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 planificaión "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*

procesos en estado listo (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.

$$nullp : PROCESS$$

$$quantum : TICK$$
$$quantum = 5$$

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

Dispatcher

$$procQueue : \mathbb{N} \rightarrowtail PROCESS$$
$$current : PROCESS$$
$$remTicks : TICK$$

InitDispatcher

Dispatcher

$procQueue = \emptyset$

$current = nullp$

$remTicks = 0$

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

InvDispatcher

Dispatcher

$current \notin \operatorname{ran} procQueue$

$nullp \notin \operatorname{ran} procQueue$

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

$\Delta Dispatcher$

$p? : PROCESS$

$procQueue \neq \emptyset$

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

$p? \neq current$

$p? \neq nullp$

$procQueue' = procQueue \cup \{max\,(\operatorname{dom} procQueue) + 1 \mapsto p?\}$

$current' = current$

$remTicks' = remTicks$

newProcessOkEmptyQueue

$\Delta Dispatcher$

$p? : PROCESS$

$procQueue = \emptyset$

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

$p? \neq current$

$p? \neq nullp$

$procQueue' = \{1 \mapsto p?\}$

$current' = current$

$remTicks' = remTicks$

NewProcessError

$\Xi Dispatcher$

$p? : PROCESS$

$p? \in \operatorname{ran} procQueue \lor p? = current \lor p? = nullp$

$NewProcess == NewProcessOk \lor newProcessOkEmptyQueue \lor NewProcessError$

Tick

$\Delta Dispatcher$

$remTicks > 0$

$current \neq nullp$

$procQueue' = procQueue$

$current' = current$

$remTicks' = remTicks - 1$

Timeout

$\Delta Dispatcher$

$procQueue \neq \emptyset$

$remTicks = 0$

$current \neq nullp$

$procQueue' = procQueue \cup \{max\,(\mathrm{dom}\, procQueue) + 1 \mapsto current\}$

$current' = nullp$

$remTicks' = remTicks$

TimeoutEmptyQueue

$\Delta Dispatcher$

$procQueue = \emptyset$

$remTicks = 0$

$current \neq nullp$

$procQueue' = \{1 \mapsto current\}$

$current' = nullp$

$remTicks' = remTicks$

DispatchProcess

$\Delta Dispatcher$

$current = nullp$

$procQueue \neq \emptyset$

$procQueue' = procQueue \setminus \{min\,(\mathrm{dom}\, procQueue) \mapsto procQueue(min\,(\mathrm{dom}\, procQueue))\}$

$current' = procQueue(min\,(\mathrm{dom}\, procQueue))$

$remTicks' = remTicks$

Idle

$\Xi Dispatcher$

$current = nullp$

$procQueue = \emptyset$

$Dispatch == Tick \lor Timeout \lor TimeoutEmptyQueue \lor DispatchProcess \lor Idle$

TerminateProcessOk

$\Delta Dispatcher$

$p? : PROCESS$

$current = p?$

$procQueue' = procQueue$

$current' = nullp$

$remTicks' = remTicks$

TerminateProcessError

$\Xi Dispatcher$

$p? : PROCESS$

$current \neq p?$

$TerminateProcess == TerminateProcessOk \lor TerminateProcessError$

3 Simulaciones en {*log*} usando el entorno NEXT

Se traduce la especificación a {*log*} (en **planificador.slog**), la cual está correctamente tipada, es decir, pasa el **type_check**. A continuación, se presentan dos simulaciones ejecutadas sobre el prototipo {*log*} usando el entorno NEXT y la traza de ejecución de cada una.

1ra simulación

```
[initial]:[invPFun,invIny,invNat,invDispatcher] >>
newProcess(NewProc:[[p1,p2,p3]]) >>
3:dispatch >>
terminateProc(Proc:p1) >>
8:dispatch.

Parameters:
Quantum = 5

Execution trace is:
ProcQueue = {},
Current = process:nullp,
RemTicks = 0
    ----> newProcess(NewProc:p1)
ProcQueue = {[1,p1]},
Current = process:nullp,
RemTicks = 0
    ----> newProcess(NewProc:p2)
ProcQueue = {[1,p1],[2,p2]},
Current = process:nullp,
RemTicks = 0
    ----> newProcess(NewProc:p3)
```

```
ProcQueue = {[1,p1],[2,p2],[3,p3]},
Current = process:nullp,
RemTicks = 0
    ----> dispatch
ProcQueue = {[2,p2],[3,p3]},
Current = p1,
RemTicks = 5
    ----> dispatch
ProcQueue = {[2,p2],[3,p3]},
Current = p1,
RemTicks = 4
    ----> dispatch
ProcQueue = {[2,p2],[3,p3]},
Current = p1,
RemTicks = 3
    ----> terminateProc(Proc:p1)
ProcQueue = {[2,p2],[3,p3]},
Current = process:nullp,
RemTicks = 0
    ----> dispatch
ProcQueue = {[3,p3]},
Current = p2,
RemTicks = 5
    ----> dispatch
ProcQueue = {[3,p3]},
Current = p2,
RemTicks = 4
    ----> dispatch
ProcQueue = {[3,p3]},
Current = p2,
RemTicks = 3
```

```
    ----> dispatch
ProcQueue = {[3,p3]},
Current = p2,
RemTicks = 2
    ----> dispatch
ProcQueue = {[3,p3]},
Current = p2,
RemTicks = 1
    ----> dispatch
ProcQueue = {[3,p3]},
Current = p2,
RemTicks = 0
    ----> dispatch
ProcQueue = {[3,p3],[4,p2]},
Current = process:nullp,
RemTicks = 0
    ----> dispatch
ProcQueue = {[4,p2]},
Current = p3,
RemTicks = 5

true
```

En esta simulación se encolan tres nuevos procesos, **p1**, **p2** y **p3**, en la cola de procesos listos y se inicia la ejecución del primero, la cual es interrumpida por la señal **terminateProc**. El planificador continúa con la ejecución del segundo proceso hasta que termina el tiempo de ejecución asignado (1 **quantum**) y es interrumpido por el dispatcher, para luego seguir con la ejecución del proximo proceso en la cola, **p3**.

Notamos como se verifican todos los invariantes durante la ejecución de la

simulación.

2da simulación

```
[initial]:[invPFun,invIny,invNat,invDispatcher] >>
dispatch >>
newProcess(NewProc:[[p1,p1,process:nullp]]) >>
2:dispatch >>
terminateProc(Proc:p2) >>
5:dispatch.

Parameters:
Quantum = 5

Execution trace is:
ProcQueue = {},
Current = process:nullp,
RemTicks = 0
    ----> dispatch
ProcQueue = {},
Current = process:nullp,
RemTicks = 0
    ----> newProcess(NewProc:p1)
ProcQueue = {[1,p1]},
Current = process:nullp,
RemTicks = 0
    ----> newProcess(NewProc:p1)
ProcQueue = {[1,p1]},
Current = process:nullp,
RemTicks = 0
    ----> newProcess(NewProc:process:nullp)
```

```
ProcQueue = {[1,p1]},
Current = process:nullp,
RemTicks = 0
    ----> dispatch
ProcQueue = {},
Current = p1,
RemTicks = 5
    ----> dispatch
ProcQueue = {},
Current = p1,
RemTicks = 4
    ----> terminateProc(Proc:p2)
ProcQueue = {},
Current = p1,
RemTicks = 4
    ----> dispatch
ProcQueue = {},
Current = p1,
RemTicks = 3
    ----> dispatch
ProcQueue = {},
Current = p1,
RemTicks = 2
    ----> dispatch
ProcQueue = {},
Current = p1,
RemTicks = 1
    ----> dispatch
ProcQueue = {},
Current = p1,
RemTicks = 0
```

```
    ----> dispatch
ProcQueue = {[1,p1]},
Current = process:nullp,
RemTicks = 0

true
```

La idea de esta simulación es comprobar el correcto funcionamiento de las operaciones en los casos de error o casos de ejecución borde. Por ejemplo la ejecución de las operaciones **idle**, **newProcessError**, **timeoutEmptyQueue** y **terminateProcError**.

Notamos como se verifican todos los invariantes durante la ejecución de la simulación.

4 Generado de Condiciones de Verificación

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 automaticamente.

Sobre la interacción con el VCG

En una primera iteración del comando `check_vcs_planificador`, hubo algunas condiciones de verificación que no fueron descargadas exitosamente. Por lo que fue necesario agregar las siguientes hipótesis en las condiciones de verificación que lo requerían:

- hipótesis `invDispatcher(ProcQueue, Current)` en `dispatch_pi_invIny`

- hipótesis `invIny(ProcQueue)` en `dispatch_pi_invDispatcher`

En ambos casos plantee la negación del enunciado y analicé el contraejemplo que presentaba para saber qué hipótesis era necesario agregar. No utilicé los comandos `vcgce`, `vcacg` y `findh`.

También fue necesario aumentar el timeout para que se logren descargar algunas pruebas, esto lo hice modificando el argumento de `def_to`.

5 Casos de Prueba con {*log*}-TTF

Se generan casos de prubea utilizando {*log*}-TTF para todas las operaciones.

Comandos utilizados

```
loadspec ../fastest.tex
selop NewProcess
genalltt
addtactic NewProcess_DNF_1 SP \notin p? \notin \ran procQueue
addtactic NewProcess_DNF_3 SP \in p? \in \ran procQueue
genalltt
genalltca
```

Con estos comandos se carga la especificación, se selecciona la operación para la cual generar los casos de prueba y se aplican las tacticas de testing. En primer lugar, se aplica *Disjunctive Normal Form (DNF)*, que lleva la operación a su forma normal disyuntiva, ya que es la táctica por defecto que aplica FASTEST al ejecutar el comando `genalltt`. Esto particiona la operación de acuerdo a las precondiciones de las suboperaciones que la definen, donde *NewProcess_DNF_1* corresponde a *newProcessOk*, *NewProcess_DNF_2* corresponde

a *newProcessOkEmptyQueue* y las tres restantes corresponden a las tres alternativas de *newProcessError*. Luego, se aplican las tacticas de partición estándar (*Standard Partition (SP)*) de $\notin$ y $\in$, donde resulta útil. Se obtiene el siguiente árbol:

- *NewProcess_VIS*
  - *NewProcess_DNF_1*
    - *NewProcess_SP_1*
    - *NewProcess_SP_2*
  - *NewProcess_DNF_2*
  - *NewProcess_DNF_3*
    - *NewProcess_SP_3*
    - *NewProcess_SP_4*
  - *NewProcess_DNF_4*
  - *NewProcess_DNF_5*

Finalmente se generan los casos de prueba, obteniendo el siguiente árbol de clases de prueba:

- *NewProcess_VIS*
  - *NewProcess_DNF_1*
    - *NewProcess_SP_2*
      - *NewProcess_SP_2_TCASE*
  - *NewProcess_DNF_2*
    - *NewProcess_DNF_2_TCASE*
  - *NewProcess_DNF_3*
    - *NewProcess_SP_3*
      - *NewProcess_SP_3_TCASE*
    - *NewProcess_SP_4*
      - *NewProcess_SP_4_TCASE*
  - *NewProcess_DNF_4*
    - *NewProcess_DNF_4_TCASE*
  - *NewProcess_DNF_5*
    - *NewProcess_DNF_5_TCASE*

Notamos que no se generó un caso de prueba para la hoja `NewProcess_SP_1`. Esto es porque es una clase de prueba vacía, ya que contiene tanto $procQueue \neq \{\}$ y $procQueue = \{\}$ como precondiciones.

Esquemas Z