Betriebssysteme 2
Übungsblatt 1

Felix J. Oppermann

4. November 2007

Aufgabe 1.1 *Stromorientiertes Dateisystem*

Aufgabe. Mit dieser einfachen Dateischnittstelle sollen Operationen zur Bearbeitung von Datensätzen mit *variabler* Satzlänge, die nach einem *Schlüssel* sotiert sind, realisiert werden.

Implementieren sie folgende Operationen:

```
s_read :   PROCEDURE ( fn : IN file_no; key : IN LONG_CARDINAL;
                       found, error : OUT BOOLEAN;
                       rec : OUT s_record );
s_insert : PROCEDURE ( fn : IN file_no; new : IN s_record;
                       error : OUT BOOLEAN );
```

s_read liefert in `rec` den Datensatz mit dem Schlüssel `key` zurück. Falls kein derartiger Datensatz exisitiert wird `found` auf `FALSE` gesetzt, falls ein Fehler auftritt, erhält `error` den Wert `TRUE`.

s_insert fügt den Satz an der dem Schlüssel entsprechenden Position in der Datei ein. Falls bereits ein Datensatz mit dem Schlüsselwert aus `new` existiert, wird er überschrieben.

Lösung

```
{ Die Datenstruktur ermöglicht das einlesen des Datensatz-Headers ohne
  die eigentlichen Daten das Datensatzes lesen zu müssen. Die Struktur wird bei
  der Suche verwendet. }
TYPE s_record_header = RECORD
    lng : CARDINAL;
    key : LONG_CARDINAL;
END;

s_record_header_size : CONSTANT CARDINAL == 6;



{ Größe in Byte des beim Verschieben verwendeten Puffers }
buffer_size : CONSTANT CARDINAL == 100;



s_read : PROCEDURE ( fn : IN file_no; key : IN LONG_CARDINAL;
                     found, error : OUT BOOLEAN;
                     rec : OUT s_record );
    old_read_position : CARDINAL;
    header : s_record_header;
    readcount : CARDINAL;
BEGIN
    { Sichern der Zeigerpositionen um diese später wiederherstellen zu können }
    old_read_position := l_pos(fn, 0, relative, read);

    l_pos(fn, 0, absolute, read);

    { Durchlaufen der Datensätze bis der gesuchte Datensatz gefunden oder
      das Ende der Datei erreicht ist. Bei der Suche werden zunächst nur
      die Header der Datensätze gelesen.}
    WHILE TRUE
    LOOP
        l_read(fn, header, s_record_header_size, readcount );
        IF readcount < s_record_header_size
        THEN
            { Das Dateiende ist erreicht. Die Suche kann ohne Erfolg beendet
              werden }
            found := FALSE;
```

```
            error := FALSE;
            EXIT;
        END;
        IF header.key == key
        THEN
            { Der gesuchte Datensatz ist gefunden. Er kann nun vollständig
              gelesen und an den Aufrufer zurück gegebenwerden }
            found := TRUE;
            l_read(fn, rec, header.lng + s_record_header_size, readcount);
            IF readcount < header.lng
            THEN
                error = TRUE;
            ELSE
                error = FALSE;
            END;
            EXIT;
        END
        IF header.key > key
        THEN
            { Der gesuchte Datensatz wurde nicht gefunden. Da die Datensätze
              geordnet sind kann die Suche abgebrochen werden. }
            found := FALSE;
            error := FALSE;
            EXIT;
        END;
        { Weiter beim nächsten Datensatz }
        l_pos(fn, header.lng + s_record_header_size, relative, read);
    REPEAT;

    { Zeigerpositionen wieder herstellen }
    l_pos(fn, old_read_position, absolute, read);
END;



s_insert : PROCEDURE ( fn : IN file_no; new : IN s_record;
                       error : OUT BOOLEAN );
    old_read_position : CARDINAL;
    old_write_position : CARDINAL;
    header : s_record_header;
    readcount : CARDINAL;
    move_error : BOOLEAN;
BEGIN
    { Sichern der Zeigerpositionen um diese später wiederherstellen zu können }
    old_read_position := l_pos(fn, 0, relative, read);
    old_write_position := l_pos(fn, 0, relative, write);

    l_pos(fn, 0, absolute, read);
    l_pos(fn, 0, absolute, write);

    { Durchlaufen der Datensätze in der Datei, bis die passende Einfügeposition
      für den neuen Datensatz gefunden ist. Bei der Suche werden nur
      die Header der Datensätze betrachtet.}
    WHILE TRUE
    LOOP
        l_read(fn, header, s_record_header_size, readcount);
        IF readcount < s_record_header_size
        THEN
            { Nichts mehr zu tun. Der Datensatz kann direkt angehängt werden. }
            EXIT;
        END;
```

```
        IF header.key == key
        THEN
            { Alle Daten beginnend beim nächsten Datensatz müssen verschoben
              werden, bevor der Datensatz an der aktuellen Position geschreiben
              werden kann. }
            s_move(l_pos( fn, 0, relative, read)
                   + header.lng + s_record_header_size,
                   new.lng - header.lng, move_error);
            IF move_error
            THEN
                error := TRUE;
                { Zeigerpositionen wieder herstellen }
                l_pos(fn, old_read_position, absolute, read);
                l_pos(fn, old_write_position, absolute, write);
                RETURN;
            END;
            EXIT;
        END;
        IF header.key > key
        THEN
            { Alles Daten beginnend beim aktuelle Datensatz müssen verschoben
              werden, bevor der Datensatz an der aktuellen Position geschreiben
              werden kann. }
            s_move(l_pos( fn, 0, relative, read),
                   new.lng + s_record_header_size, move_error);
            IF move_error
            THEN
                error := TRUE;
                { Zeigerpositionen wieder herstellen }
                l_pos(fn, old_read_position, absolute, read);
                l_pos(fn, old_write_position, absolute, write);
                RETURN;
            END;
            EXIT;
        END;
        { Weitersuchen }
        l_pos(fn, header.lng + s_record_header_size, relative, read);
    REPEAT;

    { Schreibzeiger auf die durch die Suche bestimmte Position des Lesezeigers
      setzen und die neuen in die Datei Daten schreiben }
    l_pos(fn, l_pos( fn, 0, relative, read), absolute, write)
    l_write(fn, new, new.lng + s_record_header_size);

    { Zeigerpositionen wieder herstellen }
    l_pos(fn, old_read_position, absolute, read);
    l_pos(fn, old_write_position, absolute, write);
END;



s_move : PROCEDURE ( fn : IN file_no; start : IN CARDINAL; move : IN CARDINAL
                     error : OUT BOOLEAN );
    old_read_position : CARDINAL;
    old_write_position : CARDINAL;
    position : CARDINAL;
    buffer : ARRAY[0..buffer_size] OF BYTE;
    readcount : CARDINAL;
BEGIN
    { Sichern der Zeigerpositionen um diese später wiederherstellen zu können }
    old_read_position := l_pos(fn, 0, relative, read);
```

```
old_write_position := l_pos(fn, 0, relative, write);

IF move > 0
THEN
    { Die Daten in Blöcken der Größe block_size verschieben, bis der
      Rest kleiner als block_size ist. }
    position := l_length(fn) - block_size;
    WHILE position >= start
    LOOP
        l_pos(fn, position, absolute, read);
        l_pos(fn, position + move, write);
        l_read(fn, buffer, buffer_size, readcount)
        IF readcount < block_size
        THEN
            { Fehler: Hier müsste der Ausgangszustand wieder hergestellt
                      werden! }
            error := TRUE;
            RETURN;
        END;
        l_write(fn, buffer, buffer_size)
        position := position - block_size;
    REPEAT;
    { Den Rest verschieben }
    l_pos(fn, start, absolute, read);
    l_pos(fn, start + move, write);
    l_read(fn, buffer, buffer_size - (start - position), readcount);
    IF readcount < block_size
    THEN
        { Fehler: Hier müsste der Ausgangszustand wieder hergestellt
                  werden! }
        error := TRUE;
        RETURN;
    END;
    l_write(fn, buffer, buffer_size - (start - position));
ELSE
    { Die Daten in Blöcken der Größe block_size verschieben, bis der
      Rest kleiner als block_size ist. }
    position := start;
    WHILE position < l_length(fn)
    LOOP
        l_pos(fn, position, absolute, read);
        l_pos(fn, position + move, write);
        l_read(fn, buffer, buffer_size, readcount)
        IF readcount < block_size
        THEN
            { Fehler: Hier müsste der Ausgangszustand wieder hergestellt
                      werden! }
            error := TRUE;
            RETURN;
        END;
        l_write(fn, buffer, buffer_size)
        position := position + block_size;
    REPEAT;
    { Den Rest verschieben }
    l_pos(fn, position, absolute, read);
    l_pos(fn, position + move, absolute, write);
    l_read(fn, buffer, l_length(fn) - position, readcount);
    IF readcount < block_size
    THEN
        { Fehler: Hier müsste der Ausgangszustand wieder hergestellt
                  werden! }
```

```
                error := TRUE;
                RETURN;
            END;
            l_write(fn, buffer, l_length(fn) - position);
            { Das jetzt ungenutzte Dateiende löschen }
            l_delete(fn, -move);
        END;

        { Zeigerpositionen wieder herstellen }
        l_pos(fn, old_read_position, absolute, read);
        l_pos(fn, old_write_position, absolute, write);
END;
```