The UNIX file system: fragments of a formal specification

Mike Spivey

Michaelmas Term 1993

Introduction

Based on a chapter of

> Hayes, *Specification case studies*, Prentice–Hall International, 1993 (second ed.).

Other recommended books on Z:

> Potter, Sinclair and Till, *An introduction to formal specifications and Z*, Prentice–Hall International, 1991.
>
> Spivey, *The Z notation: a reference manual*, Prentice–Hall International, 1992 (second ed.).

The system call interface

Examples:

- ```
  fd = open("/usr/mike/foo", O_READ);
  ```

  locates a named file, returns a 'file descriptor'.

- ```
  n = read(fd, buf, 256);
  ```

  reads a block of characters, returns the number actually read.

- ```
  close(fd);
  ```

  breaks the link between file description and file.

Questions:
  ```

- What happens if I try to read beyond the end of a file?
- Can I open a file twice at the same time?
- What happens if I try to read from a file descriptor that is not opened?

Formal specification

... is better than reading the code!

- *Data abstraction*: use 'mathematical' data types – sets, sequences, functions – to model the data.
- *Procedural abstraction*: use predicate logic to describe transformations on the data.

Files

A file is a sequence of bytes:

$$FILE == \operatorname{seq} BYTE$$

In Z, a sequence is a function that has domain $1 \mathinner{..} n$ for some value of n. So $f(1)$ is the first byte; also $f(\#f)$ is the last byte.

Reading a file

Reading starts from a certain *offset*:

```
      ______    <-- length = 6
This is a file
      ^         <-- offset = 6
```

The bytes read are `s␣a␣fi`.

$$data = (1 \mathinner{..} length) \lhd (file \ \mathsf{after} \ offset)$$

The 'after' operator

Declaration:

$$\_ \ \mathsf{after} \ \_ : \operatorname{seq} X \times \mathbb{N} \longrightarrow \operatorname{seq} X$$

Some properties:

$$\#(f \text{ after } n) = max\{\#f - n, 0\}$$

$$(\forall i : 1 \mathinner{..} \#(f \text{ after } n) \bullet \\ (f \text{ after } n)(i) = f(n + i))$$

Example: if

$f =$ "`This␣is␣a␣file`"

then

f after $6 =$ "`s␣a␣file`"

The $\lhd$ operator

If $s = (1 \mathinner{..} n) \lhd f$ then

$$\#s = min\{n, \#f\}$$

$$(\forall i : 1 \mathinner{..} \#s \bullet \\ s(i) = f(i))$$

Putting it all together

ReadFile

$file, file' : FILE$
$offset?, length? : \mathbb{N}$
$data! : \text{seq}\, BYTE$

$file' = file$
$data! = (1 \mathinner{..} length?) \lhd (file \text{ after } offset?)$

A schema: a named box; parts above and below the line; the ! and ? conventions.

Writing a file

Writing also begins at an offset:

```
This is a file
          ^    <-- offset = 10
          longer file
---------------------
This is a longer file
```

What if the offset is greater than the original length?

```
This is a file
                    ^   <-- offset = 20
                    More stuff
-----------------------------
This is a file......More stuff
```

This is described by the equation:

$$file' = zero(offset?) \oplus file \oplus (data \;\mathsf{shifted}\; offset?).$$

Zero

Definition:

$$\begin{array}{|l} zero : \mathbb{N} \longrightarrow \mathrm{seq}\, BYTE \\ \hline zero(n) = (\, \lambda\, i : 1 \,..\, n \bullet ZERO\,) \end{array}$$

Properties:

$$\#(zero(n)) = n$$

$$(\, \forall\, i : 1 \,..\, n \bullet \\ \quad zero(n)(i) = ZERO\,)$$

Shifted

Definition:

$$\begin{array}{|l} [X] \\ \hline \_\;\mathsf{shifted}\;\_ : \mathrm{seq}\, X \times \mathbb{N} \longrightarrow (\mathbb{N} \pfun X) \\ \hline \mathrm{dom}(s\;\mathsf{shifted}\;n) = n + 1 \,..\, n + \#s \\ (\, \forall\, i : 1 \,..\, \#s \bullet \\ \quad (s\;\mathsf{shifted}\;n)(n + i) = s(i)\,) \\ \hline \end{array}$$

Overriding $\oplus$

$$\mathrm{dom}\,(f \oplus g) = \mathrm{dom}\, f \cup \mathrm{dom}\, g$$

$$(\, \forall\, x : \mathrm{dom}\, f \cup \mathrm{dom}\, g \bullet \qquad x \notin \mathrm{dom}\, g \Rightarrow (f \oplus g)(x) = f(x) \land \\ \quad x \in \mathrm{dom}\, g \Rightarrow (f \oplus g)(x) = g(x)\,)$$

$$(f \oplus g) \oplus h = f \oplus (g \oplus h)$$

Putting it together

WriteFile

$$file, file' : FILE$$
$$offset? : \mathbb{N}$$
$$data? : \operatorname{seq} BYTE$$

$$file' = zero(offset?) \oplus file \oplus (data? \ \mathsf{shifted} \ offset?)$$

Storing files

$[INUM]$

$$FSTORE == INUM \pfun FILE$$

CreateFstore

$$fstore, fstore' : FSTORE$$
$$inum! : INUM$$

$$inum! \notin \operatorname{dom} fstore$$
$$fstore' = fstore \oplus \{inum! \mapsto \langle\rangle\}$$

The created file is empty.

DestroyFstore

$$fstore, fstore' : FSTORE$$
$$inum? : INUM$$

$$inum? \in \operatorname{dom} fstore$$
$$fstore' = \{inum?\} \rhd fstore$$

The file being destroyed must exist beforehand: it no longer exists afterwards.

Reading from the file store

$$
\begin{array}{|l}
\hline
fstore, fstore' : FSTORE \\
inum? : INUM \\
offset?, length? : \mathbb{N} \\
data! : \mathrm{seq}\, BYTE \\
file, file' : FILE \\
\hline
inum? \in \mathrm{dom}\, fstore \\
file = fstore(inum?) \\
fstore' = fstore \oplus \{inum? \mapsto file'\} \\
\\
file' = file \\
data! = (1..length?) \lhd (file \ \mathsf{after}\ offset?) \\
\hline
\end{array}
$$

We abbreviate this as follows:

$$
\begin{array}{|l}
\hline
ReadFstore \\
\hline
ReadFile \\
\Phi Fstore \\
\hline
\end{array}
$$

where $\Phi Fstore$ is defined by

$$
\begin{array}{|l}
\hline
\Phi Fstore \\
\hline
fstore, fstore' : FSTORE \\
inum? : INUM \\
file, file' : FILE \\
\hline
inum? \in \mathrm{dom}\, fstore \\
file = fstore(inum?) \\
fstore' = fstore \oplus \{inum? \mapsto file'\} \\
\hline
\end{array}
$$

Putting schemas together like this merges the parts above and below the horizontal dividing line. The operation of writing on a file in the file-store can be defined similarly:

$$
\begin{array}{|l}
\hline
WriteFstore \\
\hline
WriteFile \\
\Phi Fstore \\
\hline
\end{array}
$$

Sequential access

If we read several chunks of data from a UNIX file, we get successive sequential portions – so each open file has a current position, recorded in a 'file pointer':

$$
\begin{array}{|l}
\hline
Fpos \\
\hline
inum : INUM \\
pos : \mathbb{N} \\
\hline
\end{array}
$$

Operations on a file pointer tend to change the position but not the i–number:

$$
\begin{array}{|l}
\hline
\Delta Fpos \\
\hline
Fpos;\ Fpos' \\
\hline
inum' = inum \\
\hline
\end{array}
$$

The operation of reading a file via a file pointer looks like this:

$$
\begin{array}{|l}
\hline
ReadFpos \\
\hline
ReadFstore \\
\Delta Fpos \\
\hline
inum? = inum \\
offset? = pos \\
pos' = pos + \#data! \\
\hline
\end{array}
$$

The *WriteFpos* operation is similar. There's also a *SeekFpos*, which just changes the position:

$$
\begin{array}{|l}
\hline
SeekFpos \\
\hline
fstore, fstore' : FSTORE \\
\Delta Fpos \\
newpos? : \mathbb{N} \\
\hline
fstore' = fstore \\
pos' = newpos? \\
\hline
\end{array}
$$

File descriptors

Each process has a collection of 'open' files, identified by 'file descriptors', and the system keeps a mapping from these to file positions:

$$[FD]$$

$$FTABLE == FD \pfun Fpos$$