 #define p2(p,ev) ((isInRunningState(p) && lastReceivedEvent(p,ev)) -> (<>isInNoneState(p)))
5 ltl prop2 {[]((p2(0) && p2(1) && p2(2) && p2(3) && p2(4))}
```
If the state of a process is RUNNING then the process is in process list.
DETECTED FLAWS

prop2

- the state of a process is RUNNING and it receives kill/exit event, then it must eventually exit and transition to the NONE state

(process.c: 174-199)
call_process(p, ev, data)

(process.c: 185-198)

process_current = p
p -> state = PROCESS_STATE_CALLED

(process.c: 190)
ret = p -> thread( &p -> pt, ev, data)
CONCLUSIONS

- Presented State diagram of a process
- Extracted The state machine of the scheduler
- Provided Control flow abstraction of the scheduler functions
- Extracted Promela model of the scheduler
- Detected flaws
FUTURE WORK

- Specification and verification more critical properties of Contiki’s scheduler
- Validating the detected errors/design flaws with respect to the source code
- Repair the flaws of the scheduler at the formal model level
- Repair the flaws at the source code level
- Developing a dependable version of Contiki’s scheduler
https://mahmoudzadeh.iut.ac.ir/promela-model-contikis-scheduler
PROTOTHRDEA : CONTIKI'S PROGRAMMING MODEL

```
PROCESS_THREAD(processName, ev, data)
```

```
PROCESS_BEGIN()
```

```
PROCESS_WAIT_EVENT()
```

```
PROCESS_END()
```

```
pThread()
```

```
pThread_params_chan ? eval(_pid), PROCESS_EVENT_INIT, data
```

```
pThread_params_chan ? eval(_pid), ev, data
```

```
pThread_synch_chan ! PT_ENDED
```

```
end
```

```
end
```