Proof of the example scheduler

Below we provide a proof outline for the example scheduler. We establish the judgements for `schedule` and `create` in the low-level proof system required by the proof rules of our logic and verify `fork` in the high-level proof system. Note that despite assertions in the proof being long, *all* the steps in it are purely mechanical. In fact, the data structure manipulations involved are of the kind that can be handled by automatic tools based on separation logic[1].

We abbreviate `FORK_FRAME` to f. In the proof of `fork`, F is the local state of the parent, Σ the contents of its stack and P the precondition of the newly created process (excluding a copy of the parent's stack also passed to the child process). In the proof of `load_balance`, the assertion Q describes the local state of the `schedule` function calling it:

$$\begin{aligned}
&(cpu, old_process \Vdash \exists l, \vec{g}, d.\ \mathtt{if} = 0 \wedge d.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto d * d.\mathtt{prev} \mapsto _ * d.\mathtt{next} \mapsto _ *\\
&\ d.\mathtt{timeslice} \mapsto _ * d.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]))\\
&\ [\mathtt{sp} - m - 1 - \mathtt{sizeof(int)}/\mathtt{sp}]
\end{aligned}$$

```
#define FORK_FRAME   sizeof(Process*)
#define SCHED_FRAME  sizeof(Process*)+sizeof(int)

struct Process {
  Process *prev;
  Process *next;
  word kernel_stack[StackSize];
  word *saved_sp;
  int timeslice;
};

Lock *runqueue_lock[NCPUS];
Process *runqueue[NCPUS];
Process *current[NCPUS];

void schedule() {
```

$$\{\mathsf{SchedState}_k\}$$

```
  int cpu;
  Process *old_process;
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \exists l, \vec{g}, d.\ \mathtt{if} = 0 \wedge d.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto d * d.\mathtt{prev} \mapsto _ * d.\mathtt{next} \mapsto _ *\\
&\ d.\mathtt{timeslice} \mapsto _ * d.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}])\}
\end{aligned}$$

```
  savecpuid(&cpu);
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \exists l, \vec{g}, d.\ \mathtt{if} = 0 \wedge d.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto d * d.\mathtt{prev} \mapsto _ * d.\mathtt{next} \mapsto _ *\\
&\ d.\mathtt{timeslice} \mapsto _ * d.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}])\}
\end{aligned}$$

```
  load_balance(cpu);
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \exists l, \vec{g}, d.\ \mathtt{if} = 0 \wedge d.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto d * d.\mathtt{prev} \mapsto _ * d.\mathtt{next} \mapsto _ *\\
&\ d.\mathtt{timeslice} \mapsto _ * d.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}])\}
\end{aligned}$$

```
  old_process = current[cpu];
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \exists l, \vec{g}.\ \mathtt{if} = 0 \wedge\\
&\ old_process.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto old_process *\\
&\ old_process.\mathtt{prev} \mapsto _ * old_process.\mathtt{next} \mapsto _ *\\
&\ old_process.\mathtt{timeslice} \mapsto _ * old_process.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}])\}
\end{aligned}$$

... *// update the timeslice of old_process*

```
  if (old_process->timeslice) {
    // We deallocate local variables here
```

$$\{\mathsf{SchedState}_k\}$$

```
    iret();
  }
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \exists l, \vec{g}.\ \mathtt{if} = 0 \wedge\\
&\ old_process.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto old_process *\\
&\ old_process.\mathtt{prev} \mapsto _ * old_process.\mathtt{next} \mapsto _ *\\
&\ old_process.\mathtt{timeslice} \mapsto _ * old_process.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}])\}
\end{aligned}$$

```
  old_process->timeslice = SCHED_QUANTUM;
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \exists l, \vec{g}.\ \mathtt{if} = 0 \wedge\\
&\ old_process.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto old_process *\\
&\ old_process.\mathtt{prev} \mapsto _ * old_process.\mathtt{next} \mapsto _ *\\
&\ old_process.\mathtt{timeslice} \mapsto _ * old_process.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}])\}
\end{aligned}$$

```
  lock(runqueue_lock[cpu]);
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \mathsf{locked}(\mathsf{runqueue_lock}[k]) * \exists l, \vec{g}.\ \mathtt{if} = 0 \wedge\\
&\ old_process.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto old_process *\\
&\ old_process.\mathtt{prev} \mapsto _ * old_process.\mathtt{next} \mapsto _ *\\
&\ old_process.\mathtt{timeslice} \mapsto _ * old_process.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *\\
&\ \exists x, y, z.\ \mathtt{runqueue}[k] \mapsto z *\\
&\ \mathsf{desc}_0(z, _) * z.\mathtt{prev} \mapsto y * z.\mathtt{next} \mapsto x * \mathsf{dll}_\Lambda(x, z, z, y)\}
\end{aligned}$$

```
  insert_node_after(runqueue[cpu]->prev, old_process);
```

$$\begin{aligned}
&\{cpu, old_process \Vdash \mathsf{locked}(\mathsf{runqueue_lock}[k]) * \exists l, \vec{g}.\ \mathtt{if} = 0 \wedge\\
&\ old_process.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto old_process *\\
&\ old_process.\mathtt{prev} \mapsto y * old_process.\mathtt{next} \mapsto z *\\
&\ old_process.\mathtt{timeslice} \mapsto _ * old_process.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *\\
&\ \exists x, y, z.\ \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, _) *\\
&\ z.\mathtt{prev} \mapsto old_process * z.\mathtt{next} \mapsto x * \mathsf{dll}_\Lambda(x, z, old_process, y)\}
\end{aligned}$$

```
  current[cpu] = runqueue[cpu]->next;
```

$$\begin{aligned}
&\{(cpu, old_process \Vdash \mathsf{locked}(\mathsf{runqueue_lock}[k]) * \exists l, \vec{g}.\ \mathtt{if} = 0 \wedge\\
&\ old_process.\mathtt{kernel_stack} = \mathtt{ss} \wedge\\
&\ cpu = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge\\
&\ \mathtt{current}[k] \mapsto old_process *\\
&\ old_process.\mathtt{prev} \mapsto z * old_process.\mathtt{next} \mapsto z *\\
&\ old_process.\mathtt{timeslice} \mapsto _ * old_process.\mathtt{saved_sp} \mapsto _ *\\
&\ (\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *\\
&\ \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto _ *\\
&\ \mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *\\
&\ \exists x, y, z.\ \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, _) *\\
&\ z.\mathtt{prev} \mapsto old_process * z.\mathtt{next} \mapsto old_process)
\end{aligned}$$

[1] For example: H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. W. O'Hearn. Scalable shape analysis for systems code. In *CAV'08: Conference on Computer-Aided Verification*, volume 5123 of *LNCS*, pages 385–398. Springer, 2008.

$\vee$
$(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) * \exists l, \vec{g}.\, \mathtt{if} = 0 \wedge$
$\mathit{old\_process}.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$\mathit{cpu} = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto x *$
$\mathit{old\_process}.\mathtt{prev} \mapsto y * \mathit{old\_process}.\mathtt{next} \mapsto z *$
$\mathit{old\_process}.\mathtt{timeslice} \mapsto \_ * \mathit{old\_process}.\mathtt{saved\_sp} \mapsto \_ *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *$
$\exists x, y, z, w, \gamma.\, \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, \_) *$
$z.\mathtt{prev} \mapsto \mathit{old\_process} * z.\mathtt{next} \mapsto x * \mathsf{desc}_0(x, \gamma) * \mathsf{Process}(\gamma) *$
$x.\mathtt{prev} \mapsto z * x.\mathtt{next} \mapsto w * \mathsf{dll}_\Lambda(w, x, \mathit{old\_process}, y))\}$
`remove_node(current[cpu]);`
$\{(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) * \exists l, \vec{g}.\, \mathtt{if} = 0 \wedge$
$\mathit{old\_process}.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$\mathit{cpu} = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto \mathit{old\_process} *$
$\mathit{old\_process}.\mathtt{prev} \mapsto \_ * \mathit{old\_process}.\mathtt{next} \mapsto \_ *$
$\mathit{old\_process}.\mathtt{timeslice} \mapsto \_ * \mathit{old\_process}.\mathtt{saved\_sp} \mapsto \_ *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *$
$\exists x, y, z.\, \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, \_) *$
$z.\mathtt{prev} \mapsto z * z.\mathtt{next} \mapsto z)$
$\vee$
$(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) * \exists l, \vec{g}.\, \mathtt{if} = 0 \wedge$
$\mathit{old\_process}.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$\mathit{cpu} = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto x *$
$\mathit{old\_process}.\mathtt{prev} \mapsto y * \mathit{old\_process}.\mathtt{next} \mapsto z *$
$\mathit{old\_process}.\mathtt{timeslice} \mapsto \_ * \mathit{old\_process}.\mathtt{saved\_sp} \mapsto \_ *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *$
$\exists x, y, z, w, \gamma.\, \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, \_) *$
$z.\mathtt{prev} \mapsto \mathit{old\_process} * z.\mathtt{next} \mapsto w * \mathsf{desc}_0(x, \gamma) * \mathsf{Process}(\gamma) *$
$x.\mathtt{prev} \mapsto \_ * x.\mathtt{next} \mapsto \_ * \mathsf{dll}_\Lambda(w, z, \mathit{old\_process}, y))\}$
`old_process->saved_sp = _sp;`
$\{(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) * \exists l, \vec{g}.\, \mathtt{if} = 0 \wedge$
$\mathit{old\_process}.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$\mathit{cpu} = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto \mathit{old\_process} *$
$\mathit{old\_process}.\mathtt{prev} \mapsto \_ * \mathit{old\_process}.\mathtt{next} \mapsto \_ *$
$\mathit{old\_process}.\mathtt{timeslice} \mapsto \_ * \mathit{old\_process}.\mathtt{saved\_sp} \mapsto \mathtt{sp} *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *$
$\exists x, y, z.\, \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, \_) *$
$z.\mathtt{prev} \mapsto z * z.\mathtt{next} \mapsto z)$
$\vee$
$(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) * \exists l, \vec{g}.\, \mathtt{if} = 0 \wedge$
$\mathit{old\_process}.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$\mathit{cpu} = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto x *$
$\mathit{old\_process}.\mathtt{prev} \mapsto y * \mathit{old\_process}.\mathtt{next} \mapsto z *$
$\mathit{old\_process}.\mathtt{timeslice} \mapsto \_ * \mathit{old\_process}.\mathtt{saved\_sp} \mapsto \mathtt{sp} *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *$
$\exists x, y, z, w, \gamma.\, \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, \_) *$
$z.\mathtt{prev} \mapsto \mathit{old\_process} * z.\mathtt{next} \mapsto w * \mathsf{desc}_0(x, \gamma) * \mathsf{Process}(\gamma) *$
$x.\mathtt{prev} \mapsto \_ * x.\mathtt{next} \mapsto \_ * \mathsf{dll}_\Lambda(w, z, \mathit{old\_process}, y))\}$
`_sp = current[cpu]->saved_sp;`
$\{(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) * \exists l, \vec{g}.\, \mathtt{if} = 0 \wedge$
$\mathit{old\_process}.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$\mathit{cpu} = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto \mathit{old\_process} *$
$\mathit{old\_process}.\mathtt{prev} \mapsto \_ * \mathit{old\_process}.\mathtt{next} \mapsto \_ *$
$\mathit{old\_process}.\mathtt{timeslice} \mapsto \_ * \mathit{old\_process}.\mathtt{saved\_sp} \mapsto \mathtt{sp} *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *$
$\exists x, y, z.\, \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, \_) *$
$z.\mathtt{prev} \mapsto z * z.\mathtt{next} \mapsto z)$
$\vee$
$(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) *$
$\exists \mathit{old\_process}', \gamma'.\, \mathtt{if} = 0 \wedge \mathtt{current}[k] \mapsto x *$
$\mathit{old\_process}'.\mathtt{prev} \mapsto y * \mathit{old\_process}'.\mathtt{next} \mapsto z *$
$\mathsf{desc}_0(\mathit{old\_process}', \gamma') * \mathsf{Process}(\gamma') *$
$\exists x, y, z, w, \gamma.\, \mathtt{runqueue}[k] \mapsto z * \mathsf{desc}_0(z, \_) *$
$z.\mathtt{prev} \mapsto \mathit{old\_process}' * z.\mathtt{next} \mapsto w * \mathsf{Process}(\gamma) *$
$x.\mathtt{prev} \mapsto \_ * x.\mathtt{next} \mapsto \_ * \mathsf{dll}_\Lambda(w, z, \mathit{old\_process}', y) *$
$(\gamma(\mathbf{if}) = 1 \wedge \gamma(\mathbf{ss}) = x.\mathtt{kernel\_stack} \wedge \gamma(\mathbf{sp}) = \mathtt{sp} - m - s - 1 \wedge$
$0 \leq \gamma(\mathbf{sp}) - \gamma(\mathbf{ss}) \leq \mathtt{StackBound} \wedge x.\mathtt{timeslice} \mapsto \_ *$
$x.\mathtt{saved\_sp} \mapsto (\gamma(\mathbf{sp}) + m + 1 + s) *$
$\gamma(\mathbf{sp})..(\gamma(\mathbf{sp}) + m) \mapsto \gamma(\mathbf{ip})\gamma(\vec{\mathbf{gr}}) *$
$(\gamma(\mathbf{sp}) + m + s + 1)..(\gamma(\mathbf{ss}) + \mathtt{StackSize} - 1) \mapsto \_))\}$
`savecpuid(&cpu);`
`_ss = current[cpu]->kernel_stack;`
$\{(\mathit{cpu}, \mathit{old\_process} \Vdash \mathsf{locked}(\mathtt{runqueue\_lock}[k]) *$
$\exists l, \vec{g}, d.\, \mathtt{if} = 0 \wedge d.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$\mathit{cpu} = k \wedge 0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto d *$
$d.\mathtt{prev} \mapsto \_ * d.\mathtt{next} \mapsto \_ *$
$d.\mathtt{timeslice} \mapsto \_ * d.\mathtt{saved\_sp} \mapsto \mathtt{sp} *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}]) *$
$\exists x, y, z.\, \mathtt{runqueue}[k] \mapsto z *$
$\mathsf{desc}_0(z, \_) * z.\mathtt{prev} \mapsto y * z.\mathtt{next} \mapsto x * \mathsf{dll}_\Lambda(x, z, z, y)\}$
`unlock(runqueue_lock[cpu]);`
$\{\mathit{cpu}, \mathit{old\_process} \Vdash \exists l, \vec{g}, d.\, \mathtt{if} = 0 \wedge d.\mathtt{kernel\_stack} = \mathtt{ss} \wedge$
$0 \leq \mathtt{sp} - \mathtt{ss} - m - s - 1 \leq \mathtt{StackBound} \wedge$
$\mathtt{current}[k] \mapsto d * d.\mathtt{prev} \mapsto \_ * d.\mathtt{next} \mapsto \_ *$
$d.\mathtt{timeslice} \mapsto \_ * d.\mathtt{saved\_sp} \mapsto \_ *$
$(\mathtt{sp} - s - m - 1)..(\mathtt{sp} - s - 1) \mapsto l\vec{g} *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ *$
$\mathsf{Process}([\mathbf{ip} : l, \mathbf{if} : 1, \mathbf{ss} : \mathtt{ss}, \mathbf{sp} : \mathtt{sp} - s - m - 1, \vec{\mathbf{gr}} : \vec{g}])\}$
`// We deallocate local variables here`
$\{\mathsf{SchedState}_k\}$
`iret();`
`}`

`void load_balance(int cpu) {`
$\{\mathit{cpu} \Vdash 0 \leq \mathit{cpu} < \mathtt{NCPUS} \wedge Q * \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_\}$
`int cpu2, non_empty;`
`Process *proc;`
$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \leq \mathit{cpu} < \mathtt{NCPUS} \wedge$
$Q[\mathtt{sp} - 2 \cdot \mathtt{sizeof}(\mathtt{int}) - \mathtt{sizeof}(\mathtt{Process*})/\mathtt{sp}] *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_\}$
`lock(runqueue_lock[cpu]);`
$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \leq \mathit{cpu} < \mathtt{NCPUS} \wedge$
$Q[\mathtt{sp} - 2 \cdot \mathtt{sizeof}(\mathtt{int}) - \mathtt{sizeof}(\mathtt{Process*})/\mathtt{sp}] *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ * \mathsf{locked}(\mathtt{runqueue\_lock}[\mathit{cpu}]) *$
$\exists x, y, z.\, \mathtt{runqueue}[\mathit{cpu}] \mapsto z *$
$\mathsf{desc}_0(z, \_) * z.\mathtt{prev} \mapsto y * z.\mathtt{next} \mapsto x * \mathsf{dll}_\Lambda(x, z, z, y)\}$
`non_empty = (runqueue[cpu]->next != runqueue[cpu]);`
$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \leq \mathit{cpu} < \mathtt{NCPUS} \wedge$
$Q[\mathtt{sp} - 2 \cdot \mathtt{sizeof}(\mathtt{int}) - \mathtt{sizeof}(\mathtt{Process*})/\mathtt{sp}] *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ * \mathsf{locked}(\mathtt{runqueue\_lock}[\mathit{cpu}]) *$
$\exists x, y, z.\, \mathtt{runqueue}[\mathit{cpu}] \mapsto z *$
$\mathsf{desc}_0(z, \_) * z.\mathtt{prev} \mapsto y * z.\mathtt{next} \mapsto x * \mathsf{dll}_\Lambda(x, z, z, y)\}$
`unlock(runqueue_lock[cpu]);`
$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \leq \mathit{cpu} < \mathtt{NCPUS} \wedge$
$Q[\mathtt{sp} - 2 \cdot \mathtt{sizeof}(\mathtt{int}) - \mathtt{sizeof}(\mathtt{Process*})/\mathtt{sp}] *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_\}$
`if (non_empty || random(0, 1)) {`
$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash$
$Q[\mathtt{sp} - 2 \cdot \mathtt{sizeof}(\mathtt{int}) - \mathtt{sizeof}(\mathtt{Process*})/\mathtt{sp}] *$
$\mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_\}$
`// We deallocate local variables here`
$\{\mathit{cpu} \Vdash Q * \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_\}$
`return;`

```
}
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \le \mathit{cpu} < \texttt{NCPUS} \wedge$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_\}$

```
do { cpu2 = random(0, NCPUS-1); } while (cpu == cpu2);
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash$
$0 \le \mathit{cpu}, \mathit{cpu2} < \texttt{NCPUS} \wedge \mathit{cpu} \ne \mathit{cpu2} \wedge$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_\}$

```
if (cpu < cpu2) {
  lock(runqueue_lock[cpu]); lock(runqueue_lock[cpu2]);
} else {
  lock(runqueue_lock[cpu2]); lock(runqueue_lock[cpu]);
}
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \le \mathit{cpu}, \mathit{cpu2} < \texttt{NCPUS} \wedge$
$\texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu}]) * \texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu2}]) *$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ * \exists x, y, z, x', y', z', w.$
$\texttt{runqueue}[\mathit{cpu}] \mapsto z * \texttt{runqueue}[\mathit{cpu2}] \mapsto z' *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto x * \texttt{dll}_\Lambda(x, z, z, y)$
$\texttt{desc}_0(z', \_) * z'.\texttt{prev} \mapsto y' * z'.\texttt{next} \mapsto x' * \texttt{dll}_\Lambda(x', z', z', y')\}$

```
if (runqueue[cpu2]->next != runqueue[cpu2]) {
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \le \mathit{cpu}, \mathit{cpu2} < \texttt{NCPUS} \wedge$
$\texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu}]) * \texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu2}]) *$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ * \exists x, y, z, x', y', z', w.$
$\texttt{runqueue}[\mathit{cpu}] \mapsto z * \texttt{runqueue}[\mathit{cpu2}] \mapsto z' *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto x * \texttt{dll}_\Lambda(x, z, z, y)$
$\texttt{desc}_0(z', \_) * z'.\texttt{prev} \mapsto y' * z'.\texttt{next} \mapsto x' *$
$x'.\texttt{prev} \mapsto z' * x'.\texttt{next} \mapsto w *$
$(\exists \gamma.\, \texttt{desc}_0(x', \gamma) * \texttt{Process}(\gamma)) * \texttt{dll}_\Lambda(w, x', z', y')\}$

```
  proc = runqueue[cpu2]->next;
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \le \mathit{cpu}, \mathit{cpu2} < \texttt{NCPUS} \wedge$
$\texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu}]) * \texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu2}]) *$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ *$
$\exists x, y, z, y', z', w.\, \texttt{runqueue}[\mathit{cpu}] \mapsto z * \texttt{runqueue}[\mathit{cpu2}] \mapsto z' *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto x * \texttt{dll}_\Lambda(x, z, z, y)$
$\texttt{desc}_0(z', \_) * z'.\texttt{prev} \mapsto y' * z'.\texttt{next} \mapsto \mathit{proc} *$
$\mathit{proc}.\texttt{prev} \mapsto z' * \mathit{proc}.\texttt{next} \mapsto w *$
$(\exists \gamma.\, \texttt{desc}_0(\mathit{proc}, \gamma) * \texttt{Process}(\gamma)) * \texttt{dll}_\Lambda(w, \mathit{proc}, z', y')\}$

```
  remove_node(proc);
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \le \mathit{cpu}, \mathit{cpu2} < \texttt{NCPUS} \wedge$
$\texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu}]) * \texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu2}]) *$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ *$
$\exists x, y, z, y', z', w.\, \texttt{runqueue}[\mathit{cpu}] \mapsto z * \texttt{runqueue}[\mathit{cpu2}] \mapsto z' *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto x * \texttt{dll}_\Lambda(x, z, z, y)$
$\texttt{desc}_0(z', \_) * z'.\texttt{prev} \mapsto y' * z'.\texttt{next} \mapsto w *$
$\mathit{proc}.\texttt{prev} \mapsto z' * \mathit{proc}.\texttt{next} \mapsto w *$
$(\exists \gamma.\, \texttt{desc}_0(\mathit{proc}, \gamma) * \texttt{Process}(\gamma)) * \texttt{dll}_\Lambda(w, z', z', y')\}$

```
  insert_node_after(runqueue[cpu], proc);
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \le \mathit{cpu}, \mathit{cpu2} < \texttt{NCPUS} \wedge$
$\texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu}]) * \texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu2}]) *$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ *$
$\exists x, y, z, y', z', w.\, \texttt{runqueue}[\mathit{cpu}] \mapsto z * \texttt{runqueue}[\mathit{cpu2}] \mapsto z' *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto \mathit{proc} * \texttt{dll}_\Lambda(x, \mathit{proc}, z, y)$
$\texttt{desc}_0(z', \_) * z'.\texttt{prev} \mapsto y' * z'.\texttt{next} \mapsto w *$
$\mathit{proc}.\texttt{prev} \mapsto z * \mathit{proc}.\texttt{next} \mapsto x *$
$(\exists \gamma.\, \texttt{desc}_0(\mathit{proc}, \gamma) * \texttt{Process}(\gamma)) * \texttt{dll}_\Lambda(w, z', z', y')\}$

```
}
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash 0 \le \mathit{cpu}, \mathit{cpu2} < \texttt{NCPUS} \wedge$
$\texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu}]) * \texttt{locked}(\texttt{runqueue\_lock}[\mathit{cpu2}]) *$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ *$
$\exists x, y, z, x', y', z'.\, \texttt{runqueue}[\mathit{cpu}] \mapsto z * \texttt{runqueue}[\mathit{cpu2}] \mapsto z' *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto x * \texttt{dll}_\Lambda(x, z, z, y)$
$\texttt{desc}_0(z', \_) * z'.\texttt{prev} \mapsto y' * z'.\texttt{next} \mapsto x' * \texttt{dll}_\Lambda(x', z', z', y')\}$

```
unlock(runqueue_lock[cpu]);
unlock(runqueue_lock[cpu2]);
```

$\{\mathit{cpu}, \mathit{cpu2}, \mathit{non\_empty}, \mathit{proc} \Vdash$
$Q[\texttt{sp} - 2 \cdot \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_\}$

```
  // We deallocate local variables here
```

$\{\mathit{cpu} \Vdash Q * \texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_\}$

```
}

_regparam void create(Process *new_process) {
  // Here we move the parameter from gr1 into
  // the new_process local variable
```

$\{\mathit{new\_process} \Vdash \exists \gamma.\, \gamma(\mathbf{if}) = 1 \wedge$
$\texttt{SchedState}_k[\texttt{sp} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{desc}(\mathit{new\_process}, \gamma) * \texttt{Process}(\gamma)\}$

```
  int cpu;
```

$\{\mathit{new\_process}, \mathit{cpu} \Vdash \exists \gamma.\, \gamma(\mathbf{if}) = 1 \wedge$
$\texttt{SchedState}_k[\texttt{sp} - \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\texttt{desc}(\mathit{new\_process}, \gamma) * \texttt{Process}(\gamma)\}$

```
  savecpuid(&cpu);
```

$\{\mathit{new\_process}, \mathit{cpu} \Vdash \exists \gamma.\, \mathit{cpu} = k \wedge \gamma(\mathbf{if}) = 1 \wedge$
$\texttt{SchedState}_k[\texttt{sp} - \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\mathit{new\_process}.\texttt{prev} \mapsto \_ * \mathit{new\_process}.\texttt{next} \mapsto \_ *$
$\texttt{desc}_0(\mathit{new\_process}, \gamma) * \texttt{Process}(\gamma)\}$

```
  new_process->timeslice = SCHED_QUANTUM;
```

$\{\mathit{new\_process}, \mathit{cpu} \Vdash \exists \gamma.\, \mathit{cpu} = k \wedge \gamma(\mathbf{if}) = 1 \wedge$
$\texttt{SchedState}_k[\texttt{sp} - \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\mathit{new\_process}.\texttt{prev} \mapsto \_ * \mathit{new\_process}.\texttt{next} \mapsto \_ *$
$\texttt{desc}_0(\mathit{new\_process}, \gamma) * \texttt{Process}(\gamma)\}$

```
  lock(runqueue_lock[cpu]);
```

$\{\mathit{new\_process}, \mathit{cpu} \Vdash \exists \gamma.\, \mathit{cpu} = k \wedge \gamma(\mathbf{if}) = 1 \wedge$
$\texttt{SchedState}_k[\texttt{sp} - \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\mathit{new\_process}.\texttt{prev} \mapsto \_ * \mathit{new\_process}.\texttt{next} \mapsto \_ *$
$\texttt{desc}_0(\mathit{new\_process}, \gamma) * \texttt{Process}(\gamma) * \exists x, y, z.\, \texttt{runqueue}[k] \mapsto z *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto x * \texttt{dll}_\Lambda(x, z, z, y) *$
$\texttt{locked}(\texttt{runqueue\_lock}[k])\}$

```
  insert_node_after(runqueue[cpu], new_process);
```

$\{\mathit{new\_process}, \mathit{cpu} \Vdash \exists \gamma.\, \mathit{cpu} = k \wedge \gamma(\mathbf{if}) = 1 \wedge$
$\texttt{SchedState}_k[\texttt{sp} - \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\mathit{new\_process}.\texttt{prev} \mapsto z * \mathit{new\_process}.\texttt{next} \mapsto x *$
$\texttt{desc}_0(\mathit{new\_process}, \gamma) * \texttt{Process}(\gamma) * \exists x, y, z.\, \texttt{runqueue}[k] \mapsto z *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto \mathit{new\_process} *$
$\texttt{dll}_\Lambda(x, \mathit{new\_process}, z, y)\}$

$\{\mathit{new\_process}, \mathit{cpu} \Vdash \mathit{cpu} = k \wedge$
$\texttt{SchedState}_k[\texttt{sp} - \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}] *$
$\exists x, y, z.\, \texttt{runqueue}[k] \mapsto z *$
$\texttt{desc}_0(z, \_) * z.\texttt{prev} \mapsto y * z.\texttt{next} \mapsto \mathit{new\_process} *$
$\texttt{dll}_\Lambda(\mathit{new\_process}, z, z, y) * \texttt{locked}(\texttt{runqueue\_lock}[k])\}$

```
  unlock(runqueue_lock[cpu]);
```

$\{\mathit{new\_process}, \mathit{cpu} \Vdash \texttt{SchedState}_k$
$[\texttt{sp} - \texttt{sizeof(int)} - \texttt{sizeof(Process*)}/\texttt{sp}]\}$

```
  // We deallocate local variables here
```

$\{\texttt{SchedState}_k\}$

```
  iret();
}

int fork() {
```

$\{0 \le \texttt{sp} - \texttt{ss} \le \texttt{StackBound} - f \wedge$
$\texttt{ss}..(\texttt{sp} - 1) \mapsto \Sigma 0 l \vec{g} * \texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ * F * P\}$

```
  Process *new_process;
```

$\{\mathit{new\_process} \Vdash 0 \le \texttt{sp} - \texttt{ss} \le \texttt{StackBound} \wedge$
$\texttt{ss}..(\texttt{sp} - f - 1) \mapsto \Sigma 0 l \vec{g} * \texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ * F * P\}$

```
  new_process = alloc(sizeof(Process));
```

$\{\mathit{new\_process} \Vdash 0 \le \texttt{sp} - \texttt{ss} \le \texttt{StackBound} \wedge$
$\texttt{ss}..(\texttt{sp} - f - 1) \mapsto \Sigma 0 l \vec{g} * \texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ * F * P *$
$\mathit{new\_process}.\texttt{prev} \mapsto \_ * \mathit{new\_process}.\texttt{next} \mapsto \_ *$
$\mathit{new\_process}.\texttt{timeslice} \mapsto \_ * \mathit{new\_process}.\texttt{saved\_sp} \mapsto \_ *$
$\mathit{new\_process}.\texttt{kernel\_stack}..$
$(\mathit{new\_process}.\texttt{kernel\_stack} + \texttt{StackSize} - 1) \mapsto \_\}$

```
  memcpy(new_process->kernel_stack, _ss, StackSize);
```

$\{\mathit{new\_process} \Vdash 0 \le \texttt{sp} - \texttt{ss} \le \texttt{StackBound} \wedge$
$\texttt{ss}..(\texttt{sp} - f - 1) \mapsto \Sigma 0 l \vec{g} * \texttt{sp}..(\texttt{ss} + \texttt{StackSize} - 1) \mapsto \_ * F * P *$
$\mathit{new\_process}.\texttt{prev} \mapsto \_ * \mathit{new\_process}.\texttt{next} \mapsto \_ *$

$new\_process.\mathtt{timeslice} \mapsto \_ * new\_process.\mathtt{saved\_sp} \mapsto \_ *$
$new\_process.\mathtt{kernel\_stack}..$
$(new\_process.\mathtt{kernel\_stack} + \mathtt{sp} - \mathtt{ss} - f - 1) \mapsto \Sigma 0 l \vec{g} *$
$(new\_process.\mathtt{kernel\_stack} + \mathtt{sp} - \mathtt{ss} - f)..$
$(new\_process.\mathtt{kernel\_stack} + \mathtt{StackSize} - 1) \mapsto \_\}$

```
  new_process->saved_sp = new_process->kernel_stack+
      _sp-_ss-FORK_FRAME+SCHED_FRAME;
```

$\{new\_process \Vdash 0 \leq \mathtt{sp} - \mathtt{ss} \leq \mathtt{StackBound} \wedge$
$\mathtt{ss}..(\mathtt{sp} - f - 1) \mapsto \Sigma 0 l \vec{g} * \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ * F * P *$
$new\_process.\mathtt{prev} \mapsto \_ * new\_process.\mathtt{next} \mapsto \_ *$
$new\_process.\mathtt{timeslice} \mapsto \_ *$
$new\_process.\mathtt{saved\_sp} \mapsto$
$new\_process.\mathtt{kernel\_stack} + \mathtt{sp} - \mathtt{ss} - f + s *$
$new\_process.\mathtt{kernel\_stack}..$
$(new\_process.\mathtt{kernel\_stack} + \mathtt{sp} - \mathtt{ss} - f - 1) \mapsto \Sigma 0 l \vec{g} *$
$(new\_process.\mathtt{kernel\_stack} + \mathtt{sp} - \mathtt{ss} - f)..$
$(new\_process.\mathtt{kernel\_stack} + \mathtt{StackSize} - 1) \mapsto \_\}$

$\{new\_process \Vdash 0 \leq \mathtt{sp} - \mathtt{ss} \leq \mathtt{StackBound} \wedge$
$\mathtt{ss}..(\mathtt{sp} - f - 1) \mapsto \Sigma 0 l \vec{g} * \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ * F * P *$
$\exists \gamma.\, \gamma(\mathbf{ip}) = l \wedge \gamma(\vec{\mathbf{gr}}) = \vec{g} \wedge \gamma(\mathbf{ss}) = new\_process.\mathtt{kernel\_stack} \wedge$
$\gamma(\mathbf{sp}) = new\_process.\mathtt{kernel\_stack} + \mathtt{sp} - \mathtt{ss} - f - m - 1 \wedge$
$\gamma(\mathbf{if}) = 1 \wedge \mathsf{desc}(new\_process, \gamma) * \gamma(\mathbf{ss})..(\gamma(\mathbf{sp}) - 1) \mapsto \Sigma 0\}$

```
  // We assume P satisfies the premiss of the Create rule
  _icall create(new_process);
```

$\{new\_process \Vdash 0 \leq \mathtt{sp} - \mathtt{ss} \leq \mathtt{StackBound} \wedge$
$\mathtt{ss}..(\mathtt{sp} - f - 1) \mapsto \Sigma 0 l \vec{g} * \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ * F\}$

```
  // We deallocate local variables here
```

$\{0 \leq \mathtt{sp} - \mathtt{ss} - f \leq \mathtt{StackBound} \wedge$
$\mathtt{ss}..(\mathtt{sp} - 1) \mapsto \Sigma 0 l \vec{g} * \mathtt{sp}..(\mathtt{ss} + \mathtt{StackSize} - 1) \mapsto \_ * F\}$

```
  return 1;
}
```