Runtime Verification

Where to go from here?

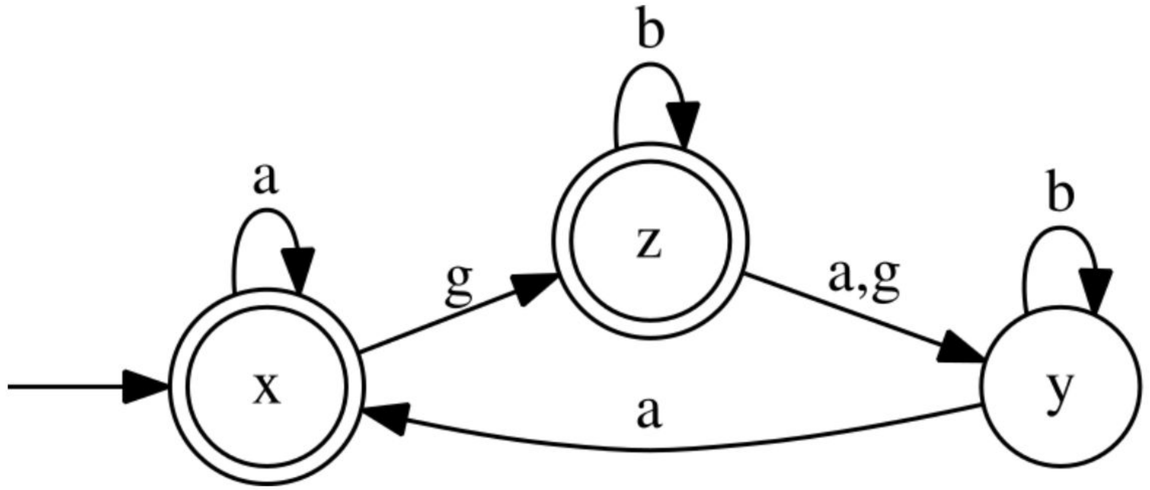
What is Runtime Verification?

Verifies the running system is doing what the model describes

- Hooks into tracepoints to move to a new state
- On a bad state, triggers a reaction (print, crash, etc)
- Very low overhead
- Can be running in a production system

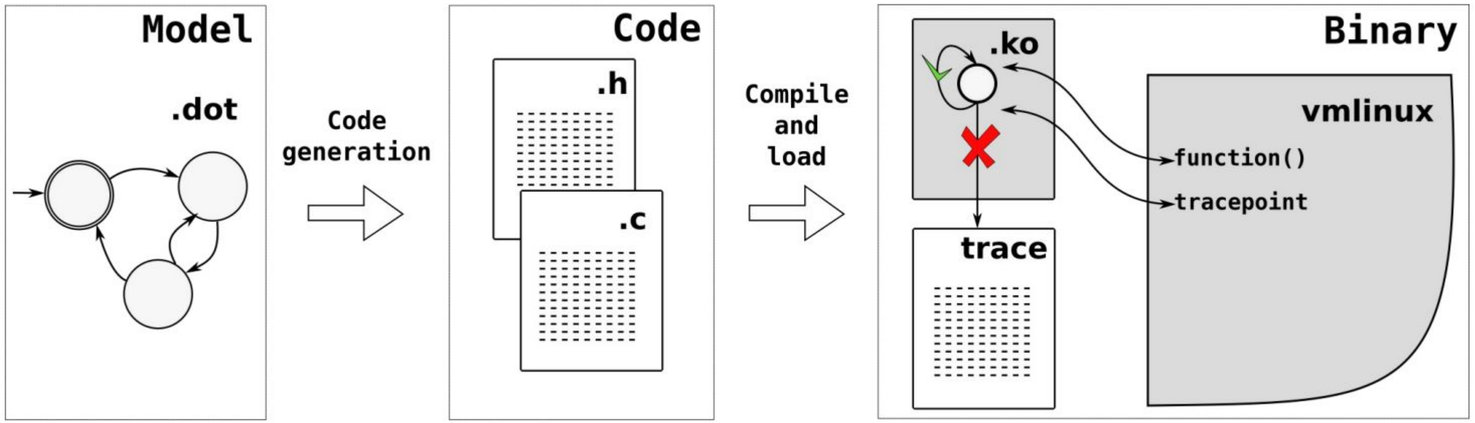
What is Runtime Verifier?

Automata and Discrete Event Systems (DES)



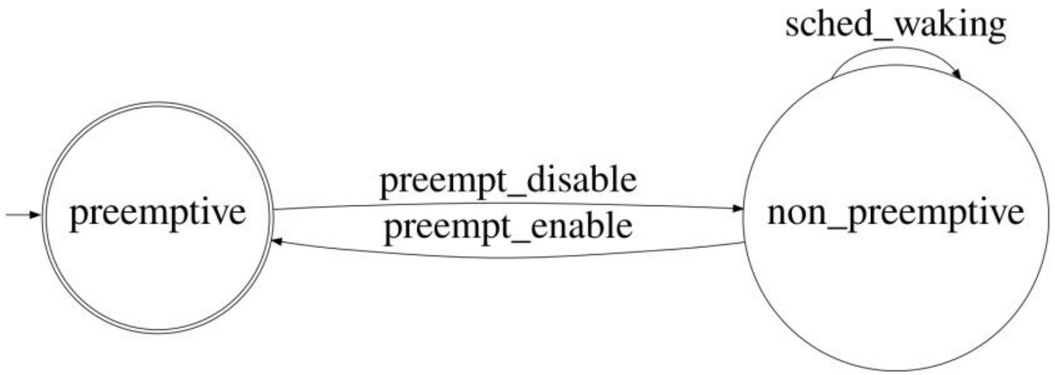
What is Runtime Verifier?

Flow of work



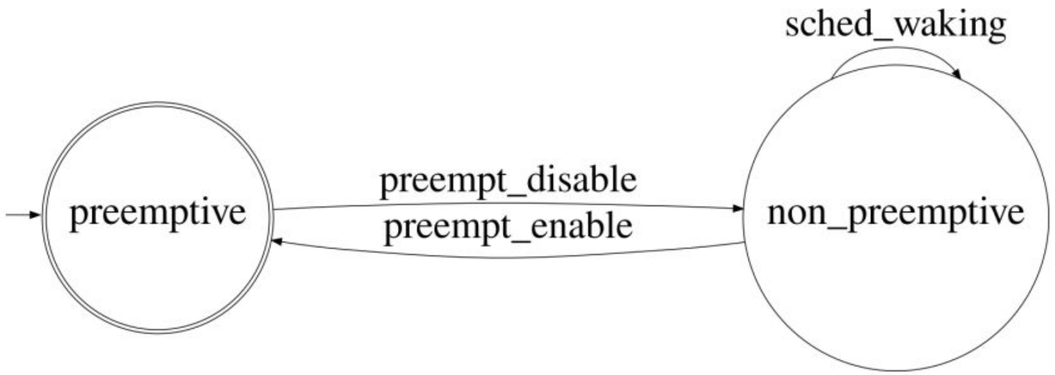
Example

WIP (Work In Progress?)



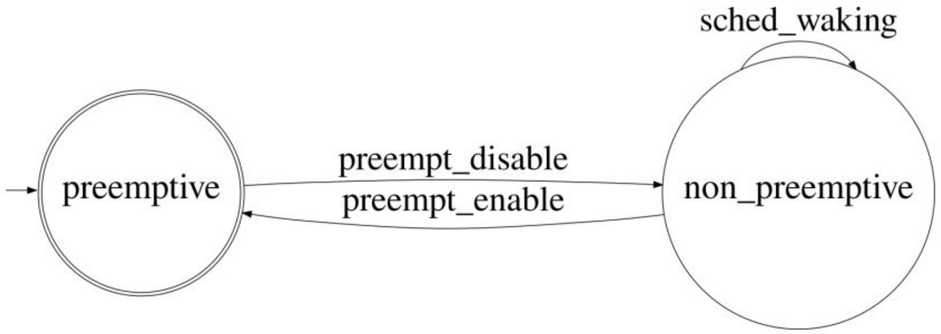
Example

WIP (Wakeup In Preempt)



Example

wip.dot




```
digraph state_automaton {
    {node [shape = circle] "non_preemptive"};
    {node [shape = plaintext, style=invis, label=""] "__init_preemptive"};
    {node [shape = doublecircle] "preemptive"};
    {node [shape = circle] "preemptive"};
    "__init_preemptive" -> "preemptive";
    "non_preemptive" [label = "non_preemptive"];
    "non_preemptive" -> "non_preemptive" [ label = "sched_waking" ];
    "non_preemptive" -> "preemptive" [ label = "preempt_enable" ];
    "preemptive" [label = "preemptive"];
    "preemptive" -> "non_preemptive" [ label = "preempt_disable" ];
    { rank = min ;
    "__init_preemptive";
    "preemptive";
    }
}
```

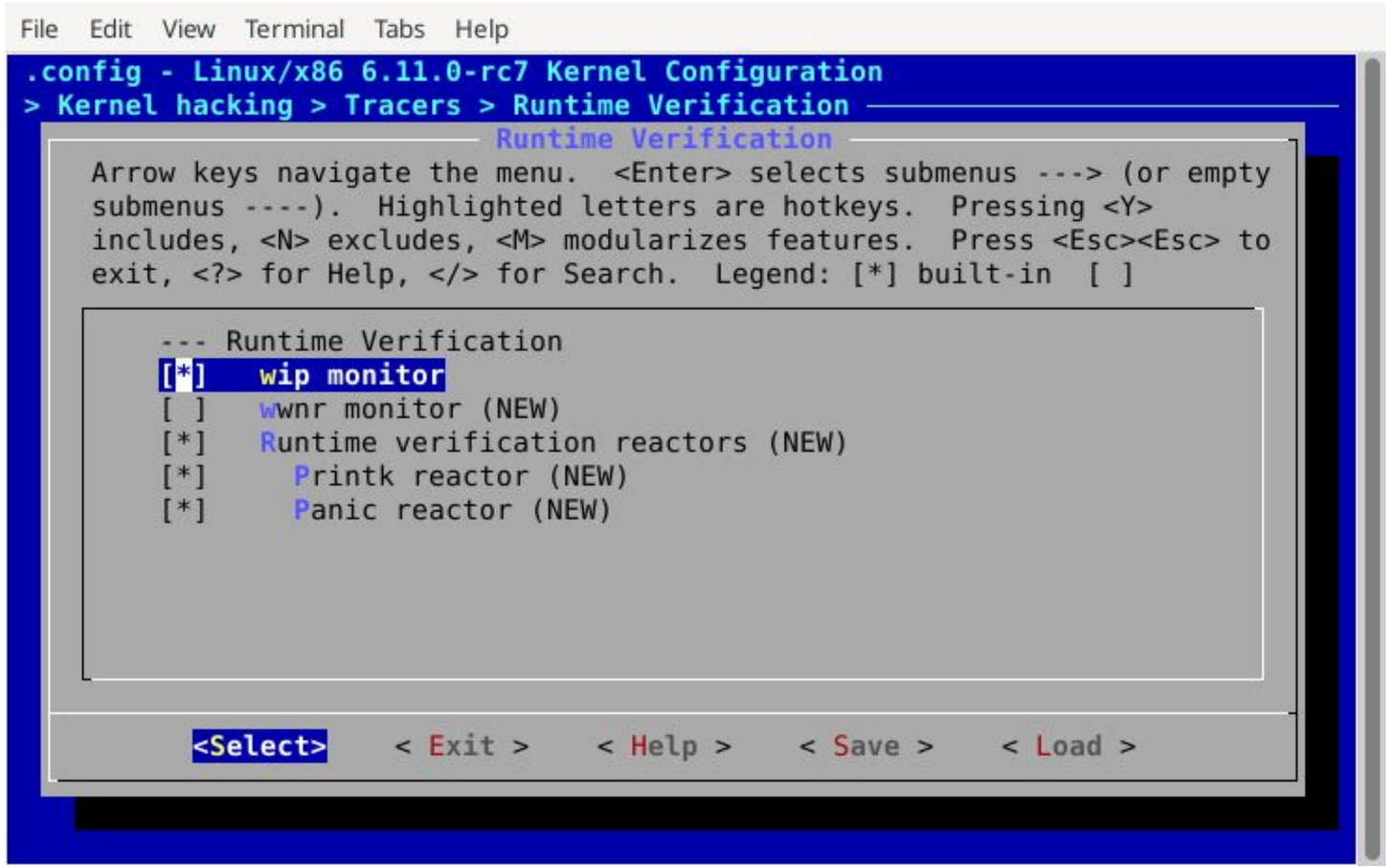
Example

wip.dot

```
$ dot2c wip.dot
enum states {
    preemptive = 0,
    non_preemptive,
    state_max
};

#define INVALID_STATE state_max

enum events {
    preempt_disable = 0,
    preempt_enable,
    sched_waking,
    event_max
};

struct automaton {
    char *state_names[state_max];
    char *event_names[event_max];
    unsigned char function[state_max][event_max];
    unsigned char initial_state;
    bool final_states[state_max];
};
```

```
static const struct automaton aut = {
    .state_names = {
    "preemptive",
    "non_preemptive"
    },
    .event_names = {
    "preempt_disable",
    "preempt_enable",
    "sched_waking"
    },
    .function = {
    { non_preemptive,  INVALID_STATE,  INVALID_STATE },
    {  INVALID_STATE,   preemptive, non_preemptive },
    },
    .initial_state = preemptive,
    .final_states = { 1, 0 },
};
```

Example


File Edit View Terminal Tabs Help
.config - Linux/x86 6.11.0-rc7 Kernel Configuration
> Kernel hacking > Tracers > Runtime Verification
Runtime Verification
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
--- Runtime Verification
[*] wip monitor
[ ] wwnr monitor (NEW)
[*] Runtime verification reactors (NEW)
[*] Printk reactor (NEW)
[*] Panic reactor (NEW)
<Select> < Exit > < Help > < Save > < Load >

Example

Module:

```
static struct rv_monitor rv_wip = {
	.name = "wip",
	.description = "wakeup in preemptive per-cpu testing monitor.",
	.enable = enable_wip,
	.disable = disable_wip,
	.reset = da_monitor_reset_all_wip,
	.enabled = 0,
};

static int __init register_wip(void)
{
	rv_register_monitor(&rv_wip);
	return 0;
}

static void __exit unregister_wip(void)
{
	rv_unregister_monitor(&rv_wip);
}
```

Example

Module:

```
static int enable_wip(void)
{
	int retval;

	retval = da_monitor_init_wip();
	if (retval)
		return retval;

	rv_attach_trace_probe("wip", preempt_enable, handle_preempt_enable);
	rv_attach_trace_probe("wip", sched_waking, handle_sched_waking);
	rv_attach_trace_probe("wip", preempt_disable, handle_preempt_disable);

	return 0;
}

static void disable_wip(void)
{
	rv_wip.enabled = 0;

	rv_detach_trace_probe("wip", preempt_disable, handle_preempt_disable);
	rv_detach_trace_probe("wip", preempt_enable, handle_preempt_enable);
	rv_detach_trace_probe("wip", sched_waking, handle_sched_waking);

	da_monitor_destroy_wip();
}
```

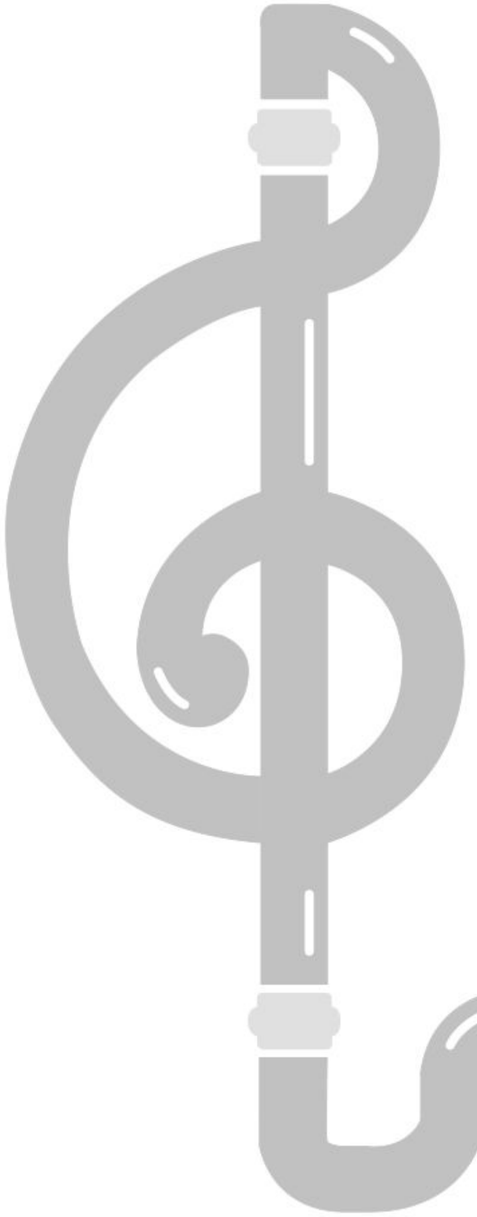
Example

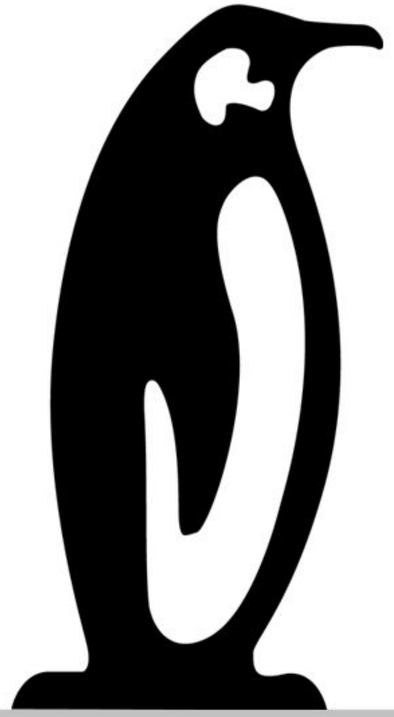
Module:

```
#include "wip.h"

static struct rv_monitor rv_wip;
DECLARE_DA_MON_PER_CPU(wip, unsigned char);

static void handle_preempt_disable(void *data, unsigned long ip, unsigned
long parent_ip)
{
      da_handle_event_wip(preempt_disable_wip);
}

static void handle_preempt_enable(void *data, unsigned long ip, unsigned long
parent_ip)
{
      da_handle_start_event_wip(preempt_enable_wip);
}

static void handle_sched_waking(void *data, struct task_struct *task)
{
      da_handle_event_wip(sched_waking_wip);
}
```

Where to go from here?

- Easier way to create DOT files (GUI tooling?)
- Verify more places in the Kernel (DRM)
- BPF hooks
- What else?





Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024