

# **An Operating System Scheduler**

Using Z

Woodcock & Davies

## Scheduler

The **scheduler** is the component of an operating system that determines which process should be run, and when.

We will specify:

- the service provided—the scheduler **specification**
- a system that provides this service—the scheduler **implementation**

## Processes

- there is a single processor to be shared
- this is made available to one process at a time
- a process that is currently making use of the processor is said to be **running**

## Process states

- Since there is a single processor, at any time, there will be at most one process running. We will call this the **current** process.
- There may be several processes that are waiting to use the processor. These processes are said to be **ready**.
- There may be some processes that are waiting, not for the processor, but for a different resource or event. These processes are said to be **blocked**.

## Specification

Our system will deal with up to  $n$  processes, where  $n$  is a natural number.

$$| \quad n : \mathbb{N}$$

Each process will be associated with a **process identifier**, or *pid*.

$$Pid == 1 .. n$$

Zero is used to represent the ‘null process’: a marker that says that there is no process where this value is found.

$$\mathit{nullPid} == 0$$

An ‘optional pid’ can be either a true pid or the null pid:

$$\mathit{OptPid} == \mathit{Pid} \cup \{\mathit{nullPid}\}$$

## Abstract state

*AScheduler*

*current* : *OptPid*

*ready* :  $\mathbb{P} \text{Pid}$

*blocked* :  $\mathbb{P} \text{Pid}$

*free* :  $\mathbb{P} \text{Pid}$

$\langle \{ \text{current} \} \setminus \{ \text{nullPid} \},$

*ready*,

*blocked*,

*free*  $\rangle$  partition *PId*

## Initialisation

*ASchedulerInit*

*AScheduler'*

*current' = nullPid*

*ready' =  $\emptyset$*

*blocked' =  $\emptyset$*

*free' = Pid*

## Operations

- create a process, adding it to the set of ready processes
- dispatch one of the ready processes to the processor
- timeout a process, removing it from the processor and returning it to the set of ready processes
- block a process, removing it from the processor and adding it to the set of blocked processes
- wake up a blocked process, moving it into the set of ready processes
- destroy a process

## Create

*ACreate*

$\Delta$ *AScheduler*

$p! : PId$

$free \neq \emptyset$

$current' = current$

$ready' = ready \cup \{p!\}$

$blocked' = blocked$

$free' = free \setminus \{p!\}$

$p! \in free$

## Dispatch

*ADispatch*

$\Delta$ *AScheduler*

$p! : PId$

$current = nullPId$

$ready \neq \emptyset$

$current' \in ready$

$ready' = ready \setminus \{current'\}$

$blocked' = blocked$

$free' = free$

$p! = current'$

## Exercises

- write schemas to describe the effects of **timeout**, **block**, and **wake up**.
- write three partial operation schemas to describe the effect of destroying:
  - the current process
  - a ready process
  - a blocked process

## Design

- our program will use a simple data structure: an array and a few counters
- our use of this data structure can be modelled using **chains**: finite injections from  $PId$  to  $PId$  with a unique start and a unique end.

## Chains

### Chain

$start, end : OptPid$

$links : Pid \rightsquigarrow Pid$

$set : \mathbb{F} Pid$

$set = \text{dom } links \cup \text{ran } links \cup (\{start\} \setminus \{nullPid\})$

$links = \emptyset \Rightarrow start = end$

$links \neq \emptyset \Rightarrow$

$\{start\} = (\text{dom } links) \setminus \text{ran } links$

$\{end\} = (\text{ran } links) \setminus \text{dom } links$

$\forall e : set \mid e \neq start \bullet start \mapsto e \in links^+$

## Exercises

- how do we know that the *start* and the *end* of a given chain are uniquely defined?
- what must be true of the *start* and the *end* pids if *set* is empty?

## Initialisation

*ChainInit*

*Chain'*

*start' = nullPid end' = nullPid*

## Operations

- **push** an element onto the end of a chain
- **pop** an element from the front of a chain
- **delete** an element from a chain

## Pop

*PopSingleton*

$\Delta Chain$

$p! : PId$

$start \neq nullPId$

$links = \emptyset$

$start' = nullPId$

$links' = links$

$p! = start$

*PopMultiple*

$\Delta Chain$

$p! : PId$

$links \neq \emptyset$

$start' = links\ start$

$links' = \{start\} \triangleleft links$

$p! = start$

$Pop \hat{=} PopSingleton \vee PopMultiple$

## Delete

$Delete \hat{=} DeleteStart \vee DeleteMiddle \vee DeleteEnd$

*DeleteStart*

$\Delta Chain$

$p? : PId$

$p? = start$

$\exists p! : PId \bullet Pop$

*DeleteEnd*

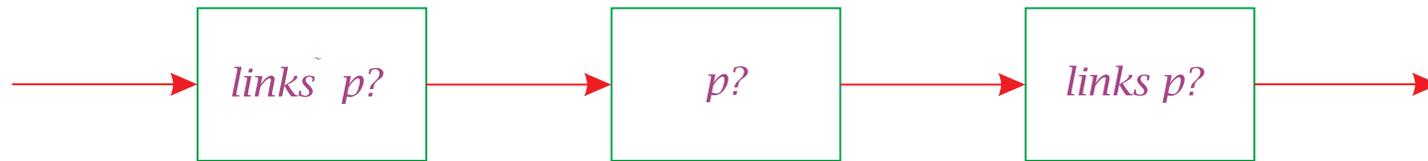
$\Delta Chain$

$p? : PId$

$p? \neq start$

$p? = end$

$links' = links \triangleright \{end\}$



Deleting a middle element

*DeleteMiddle* $\Delta Chain$  $p? : PId$  $p? \neq start$  $p? \neq end$  $p? \in set$  $links' = \{p?\} \triangleleft links \oplus \{links \sim p? \mapsto links p?\}$

## Design

*CScheduler*

*ReadyChain*

*BlockedChain*

*FreeChain*

*current* : *OptPid*

*chainstore* : *PId*  $\rightarrow$  *OptPid*

$\langle \{current\} \setminus \{nullPid\}, rset, bset, fset \rangle$  partition *PId*

*rlinks* = *rset*  $\triangleleft$  *chainstore*  $\triangleright$  *rset*

*blinks* = *bset*  $\triangleleft$  *chainstore*  $\triangleright$  *bset*

*flinks* = *fset*  $\triangleleft$  *chainstore*  $\triangleright$  *fset*

*current*  $\neq$  *nullPid*  $\Rightarrow$  *chainstore current* = *nullPid*

*ReadyChain*  $\hat{=}$

*Chain*[*rstart* / *start*, *rend* / *end*, *rlinks* / *links*, *rset* / *set*]

*BlockedChain*  $\hat{=}$

*Chain*[*bstart* / *start*, *bend* / *end*, *blinks* / *links*, *bset* / *set*]

*FreeChain*  $\hat{=}$

*Chain*[*fstart* / *start*, *fend* / *end*, *finks* / *links*, *fset* / *set*]

## Initialisation

*CSchedulerInit*

*CScheduler'*

*ReadyChainInit*

*BlockedChainInit*

*FreeChainFull*

*current' = nullPid*

*ReadyChainInit*  $\hat{=}$

*ChainInit*[*rstart'* / *start'*, *rend'* / *end'*,  
*rlinks'* / *links'*, *rset'* / *set'*]

*BlockedChainInit*  $\hat{=}$

*ChainInit*[*bstart'* / *start'*, *bend'* / *end'*,  
*blinks'* / *links'*, *bset'* / *set'*]

*FreeChainFull*

*FreeChain*

*fset'* = *PId*

## Operations

*PushReadyChain*  $\hat{=}$

*Push*[*rstart* / *start*, *rend* / *end*, *rlinks* / *links*, *rset* / *set*,  
*rstart'* / *start'*, *rend'* / *end'*, *rlinks'* / *links'*, *rset'* / *set'* ]

*PopReadyChain*  $\hat{=}$

*Pop*[*rstart* / *start*, *rend* / *end*, *rlinks* / *links*, *rset* / *set*,  
*rstart'* / *start'*, *rend'* / *end'*, *rlinks'* / *links'*, *rset'* / *set'* ]

*PopFreeChain*  $\hat{=}$

*Pop*[*fstart* / *start*, *fend* / *end*, *finks* / *links*, *fset* / *set*,  
*fstart'* / *start'*, *fend'* / *end'*, *finks'* / *links'*, *fset'* / *set'* ]

*CDispatch*

$\Delta CScheduler$

$p! : PId$

$\exists BlockedChain$

$\exists FreeChain$

$current = nullPId$

$rset \neq \emptyset$

*PopReadyChain*

$current' = p!$

*CCreate*

$\Delta CScheduler$

$p! : PId$

$\exists BlockedChain$

$fset \neq \emptyset$

$current' = current$

*PopFreeChain*

*PushReadyChain*[ $p! / p?$ ]

## Abstract state

*current* = 3

*ready* = {2, 4, 6}

*blocked* = {5, 7}

*free* = {1, 8, 9, 10}

## Possible concrete state

*current* = 3

*chainstore* = {1 ↦ 8, 2 ↦ 6, 3 ↦ 0, 4 ↦ 2, 5 ↦ 0,  
6 ↦ 0, 7 ↦ 5, 8 ↦ 9, 9 ↦ 10, 10 ↦ 0}

*rstart* = 4

*rend* = 6

*rlinks* = {4 ↦ 2, 2 ↦ 6}

*rset* = {2, 4, 6}

*bstart* = 7

*bend* = 7

*blinks* = {7 ↦ 5}

*bset* = {5, 7}

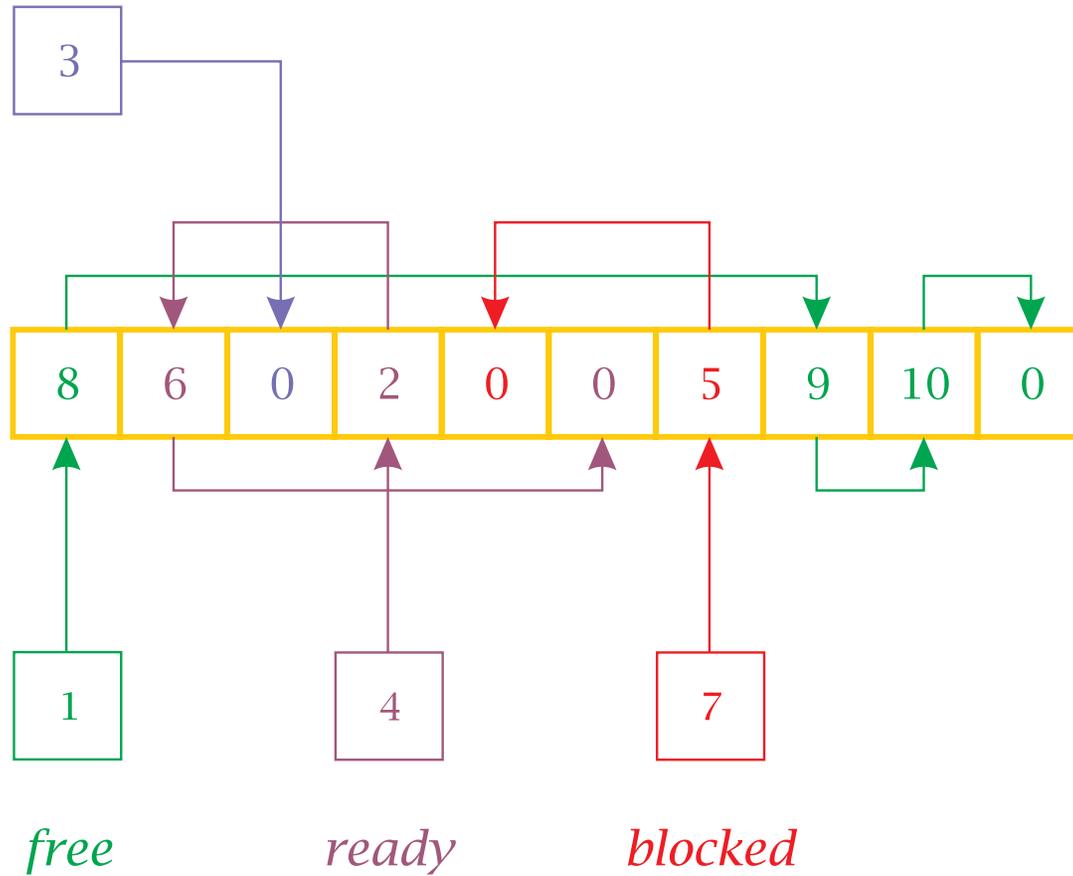
*fstart* = 1

*fend* = 10

*finks* = {1 ↦ 8, 8 ↦ 9, 9 ↦ 10}

*fset* = {1, 8, 9, 10}

*current*



A possible concrete state

## Retrieve function

*RetrScheduler*

*AScheduler*

*CScheduler*

*ready = rset*

*blocked = bset*

*free = fset*

## Correctness

$$CScheduler \vdash \exists_1 AScheduler \bullet RetrScheduler$$
$$CSchedulerInit \wedge Retr' \vdash ASchedulerInit$$
$$pre AOp \wedge Retr \wedge COp \wedge Retr' \vdash AOp$$