



Facultad de Ciencias Exactas, Ingeniería y Agrimensura
UNIVERSIDAD NACIONAL DE ROSARIO

Ingeniería de Software

Trabajo Práctico Verificación de Software

Cipullo, Inés

2025

1 Requerimientos

Se describen los requerimientos de un planificador de procesos a corto plazo, encargado de planificar los procesos que están listos para ejecución, también llamado *dispatcher*. Este planificador implementará el algoritmo de planificación “Ronda” (*Round Robin* en inglés), y este sistema tendrá un único procesador.

Un proceso puede estar en alguno de los siguientes estados: *nuevo*, *listo*, *ejecutando*, *bloqueado*, *terminado*. El dispatcher decide entre los procesos que están listos para ejecutarse y determina a cuál de ellos *activar*, y detiene a aquellos que *exceden su tiempo* de procesador, es decir, se encarga de las transiciones entre los estados **listo** y **ejecutando**.

Siguiendo Round Robin, los procesos listos se almacenan en forma de cola, cada proceso listo se ejecuta por un sólo *quantum* y si un proceso no ha terminado de ejecutarse al final de su período, será interrumpido y puesto al final de la cola de procesos listos. Se deben tener en cuenta, también, aquellas transiciones que involucran otros estados de procesos pero inciden sobre alguno de los dos estados que se controlan desde el dispatcher. Los procesos que sean agregados a la cola de listos por estas otras transiciones, se ubicarán al final de la misma.

2 Especificación

Para empezar, se dan las siguientes designaciones.

p es un proceso $\approx p \in PROCESS$

t es un contador de ticks del sistema $\approx t \in TICK$

proceso nulo $\approx nullp$

cantidad de tiempo durante el cual un proceso tiene permiso para ejecutarse en el procesador antes de ser interrumpido $\approx quantum$

cola de procesos listos para ser ejecutados $\approx procQueue$

proceso en ejecución $\approx current$

contador de ticks restantes de ejecución que le corresponden a $current \approx remTicks$

Luego, se introducen los tipos que se utilizan en la especificación.

$[PROCESS]$

$TICK == \mathbb{N}$

Además, se presentan las siguientes definiciones axiomáticas, donde se define la existencia del proceso $nullp$, que representa el proceso nulo, y la constante $quantum$, que equivale a 5 ticks.

$| \quad nullp : PROCESS$

$| \quad quantum : TICK$

$| \quad \hline quantum = 5$

Se define entonces el espacio de estados del planificador y su estado inicial.

$Dispatcher$

$procQueue : seq\langle PROCESS \rangle$

$current : PROCESS$

$remTicks : TICK$

<i>InitDispatcher</i>	_____
<i>Dispatcher</i>	_____
<i>procQueue</i> = $\langle \rangle$	
<i>current</i> = <i>nullp</i>	
<i>remTicks</i> = 0	

Como un proceso no puede estar *en ejecución* y *listo* al mismo tiempo, se plantea el siguiente invarinate de estado:

<i>InvDispatcher</i>	_____
<i>Dispatcher</i>	_____
$\neg \langle \textit{nullp} \rangle \text{ in } \textit{procQueue}$	
$\neg \langle \textit{current} \rangle \text{ in } \textit{procQueue}$	

Procedemos con la especificación de las operaciones requeridas. Estas son:

- *NewProcess*: para pasar un proceso de estado *nuevo* (o *bloqueado*) a *listo*.
- *Dispatch*: modela el funcionamiento del planificador, se encarga de las transiciones entre los estados *listo* y *en ejecución*.
- *TerminateProcess*: para pasar un proceso de estado *en ejecución* a *terminado*.

NewProcessOk

$\Delta Dispatcher$

$p? : PROCESS$

$p? \notin \text{ran } procQueue$

$p? \neq current$

$p? \neq nullp$

$procQueue' = procQueue \frown \langle p? \rangle$

$current' = current$

$remTicks' = remTicks$

NewProcessError

$\Xi Dispatcher$

$p? : PROCESS$

$p? \in \text{ran } procQueue \vee p? = current \vee p? = nullp$

$NewProcess == NewProcessOk \vee NewProcessError$

Tick

$\Delta Dispatcher$

$remTicks > 0$

$current \neq nullp$

$procQueue' = procQueue$

$current' = current$

$remTicks' = remTicks - 1$

Timeout

$\Delta Dispatcher$

$remTicks = 0$

$current \neq nullp$

$procQueue' = procQueue \frown \langle current \rangle$

$current' = nullp$

$remTicks' = remTicks$

DispatchProcess

$\Delta Dispatcher$

$current = nullp$

$procQueue \neq \langle \rangle$

$procQueue' = tail\ procQueue$

$current' = head\ procQueue$

$remTicks' = remTicks$

Idle

$\Xi Dispatcher$

$current = nullp$

$procQueue = \langle \rangle$

$Dispatch == Tick \vee Timeout \vee DispatchProcess \vee Idle$

<i>TerminateProcessOk</i>	
$\Delta Dispatcher$	
$p? : PROCESS$	
$current = p?$	
$procQueue' = procQueue$	
$current' = nullp$	
$remTicks' = remTicks$	
<i>TerminateProcessError</i>	
$\Xi Dispatcher$	
$p? : PROCESS$	
$current \neq p?$	

$$TerminateProcess == TerminateProcessOk \vee TerminateProcessError$$

3 Simulaciones en $\{log\}$

Se traduce la especificación a $\{log\}$ (en **planificador.slog**). A continuación, se presentan dos simulaciones ejecutadas sobre $\{log\}$ y la primera solución de cada una.

1ra simulación

```

initDispatcher(PQ0, C0, RT0)           &
newProcess(PQ0, C0, RT0, p1, PQ1, C1, RT1) &
dispatch(PQ1, C1, RT1, PQ2, C2, RT2)   &
dispatch(PQ2, C2, RT2, PQ3, C3, RT3)   &
dispatch(PQ3, C3, RT3, PQ4, C4, RT4)   &

```

```
dispatch(PQ4, C4, RT4, PQ5, C5, RT5)      &
dispatch(PQ5, C5, RT5, PQ6, C6, RT6)      &
dispatch(PQ6, C6, RT6, PQ7, C7, RT7)      &
dispatch(PQ7, C7, RT7, PQ8, C8, RT8) .
```

La idea de esta simulación es encolar un nuevo proceso **p1** en la cola de procesos listos, luego iniciar su ejecución hasta que termine el tiempo de ejecución asignado (1 **quantum**) y sea interrumpido por el dispatcher. El resultado es el esperado:

```
PQ0 = {},
C0 = nullp,
RT0 = 0,
PQ1 = {[1,p1]},
C1 = nullp,
RT1 = 0,
PQ2 = {},
C2 = p1,
RT2 = 5,
PQ3 = {},
C3 = p1,
RT3 = 4,
PQ4 = {},
C4 = p1,
RT4 = 3,
PQ5 = {},
C5 = p1,
RT5 = 2,
PQ6 = {},
C6 = p1,
RT6 = 1,
```



```

PQ7 = {},
C7 = p1,
RT7 = 0,
PQ8 = {[1,p1]},
C8 = nullptr,
RT8 = 0

```

2da simulación

```

initDispatcher(PQ0, C0, RT0)           &
newProcess(PQ0, C0, RT0, p1, PQ1, C1, RT1) &
newProcess(PQ1, C1, RT1, p2, PQ2, C2, RT2) &
dispatch(PQ2, C2, RT2, PQ3, C3, RT3)   &
dispatch(PQ3, C3, RT3, PQ4, C4, RT4)   &
terminateProc(PQ4, C4, RT4, p1, PQ5, C5, RT5) &
dispatch(PQ5, C5, RT5, PQ6, C6, RT6).

```

La idea de esta simulación es encolar dos nuevos procesos, **p1** y **p2**, en la cola de procesos listos, luego iniciar la ejecución del primero, que ejecute por un tick y que termine su ejecución. Después se inicia la ejecución del segundo. El resultado es el esperado:

```

PQ0 = {},
C0 = nullptr,
RT0 = 0,
PQ1 = {[1,p1]},
C1 = nullptr,
RT1 = 0,
PQ2 = {[2,p2],[1,p1]},
C2 = nullptr,
RT2 = 0,

```

```

PQ3 = {[1,p2]},
C3 = p1,
RT3 = 5,
PQ4 = {[1,p2]},
C4 = p1,
RT4 = 4,
PQ5 = {[1,p2]},
C5 = nullp,
RT5 = 0,
PQ6 = {},
C6 = p2,
RT6 = 5

```

4 Generado de Condiciones de Verificación (WIP)

Se utiliza el VCG para generar condiciones de verificación sobre la especificación en *{log}*, lo cual genera el archivo **planificador-vc.slog**. Luego el comando `check_vcs_planificador` realiza todas las descargas de prueba automáticamente.

Sobre la interacción con el VCG

Fue necesario agregar la hipótesis `invSeq(ProcQueue)` en la condición de verificación `dispatch_pi_invDispatcher`. Sin esa hipótesis se presentaba un contraejemplo donde `ProcQueue = {[1,C_],[_N2,C_]}`, lo cual no es válido.

También tuve que aumentar el timeout, ya que algunas pruebas resultaron en timeout al principio. Esto fue modificando el parámetro de `def_to`.

5 Demostración en *Z/EVES*

Utilizando el asistente de pruebas *Z/EVES*, vamos a demostrar que la operación `TerminateProcess` preserva el invariante de estado `InvDispatcher`.

theorem `TerminateProcessInvDispatcher`

$$InvDispatcher \wedge TerminateProcess \Rightarrow InvDispatcher'$$

proof[`TerminateProcessInvDispatcher`]

```

  invoke TerminateProcess;
  split TerminateProcessOk;
  simplify;
  cases;
  invoke TerminateProcessOk;
  invoke InvDispatcher;
  equality substitute procQueue';
  simplify;
  next;
  invoke TerminateProcessError;
  invoke InvDispatcher;
  invoke  $\Xi$  Dispatcher;
  rewrite;
  next;

```

■

6 Casos de Prueba con FASTEST