University of Saarland
Department of Computer Science

Formal Specification and Verification of Functions of the VAMOS Scheduler

Master’s Seminar

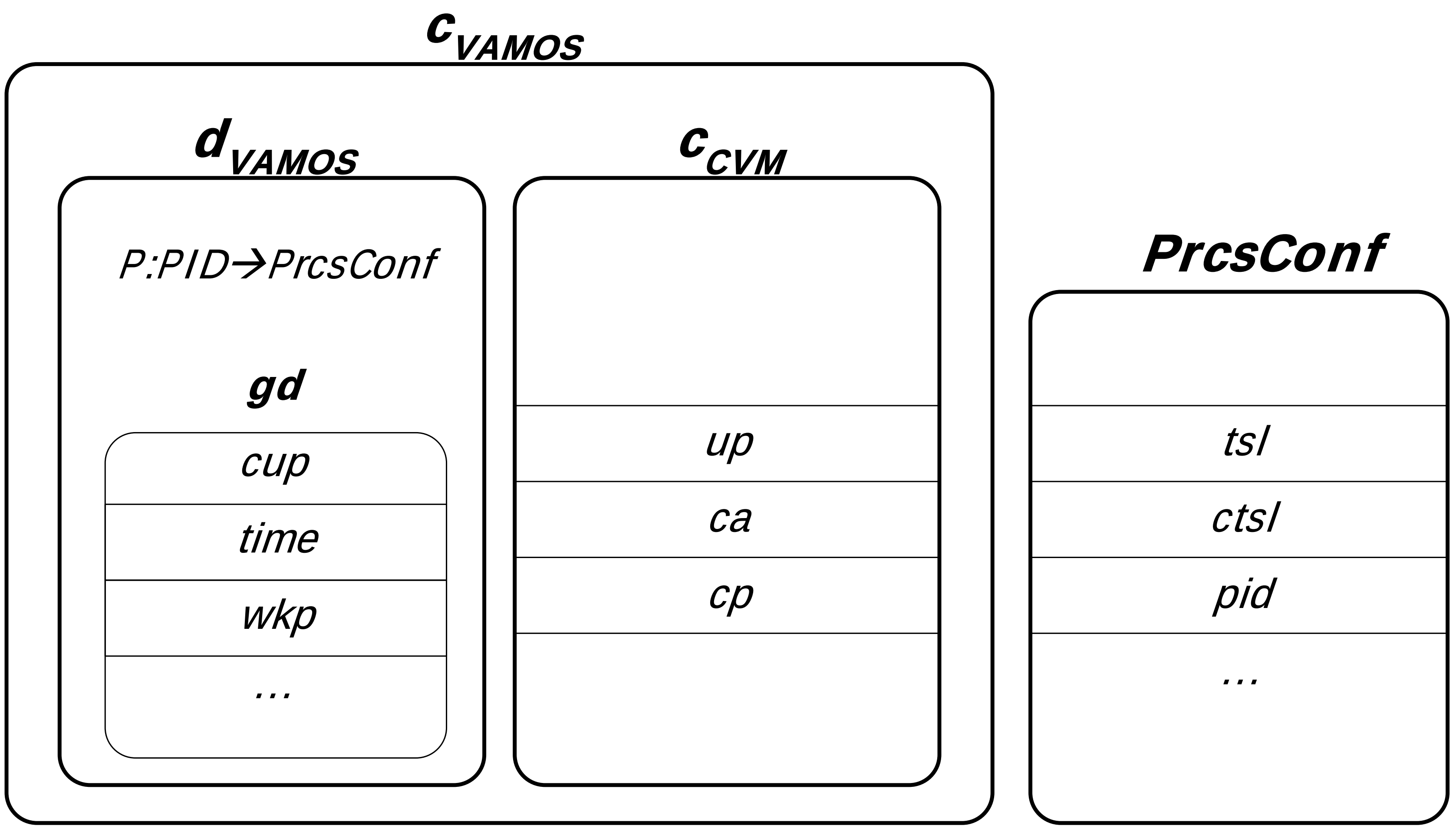
Yury Chebiryak
chebiryak@mail.ru

19.07.2005

Overview

- **VAMOS configuration**
- Assumptions
- Functional Verification
  - Specification
  - Proof Sketch
- Invariants

VAMOS Configuration



VAMOS configuration: gd

$c_{VAMOS}.d_{VAMOS}.gd$

…	
cup : PID	currently running process
rdy : list (list $_{PID}$)	array of ready lists
wkp : list $_{PID}$	list of PIDs, waiting for IPC
inactive : list $_{PID}$	…, not yet activated
current_max_prio : PRIO	current maximum priority,
…	

PRIO={0, …, MAX_PRIO - 1}

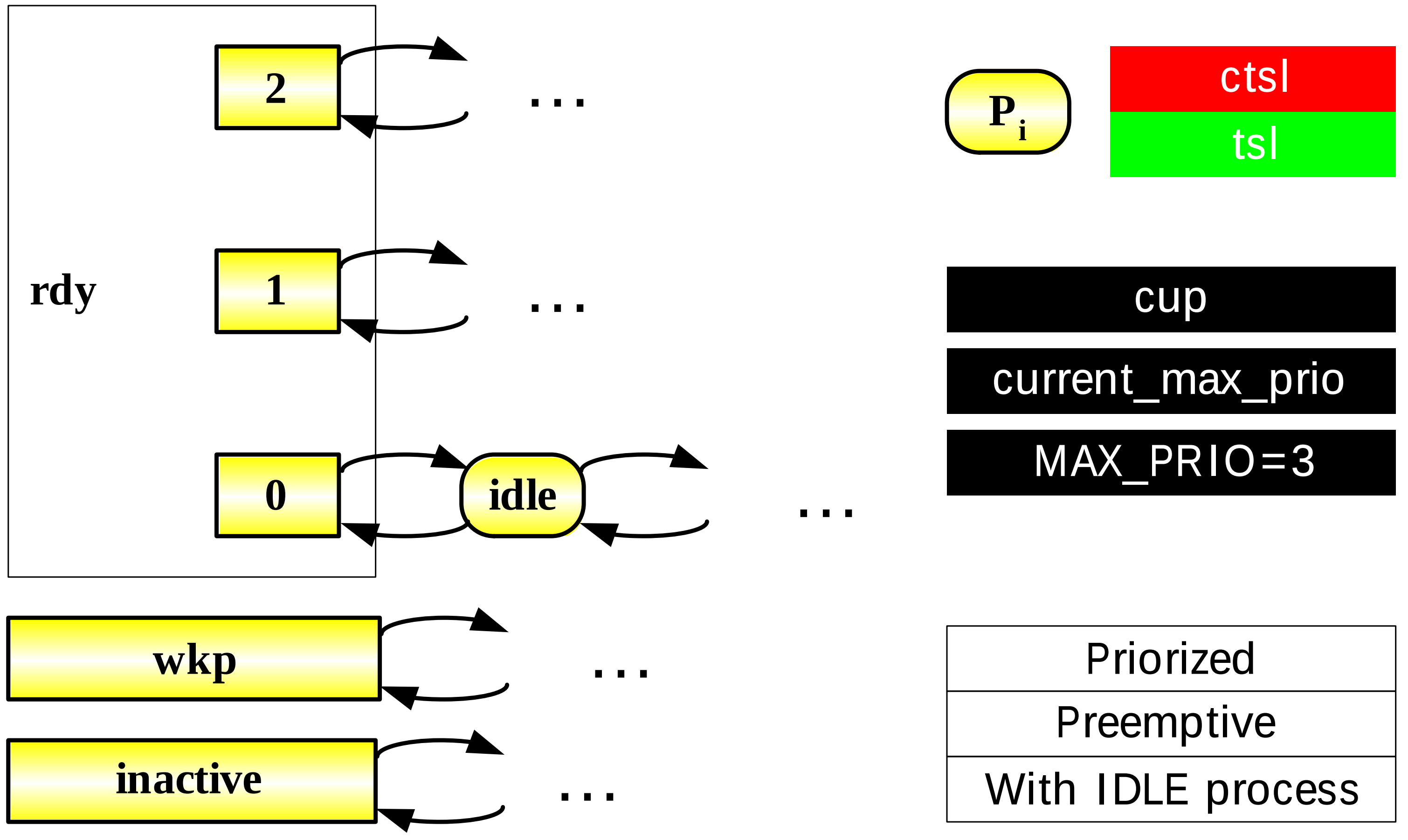
PID={0, …, PID_MAX - 1}

VAMOS configuration: PrcsConf

PrcsConf

...	
pid : PID	process ID
pri : PRIO	priority (higher is better)
state : nat	current status={READY, INACTIVE ...}
tsl : nat	amount of CPU clock ticks to use
ctsl : nat	already consumed timeslice
...	

VAMOS Scheduler



Overview

- VAMOS configuration
- **Assumptions**
- Functional Verification
  - Specification
  - Proof Sketch
- Invariants

vamosData

$$\begin{tikzcd}[column sep=huge, row sep=huge]
c_{VAMOS} \arrow[r, "\textit{global}", "\textit{computation}"'] \arrow[d, phantom, "\rotatebox{90}{$\equiv$}"{description}, "\supseteq \mathit{VamosData}(S_{VAMOS}, d_{VAMOS})"] & c'_{VAMOS} \arrow[d, phantom, "\rotatebox{90}{$\equiv$}"{description}, "\supseteq \mathit{VamosData}(S'_{VAMOS}, d'_{VAMOS})"] \\
S_{VAMOS} \arrow[r, "\textit{concrete}", "\textit{computation}"'] & S'_{VAMOS}
\end{tikzcd}$$

vamosData

- Disjointness of lists of VAMOS

$$
\begin{aligned}
&\forall i \in PID.\\
&\quad (d_{VAMOS}.gd.P(i).state = INACTIVE \leftrightarrow i \in d_{VAMOS}.gd.inactive) \wedge\\
&\quad (d_{VAMOS}.gd.P(i).state = READY \quad \leftrightarrow i \in (d_{VAMOS}.gd.rdy\ [P(i).pri]) \wedge\\
&\quad (d_{VAMOS}.gd.P(i).state \neq INACTIVE \wedge d_{VAMOS}.gd.P(i).state \neq READY\\
&\qquad \leftrightarrow i \in d_{VAMOS}.gd.wkp)
\end{aligned}
$$

- Current process is always ready

$$d_{VAMOS}.gd.P(cup).state = READY$$

vamosData

- Always exist at least one ready process (IDLE)

$$d_{VAMOS}.gd.rdy\ [0] \neq []$$

- Length of ReadyListsArray

$$length(d_{VAMOS}.gd.rdy) = MAX\_PRIO$$

- Priority is in correct range

$$\forall i \in PID.\quad d_{VAMOS}.gd.P(i).pri < MAX\_PRIO$$

vamosData

- Current Maximum Priority (*CMP*) is in correct range

$$d_{VAMOS}.gd.current\_max\_prio < MAX\_PRIO$$

- Ready List indexed by *CMP* is not empty

$$d_{VAMOS}.gd.rdy[d_{VAMOS}.gd.current\_max\_prio] \neq []$$

- Current maximum priority is correct

$$\boxed{\begin{aligned} & cmp\_correct(d_{VAMOS}) = \\ & \quad Max\ \{i \mid i < MAX\_PRIO \wedge d_{VAMOS}.gd.rdy[i] \neq []\} \end{aligned}}$$

Overview

- VAMOS configuration
- Assumptions
- **Functional Verification**
  - **Specification**
  - Proof Sketch
- Invariants

Specification: search_next_process

Signature: $search\_next\_process : D_{VAMOS} \rightarrow D_{VAMOS}$

Assumptions:

$$VamosData(S_{VAMOS}, d_{VAMOS}) \wedge cmp\_correct(d_{VAMOS})$$

Let

$$d'_{VAMOS} = search\_next\_process \;\; d_{VAMOS}$$

Result:

$$cmp\_correct(d'_{VAMOS}) \wedge VamosData(S'_{VAMOS}, d'_{VAMOS}) \wedge$$
$$d'_{VAMOS}.gd'.cup' = d_{VAMOS}.gd.rdy[d_{VAMOS}.gd.current\_max\_prio[0]]$$

Specification: compute_max_prio

Signature: $compute\_max\_prio : D_{VAMOS} \rightarrow nat$

Assumptions: $VamosData(S_{VAMOS}, d_{VAMOS})$

Let

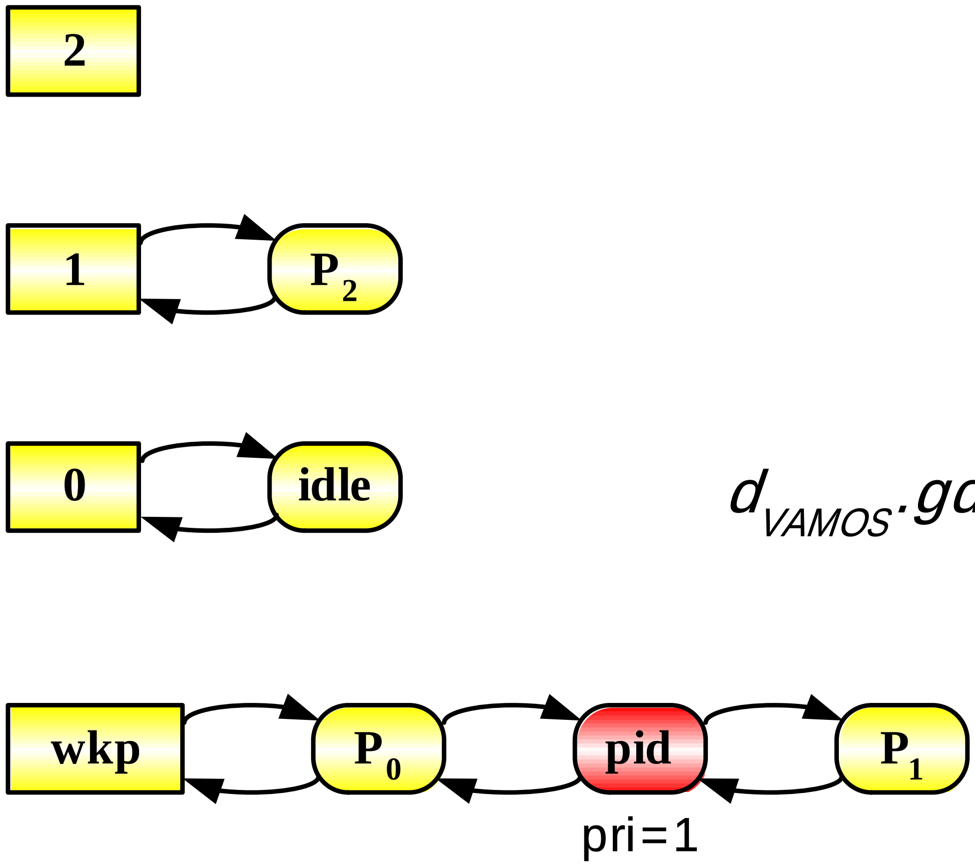
$$res = compute\_max\_prio\ (d_{VAMOS})$$

Result:

$$res = Max\ \{i \mid i < MAX\_PRIO \wedge d_{VAMOS}.gd.rdy[i] \neq []\}$$

Specification: wake_up

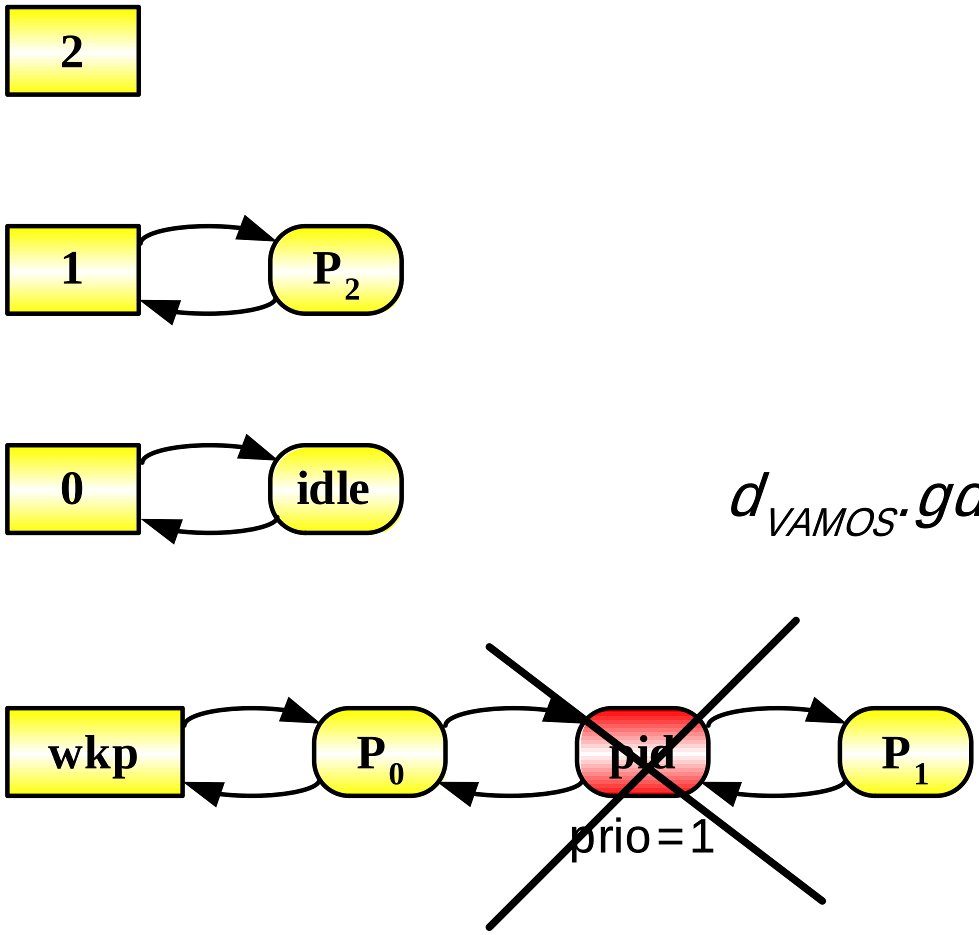
Case P(p).pri ≤ current_max_prio



$d_{VAMOS}.gd.current\_max\_prio = 1$

Specification: wake_up

Case P(p).pri ≤ current_max_prio

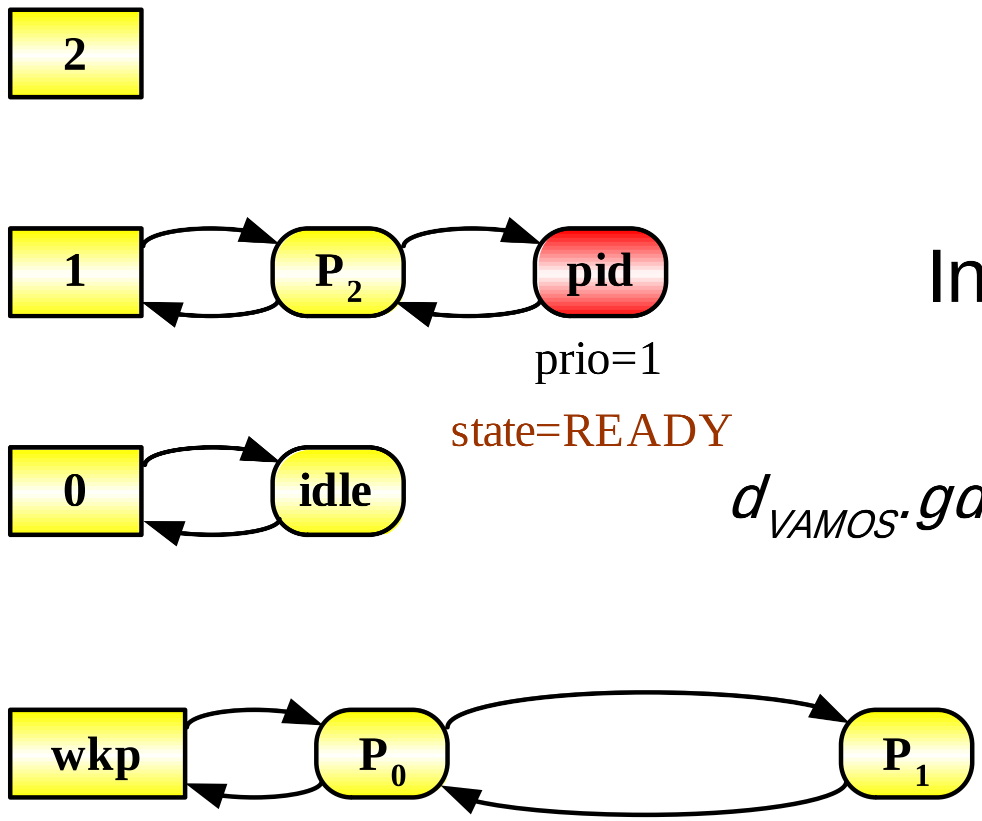


$d_{VAMOS}.gd.current\_max\_prio = 1$

Delete

Specification: wake_up

Case P(p).pri ≤ current_max_prio

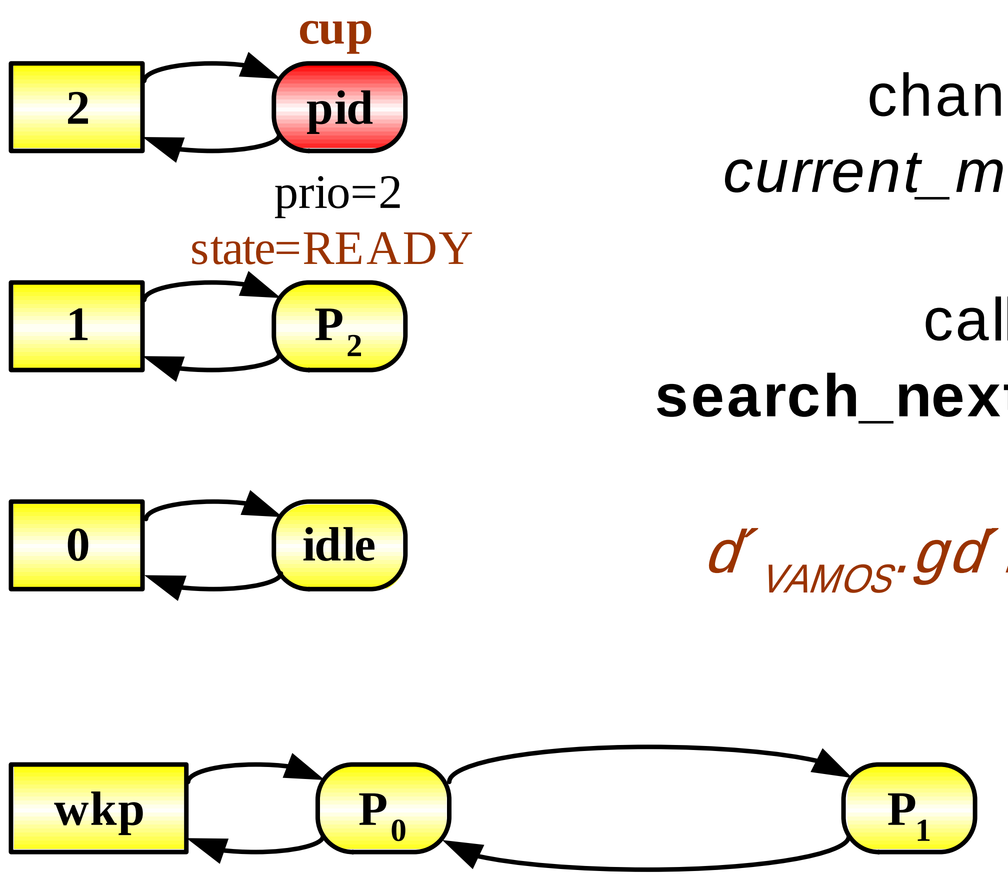


InsertTail

$d_{VAMOS}.gd.current\_max\_prio = 1$

Specification: wake_up

Case P(p).pri > current_max_prio



change
current_max_prio,

call
search_next_process

ď $_{VAMOS}$*·gď .current_max_prió =2*

ď $_{VAMOS}$*·gď .cupʹ = pid*

Specification: wake_up

Signature: $wake\_up : D_{VAMOS} \times PID \rightarrow D_{VAMOS}$

Let $d'_{VAMOS} = wake\_up \;\; d_{VAMOS} \;\; pid$

Assumptions: $pid \in d_{VAMOS}.gd.wkp \wedge$

$VamosData(S_{VAMOS},\; d_{VAMOS}) \wedge$

$cmp\_correct(d_{VAMOS})$

Let

$$pid\_pri = d_{VAMOS}.gd.P(pid).pri$$

Specification: wake_up

Result:

$$d'_{VAMOS}.gd'.wkp' = d_{VAMOS}.gd.wkp \setminus \{pid\}$$

$$\wedge$$

$$d'_{VAMOS}.gd'.rdy' = d_{VAMOS}.gd.rdy\ [$$
$$pid\_pri := (\, d_{VAMOS}.gd.rdy\,[\, pid\_pri\,]\,)\ @(\, pid\,)\ ]$$

$$\wedge$$

$$d'_{VAMOS}.gd'.P'\ (pid).state' = READY$$

$$\wedge$$

$$(\, d_{VAMOS}.gd.current\_max\_prio < pid\_pri \rightarrow$$
$$d'_{VAMOS}.gd'.current\_max\_prio' = pid\_pri \wedge$$
$$d'_{VAMOS}.gd'.cup' = pid\,)$$

$$\wedge$$

$$VamosData(S'_{VAMOS}\,,\, d'_{VAMOS}\,) \wedge cmp\_correct\,(\, d'_{VAMOS}\,)$$

Specification: process_get_mypid

Signature: $process\_get\_mypid : D_{VAMOS} \rightarrow PID$

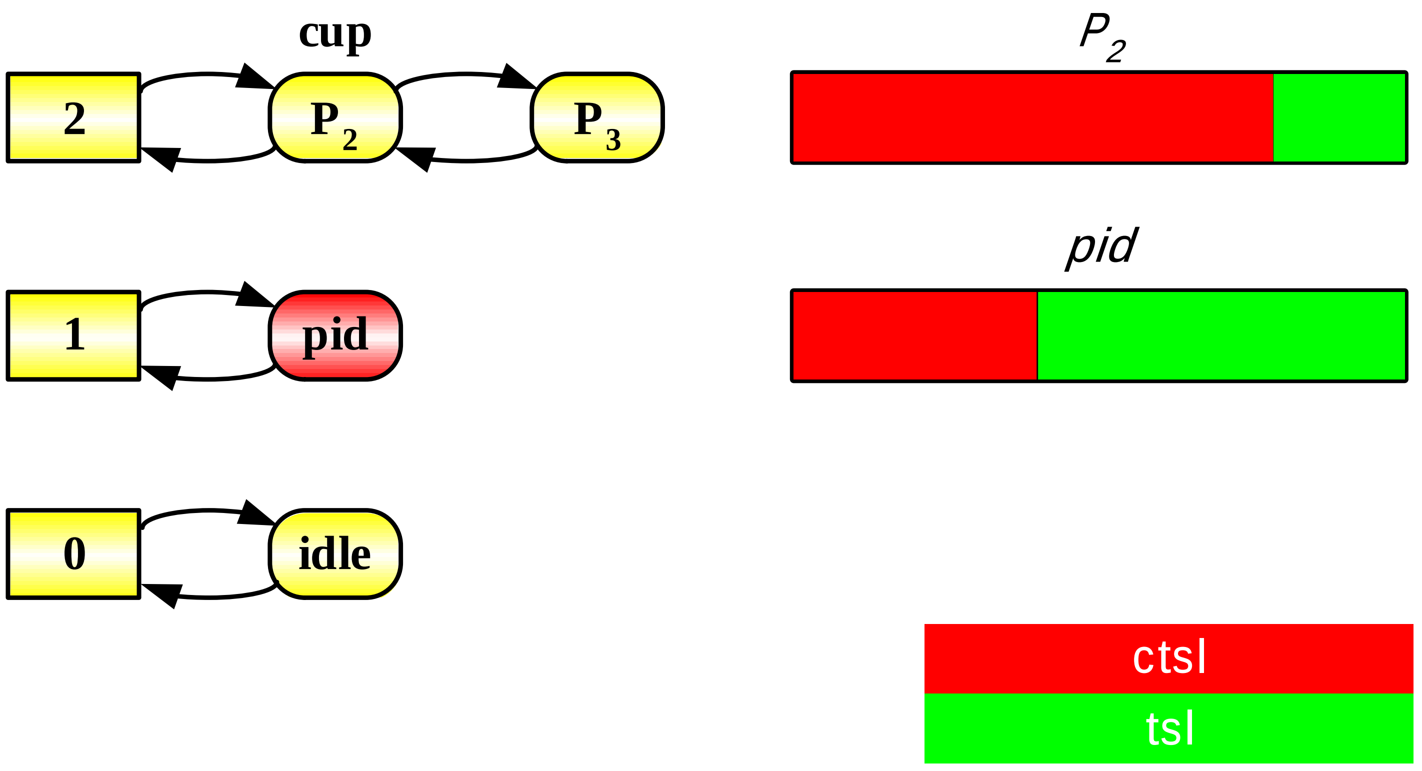
Assumptions:

$$VamosData(S_{VAMOS}, d_{VAMOS}) \wedge cmp\_correct(d_{VAMOS})$$

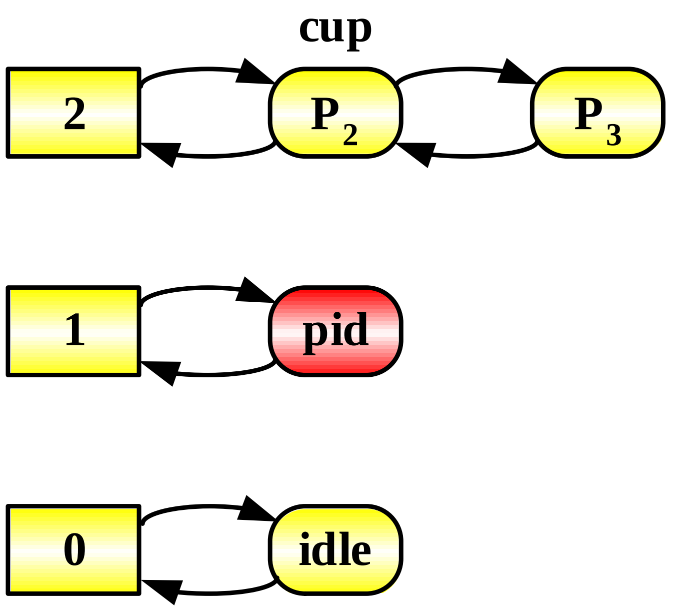
Result:

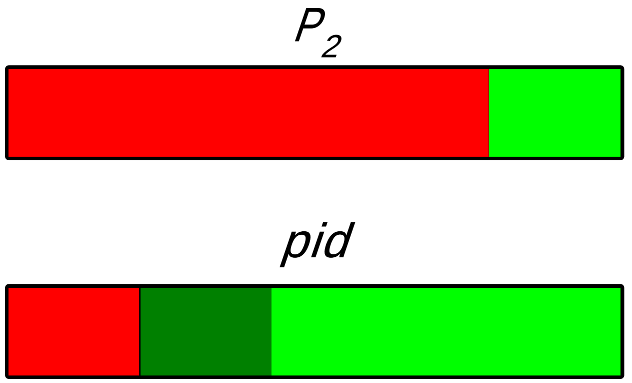
$$process\_get\_mypid(d_{VAMOS}) = d_{VAMOS}.gd.cup$$

Specification: process_switch_to



Specification: process_switch_to

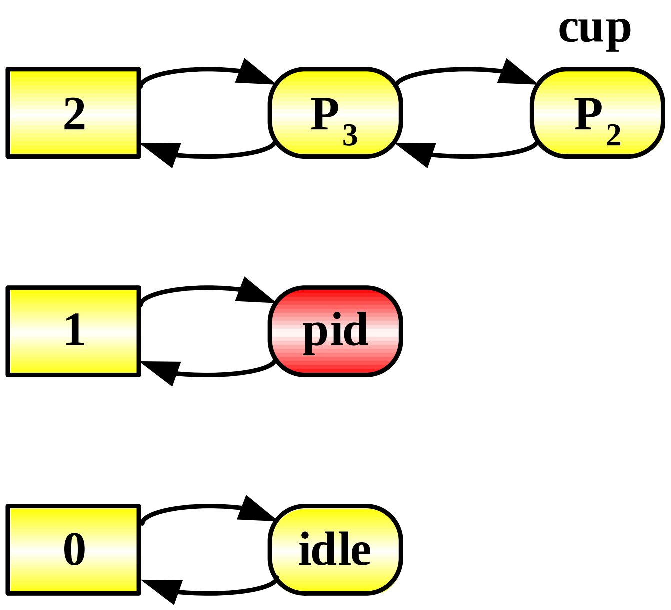


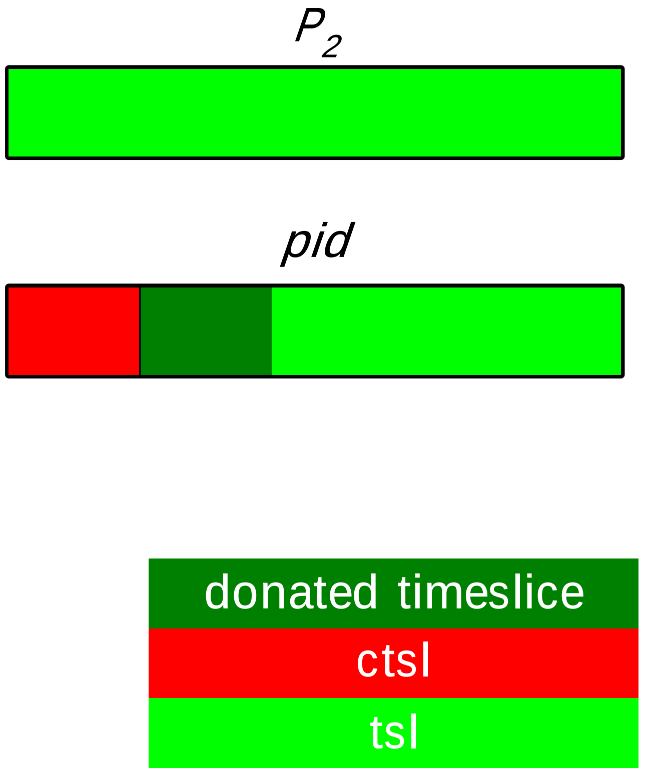


donate timeslice

donated timeslice
ctsl
tsl

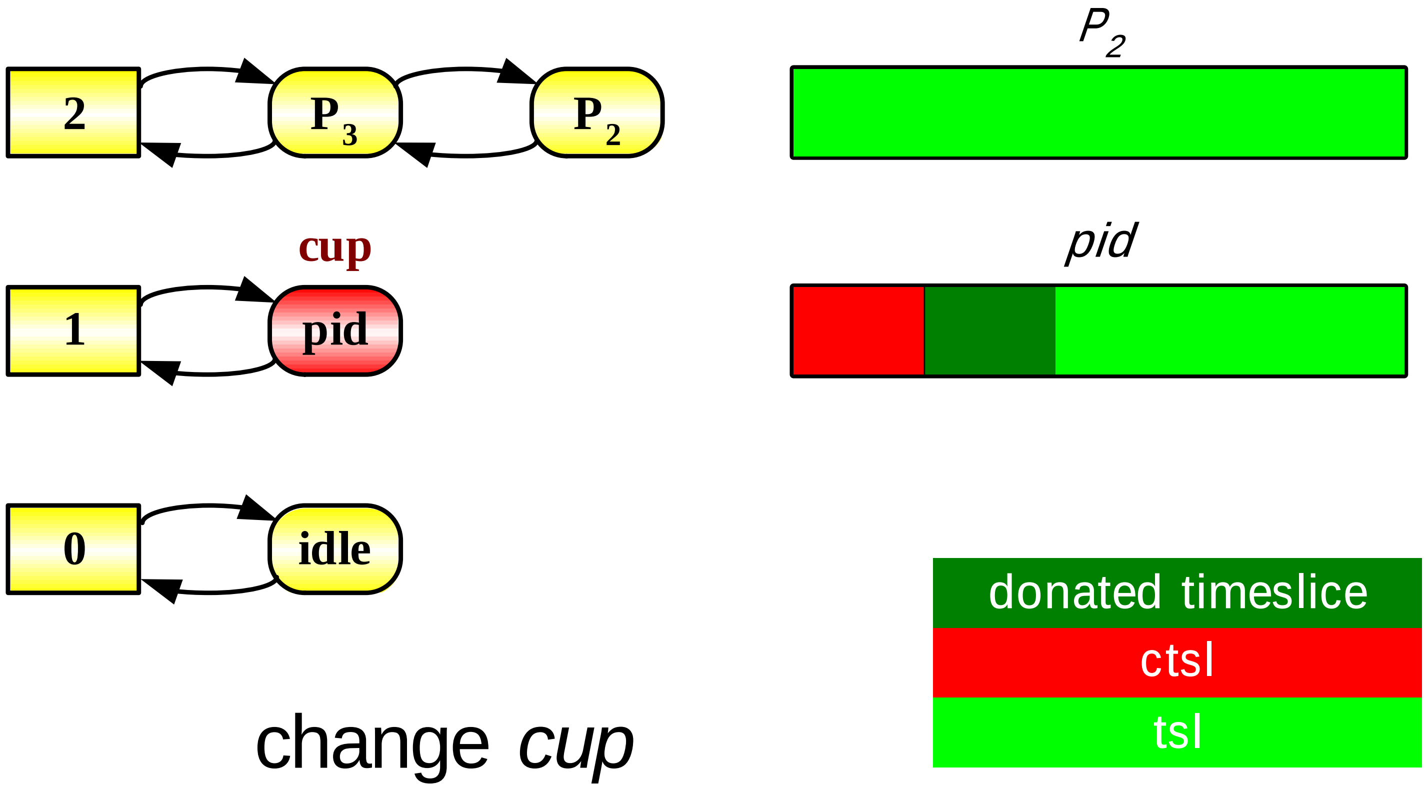
Specification: process_switch_to





Delete, then InsertTail

Specification: process_switch_to



change *cup*

Specification: process_switch_to

Signature: $process\_switch\_to : D_{VAMOS} \times PID \rightarrow D_{VAMOS}$

Assumptions:

$$VamosData(S_{VAMOS}, d_{VAMOS}) \wedge cmp\_correct(d_{VAMOS})$$

Let

$$d'_{VAMOS} = process\_switch\_to \quad d_{VAMOS} \quad pid,$$
$$pid\_is\_valid = (pid \neq 0 \wedge pid < PID\_MAX),$$
$$pid\_is\_ready = (d_{VAMOS}.P(pid).state = READY),$$
$$precond = pid\_is\_valid \wedge pid\_is\_ready$$

Specification: process_switch_to

Abbreviations:

$$
\begin{aligned}
CUP &= d_{VAMOS}.gd.cup, \\
CUP' &= d'_{VAMOS}.gd'.cup', \\
cup\_pid &= d_{VAMOS}.gd.P(CUP).pid, \\
pid\_ctsl &= d_{VAMOS}.gd.P(pid).ctsl, \\
pid\_ctsl' &= d'_{VAMOS}.gd'.P'(pid).ctsl', \\
cup\_tsl &= d_{VAMOS}.gd.P(CUP).tsl, \\
cup\_ctsl &= d_{VAMOS}.gd.P(CUP).ctsl
\end{aligned}
$$

Specification: process_switch_to

Result(1): $precond \wedge (pid \neq CUP) \rightarrow$

$$CUP' = pid \wedge d'_{VAMOS}.gd'.P'(CUP).ctsl' = 0$$

$$\wedge$$

$$d'_{VAMOS}.gd'.rdy'\ [$$

$$cup\_pid := (d_{VAMOS}.gd.rdy \setminus \{CUP\})\ @\ [CUP]\ \ ]$$

$$\wedge$$

$$(pid\_ctsl < (cup\_tsl - cup\_ctsl) \rightarrow pid\_ctsl' = 0)$$

$$\wedge$$

$$(\neg(pid\_ctsl < (cup\_tsl - cup\_ctsl)) \rightarrow$$

$$pid\_ctsl' = pid\_ctsl - (cup\_tsl - cup\_ctsl))$$

$$\wedge$$

$$(VamosData\ (S'_{VAMOS}, d'_{VAMOS}) \wedge cmp\_correct(d'_{VAMOS}\ )$$

Specification: process_switch_to

Result(2):

$$precond \wedge (pid = c_{VAMOS}.d_{VAMOS}.gd.cup) \rightarrow d'_{VAMOS} = d_{VAMOS}$$

Error Result:

$$\overline{precond} \rightarrow d'_{VAMOS} = d_{VAMOS}$$

Specification: prcs_ch_sch_p

Signature:

$$prcs\_ch\_sch\_p : D_{VAMOS} \times pid \times pri \times tsl \rightarrow D_{VAMOS}$$

Assumptions:

$$VamosData(S_{VAMOS}, d_{VAMOS}) \wedge cmp\_correct(d_{VAMOS})$$

Specification: prcs_ch_sch_p

Let

$$\begin{aligned}
d'_{VAMOS} &= prcs\_ch\_sch\_p \quad d_{VAMOS} \quad pid \quad pri \quad tsl, \\
pid\_is\_valid &= (p \neq 0 \wedge pid < PID\_MAX), \\
pid\_is\_inactive &= (d_{VAMOS}.gd.P(pid).state = INACTIVE), \\
pid\_is\_ready &= (d_{VAMOS}.gd.P(pid).state = READY), \\
pri\_is\_valid &= pri < MAX\_PRIO, \\
pid\_tsl &= d_{VAMOS}.gd.P(pid).tsl, \\
pid\_pri &= d_{VAMOS}.gd.P(pid).pri
\end{aligned}$$

Specification: prcs_ch_sch_p

Let

$$precond = pid\_is\_valid \wedge \overline{pid\_is\_inactive} \wedge pri\_is\_valid$$

$$\begin{aligned}
rdy' = d_{VAMOS}.gd.rdy[\\
pid\_pri &:= d_{VAMOS}.gd.rdy[pid\_pri] \setminus \{pid\}, \\
pri &:= d_{VAMOS}.gd.rdy[pri] \; @ \; [pid] \\
&]
\end{aligned}$$

Error Result:

$$\overline{precond} \rightarrow d'_{VAMOS} = d_{VAMOS}$$

Specification: prcs_ch_sch_p

Result:

$$precond \rightarrow (VamosData\ (S'_{VAMOS}, d'_{VAMOS}) \wedge cmp\_correct(d'_{VAMOS}) \wedge$$

$$d'_{VAMOS}.gd'.P'(pid).tsl' = tsl$$

$$\wedge$$

$$d'_{VAMOS}.gd'.P'(pid).pri' = pid\_pri$$

$$\wedge$$

$$(pid\_is\_ready \wedge pri \neq pid\_pri \rightarrow$$

$$d'_{VAMOS}.gd'.rdy' = rdy'$$

$$\wedge$$

$$d'_{VAMOS}.gd'.current\_max\_prio' =$$

$$Max\ \{i \mid \ i < MAX\_PRIO \wedge rdy'[i] \neq [] \ \})$$

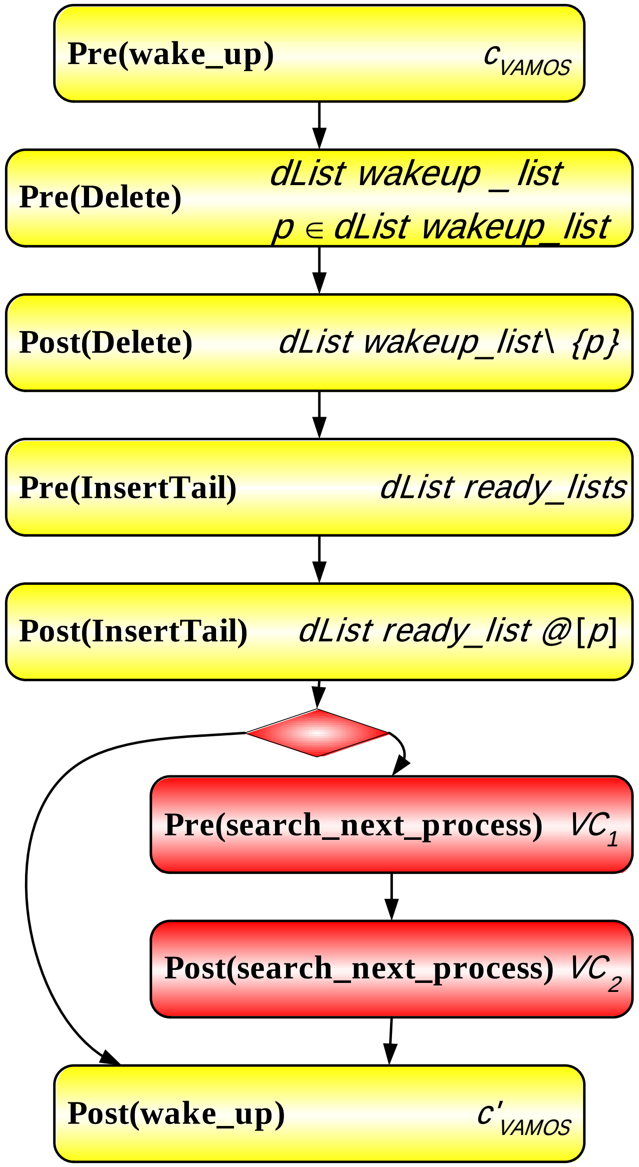
Overview

- VAMOS configuration
- Assumptions
- **Functional Verification**
  - Specification
  - **Proof Sketch**
- Invariants

Proof Sketch: wake_up

```
int wake_up(pib_p p)
{
  pib_p dummy;
  int dummy_int;

wakeup_list =
dlist_pib_t_queue_Delete(wakeup_list,p);

ready_lists[p->priority] =
queue_InsertTail(ready_lists[p->priority],p);
p->state = VAMOS_PROCESS_READY;

if (p->priority > current_max_prio)
    {
      current_max_prio = p->priority;
      dummy_int = search_next_process();
    }

return 0;
}
```

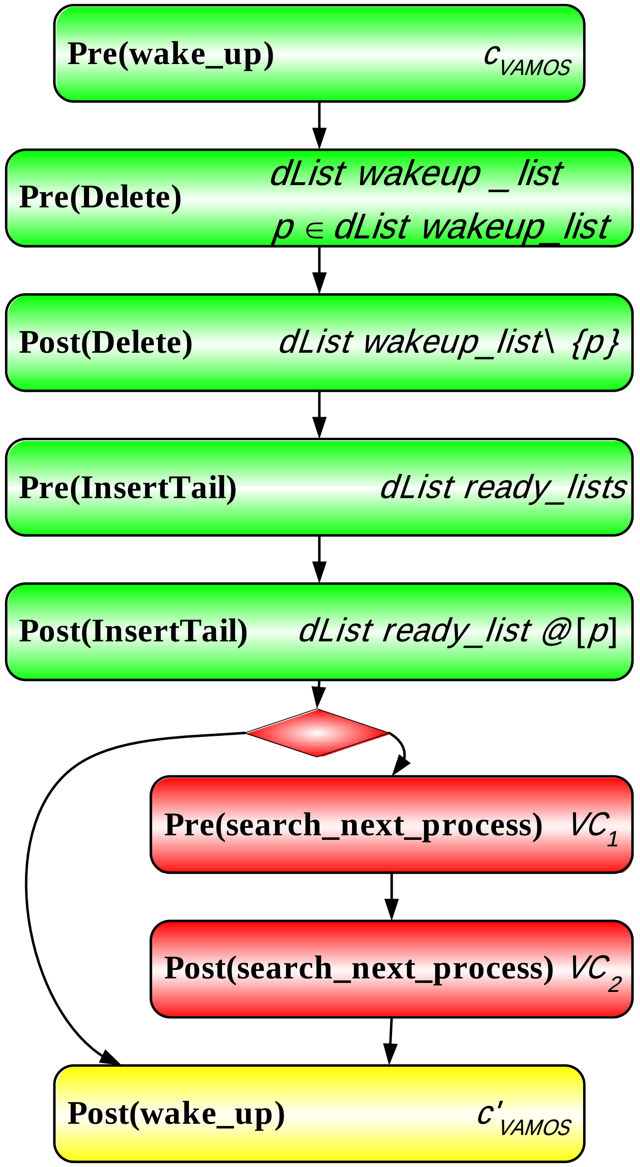
Proof Sketch: wake_up



Pre(wake_up) c_{VAMOS}
Pre(Delete) $dList\ wakeup\_list$ $p \in dList\ wakeup\_list$
Post(Delete) $dList\ wakeup\_list \setminus \{p\}$
Pre(InsertTail) $dList\ ready\_lists$
Post(InsertTail) $dList\ ready\_list\ @[p]$
Pre(search_next_process) VC_1
Post(search_next_process) VC_2
Post(wake_up) c'_{VAMOS}

```
int wake_up(pib_p p)
{
  pib_p dummy;
  int dummy_int;

wakeup_list =
dlist_pib_t_queue_Delete(wakeup_list,p);

ready_lists[p->priority] =
queue_InsertTail(ready_lists[p->priority],p);
p->state = VAMOS_PROCESS_READY;

if (p->priority > current_max_prio)
    {
      current_max_prio = p->priority;
      dummy_int = search_next_process();
    }

return 0;
}
```

Proof Sketch: wake_up


Pre(wake_up) c_{VAMOS}
Pre(Delete) $dList\ wakeup\_list$ $p \in dList\ wakeup\_list$
Post(Delete) $dList\ wakeup\_list \setminus \{p\}$
Pre(InsertTail) $dList\ ready\_lists$
Post(InsertTail) $dList\ ready\_list\ @[p]$
Pre(search_next_process) VC_1
Post(search_next_process) VC_2
Post(wake_up) c'_{VAMOS}

```
int wake_up(pib_p p)
{
  pib_p dummy;
  int dummy_int;

wakeup_list =
dlist_pib_t_queue_Delete(wakeup_list,p);

ready_lists[p->priority] =
queue_InsertTail(ready_lists[p->priority],p);
p->state = VAMOS_PROCESS_READY;

if (p->priority > current_max_prio)
    {
      current_max_prio = p->priority;
      dummy_int = search_next_process();
    }

return 0;
}
```

Overview

- VAMOS configuration
- Assumptions
- Functional Verification
  - Specification
  - Proof Sketch
- **Invariants**

Invariants: Safety properties

- Proceses from *wakeup_list* are not scheduled

- After call of **search_next_process** current process has priority *current_max_prio*

- If process *i* has priority less than *current_max_prio* it can only be scheduled if **switch_to**(*i*) is invoked (not by **handler_clock**)

Invariants: Safety properties

- If during execution of process *i* there was no call of **wake_up** of process with priority higher than *current_max_prio*, then process *i* consumes its whole timeslice (it will not be interrupted)
- If there is only one process with priority *current_max_prio*, then it will run until
  - call of switch_to() or
  - call of process_kill performing suicide or
  - wake_up of process with higher priority.

Invariants: Liveness properties

- Processes with *current_max_prio* share CPU time with respect to their timeslices

Summary

Function call	**Specification**	**Implementation**	**Verification**
InsertTail	+	+	+
Rotate	+	+	+
search_next_process	+	+	+
compute_max_prio	+	+	+
wake_up	+	+	-
process_get_mypid	+	+	+
process_switch_to	+	+	-
prcs_ch_sch_param	+	+	-

Thank you.

Questions?