Technische Universität München
Fakultät für Informatik
Dr. M. Schneider, G. Bauer,
Dr. M. Pizka, Dr. B. Rumpe,
Dr. R. Steinbrüggen

WS 2002/2003
Lösungsvorschläge zu Blatt 10
10. Januar 2003

Übungen zur Vorlesung Einführung in die Informatik III

Aufgabe 35 Synchronisation mit busy waiting auf der MI

Wir betrachten eine Erweiterung der fiktiven Rechenmaschine MI mit vier Rechnerkernen und einem gemeinsamen Speicher. Auf dieser soll ein Programmstück *P* als kritischer Bereich über busy waiting realisiert werden.

(a)

```
aufg35:    SEG
           JUMP start

sema:      DD B 1                    -- Semaphor, zu Beginn frei

start:     MOVE W I H'10000', SP     -- Keller initialisieren
           LSCBADR I H'11000'        -- Systemkontrollblock initialisieren

loop:      JBCCI I 7, sema, loop     -- Semaphor belegen

           -- Durchlaufen von Programmstück P

           MOVE W R0, R0             -- nur für Testhilfe
           MOVE B I 1, sema          -- Semaphor freigeben

endloop:   HALT                      -- Rechnerkern anhalten
           END
```

(b)

```
aufg35:    SEG
           JUMP start

sema:      DD B 1                    -- Semaphor, zu Beginn frei
sollRk:    DD W 3                    -- gewünschte RK-Nummer
NrRk:      RES 4                     -- eigene RK-Nummer

start:     MOVE W I H'10000', SP     -- Keller initialisieren
           LSCBADR I H'11000'        -- Systemkontrollblock initialisieren

loop:      JBCCI I 7, sema, loop     -- Semaphor belegen

           SPRKMAP NrRk              -- RK-Nummer beschaffen
           SH I -16, NrRk, NrRk      -- (RK-Nummer steht im linken Halbwort)
if:        CMP W NrRk, sollRk
           JNE endif
```

```
            -- Durchlaufen von Programmstück P

then:       MOVE W R0, R0               -- nur für Testhilfe
            SUB W I 1, sollRk           -- nächsten RK freigeben

endif:      MOVE B I 1, sema            -- Semaphor freigeben
test:       CMP W sollRk, I 0
            JGE loop

endloop:    HALT                        -- Rechnerkern anhalten
            END
```

Aufgabe 36 Speichersegmente

a) Diese erste Implementierung ist sehr naiv. Sie spart zwar Speicherplatz, wird aber wegen des hohen Aufwands für das Verschieben von Segmenten nicht eingesetzt. Wir merken uns in einem globalen Feld

⌈1:maxseg⌉ **array var adr** segstart;

die Anfangsadressen der Segmente und in einer globalen Variablen

var {0:maxseg} k := 0;

die zuletzt vergebene Segmentnummer. Eine weitere Variable

var adr eosp := 1;

gibt das Ende des belegten Speichers an.

func newSegment = () **snr**:
⌈ k := k + 1;
segstart⌈k⌉ := eosp;
⌊ k

trägt ein neues leeres Segment am Ende des Speichers ein.

func setSegmentSize = (**snr** s, **nat** sz):
⌈ **nat** diff := sz - (**if** s < k **then** segstart⌈s+1⌉ **else** eosp **fi** - segstart⌈s⌉);
„Verschiebe den Speicherinhalt ab Adresse segstart⌈s+1⌉ um diff Positionen“;
eosp := eosp + diff;
⌊ **for** i := s + 1 **to** k **do** segstart⌈i⌉ := segstart⌈i⌉ + diff **od**

sorgt durch Verschieben dafür, dass das Segment s zusammenhängenden Speicherplatz hat. Analog:

```
proc removeSegment = (snr s):
    ⌈ nat length = if s = k then eosp else segstart[s+1] fi - segstart[s];
      „Verschiebe den Speicherinhalt ab Adresse segstart[s+1] um length Po-
      sitionen zurück“;
      for i := s + 1 to k do segstart[i] := segstart[i] - length od
      eosp := eosp - length;
      if s = k then segstart[s] := eosp
    ⌊          else segstart[s] := segstart[s+1] fi
```

(Die Segmentnummer s wird nicht mehr wiederverwendet.)

```
func getValue = (snr s, adr i, m v) m:
    ⌈ sp[segstart[s]+i] ⌋
```

Analog: setValue

	eosp	*k*	*segstart*		
1) Kreieren dreier Segmente	0	3	0	0	0
2) Setzen der Größe Segment 1 auf 5	5	3	0	5	5
3) Setzen der Größe Segment 2 auf 5	10	3	0	5	10
4) Setzen der Größe Segment 3 auf 5	15	3	0	5	10
5) Setzen der Größe Segment 1 auf 8	18	3	0	8	13
6) Setzen der Größe Segment 2 auf 7	20	3	0	8	15
7) Löschen von Segment 1	12	3	0	0	7
8) Setzen der Größe Segment 3 auf 8	15	3	0	0	7
9) Setzen der Größe Segment 1 auf 2	17	3	0	2	9

Tab. 36.1 Zustand der Segment-Tabelle

b) Anstelle der obigen Relativadressierung

```
getValue(s,i) = sp[segstart[s]+i]
```

führen wir eine Adressierung über eine Menge von Hilfsfeldern ein. Jedes Segment erhält eine Tabelle, die die tatsächliche Lage der angesprochenen Zelle im Speicher angibt. Die Zellen heißen meist Seiten, die Hilfstabellen Seiten-Kachel-Tabellen, die Hilfstabellen sind über eine Segmenttabelle nach Segment-Nummern organisiert. Auf der MI sind für die Segmente P0-Bereich, P1-Bereich und Systembereich statt einer Segmenttabelle drei Register vorgesehen, die für jeden Prozess neu gesetzt werden. Diese Register erhalten jeweils die Basisadresse einer Seiten-Kachel-Tabelle, siehe Abb. 36.1.

Globale Datenstrukturen:

```
[1:maxseg] array [1:n] array var adr st
```

(Segment-Tabelle und Seiten-Kachel-Tabelle)

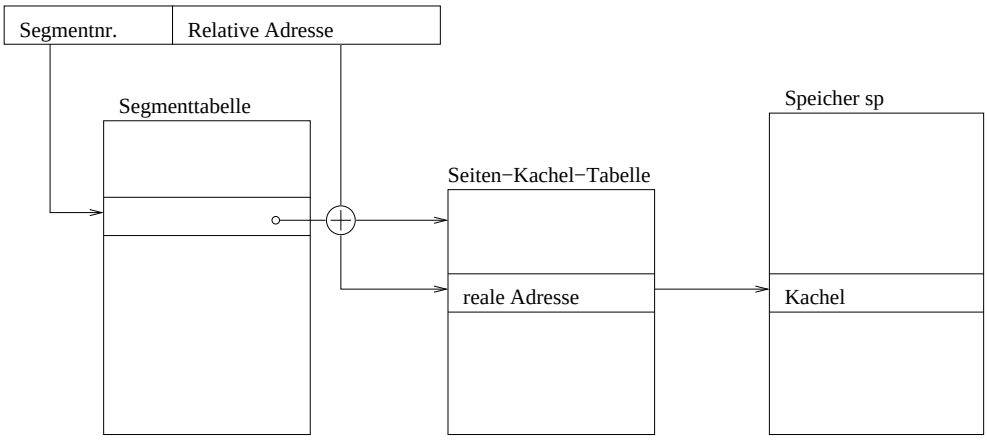


Abb. 36.1 Seiten-Kachel-Tabelle

var {0:maxseg} k := 0

Information, welche Zellen des realen Speichers („Kacheln“) zur Zeit frei sind:

⌈1:n⌉ **array var bool** use

(oder eine Liste der freien Kacheln)

```
func newSegment = snr:
    ⌈ k := k + 1;
      for i := 1 to n do st[k][i] := 0 od;
    ⌊ k
```

initialisiert die Seiten-Kachel-Tabelle des neuen Segments mit „noch nicht zuweisen“.

```
proc setSegmentSize = (snr s, nat sz):
    ⌈ for i := „größte belegte Relativadresse im Segment s“ to sz do
        adr f = „freie Adresse in sp“;
        use[f] := true;                          (* Zelle ist jetzt besetzt *)
        adr j = „kleinste freie Relativadresse im Segment s“;
    ⌊   st[s][j] := f od
```

```
proc removeSegment = (snr s):
    ⌈ for i := 1 to n do
        if st[s][i] ≠ 0 then
        use[st[s][i]] := false;
    ⌊ st[s][i] := 0 fi od
```

```
func getValue = (snr s, adr i) m:
    ⌈ sp[st[s][i]] ⌋
```

In getValue wird keine Fehlerbehandlung für den Zugriff auf eine nicht existente Seite durchgeführt. Weitere Effizienzsteigerungen können bei der Verwaltung der Segment- und

Seiten-Kachel-Tabelle durchführt werden. So können die Seiten-Kachel-Tabellen selbst dynamisch erweiterbar sein. Darüber hinaus können Strategien integriert werden, Seiten, die länger nicht benutzt werden, auf einen Hintergrundspeicher auszulagern.

c) Wir benötigen

- $maxseg * n$ Worte für die Segment-Tabelle (*st*)
- n Worte (bzw. Bits) für die Freispeicherverwaltung (*use*)

d.h. $(maxseg + 1) * n$ Worte.

Der Aufwand verteilt sich auf n Elemente aus sp, die jeweils aus b Bytes bestehen. Also ist der zusätzliche Aufwand pro Wort:

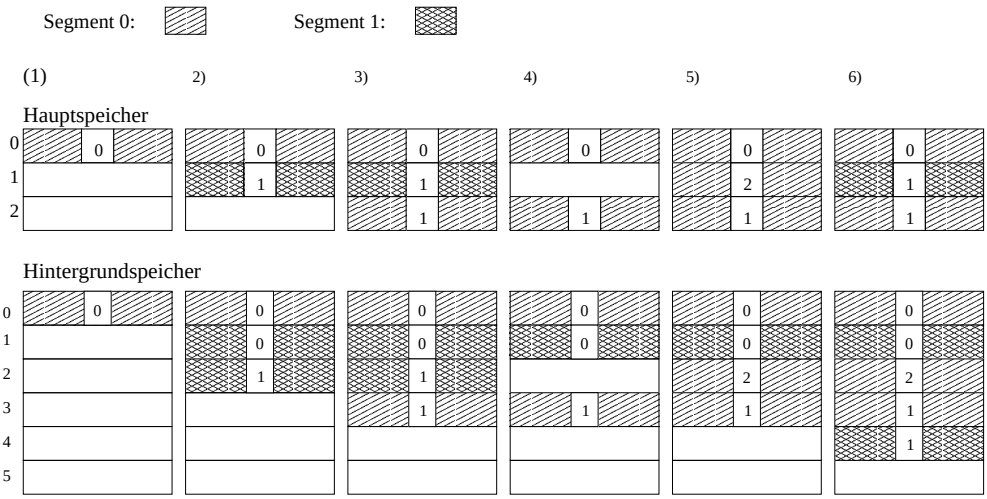
$$\frac{(maxseg + 1) * n}{b * n} = \frac{maxseg + 1}{b}$$

(i) Für Ein-Byte-Zellen (b=1) ist der Aufwand indiskutabel hoch: 4 zusätzliche Bytes.

(ii) Dagegen kann sich die Verwaltung bei größeren Blöcken, z.B. von 512 Bytes auszahlen:
$\frac{4}{512} \approx 0.008$ zusätzliche Bytes pro „genutztem“ Byte.

Aufgabe 37 Seitenaustauschverfahren

a) Die Nummer in den Kacheln geben die Seitenummer im Segment an:



Setzt man A=0 und B=1, dann stimmt der Zustand bei (6) mit der Abbildung aus Aufgabe 1 überein

b) ** Konstanten: maximum segment index: ms, maximum memory page index: mmp, maximum file page index: mfp, maximum cell index: mc page size: ps*/*

nat ps:=100;

nat ms,mmp,mfp,mc:=maxseg-1,2,5,ps-1;

** Speicher als Arrays von Kacheln */*

[0:mfp] **array** [0:mc] **array var m** file;

```
[0:mmp] array [0:mc] array var m mem;
/* Segment-Tabelle -> Seiten-Kachel-Tabelle -> ( Speicher Kachel-Nr. , File Kachel-Nr. ) */
[0:ms] array [0:mfp] array [0:1] array var nat st;
/* Zähler */
var {0:ms} sc:= 0; /* Anzahl Einträge in Segment-Tabelle: segment counter */
/* Anzahl Seiten in der Seiten-Kachel-Tabelle: segment page counter */
[0:ms] array var {0:ms} spc; for i:= 0 to ms do spc[i]:=0 od;
var nat uc:=0; /* Zugriffszähler als relative Zeit: usage counter*/
/* Zusatzinformationen */
[0:mfp] array var bool fileUsed; for i:= 0 to mfp do fileUsed[i]:= false od;
[0:mmp] array var bool memUsed; for i:= 0 to mmp do memUsed[i]:= false od;
[0:mmp] array var nat memUsedDate;
[0:mmp] array [0:1] array var nat memSegmentPage;

func newSegment = snr:
    ⌈ for i:= 0 to mfp do st[sc][i][0],st[sc][i][1]:=-1,-1 od; sc:=sc+1; sc-1 ⌋


proc removeSegment = ( snr s): ⌈ setSegmentSize( s,0 )⌋

proc setSegmentSize = ( snr s, nat sz ):
    ⌈ nat pc = ((sz-1)/ps)+1; if pc > spc[s ]  then for i:=spc[s] to pc-1 do addPage(s) od
                              else if pc < spc[s] then
                                                  for i:=spc[s] to pc+1 do removePage(s) od fi fi ⌋


proc setValue = ( snr s, nat n, m v): ⌈ mem[getMemoryPage(s,n/ps)][n mod ps]:=v ⌋
func getValue = ( snr s, nat n ) m:  ⌈ mem[getMemoryPage(s,n/ps)][n mod ps] ⌋

func getMemoryPage = ( snr s, nat p ) nat : /* Stelle Hauptspeicherseite bereit */
    ⌈ nat mp=st[s][p][1]; if mp=-1 then mp:=loadPage(s,p) fi;
    uc:=uc+1;memUsedDate[mp]:=uc; mp⌋

proc addPage = ( snr s ): /* Finde erste freie File-Kachel */
    ⌈ var nat fp:=-1; for i:= 0 to mfp do if fp=-1 ∧   ¬fileUsed[i] then fp:=i od;
     for i:=0 to mc do file[fp][i]=0 od;fileUsed[fp]:=true;
    st[s][spc[s]][1]:=fp; spc[s]=spc[s]+1 ⌋


proc removePage = ( snr s )
    ⌈ nat p = spc[s]-1; nat mp,fp = st[s][p][0],st[s][p][1];
    if mp ≠-1 then memUsed[mp]:=false fi; if fp ≠-1 then fileUsed[fp]:=false fi;
    st[s][p][0],st[s][p][1]:=-1,-1; spc[s]:=spc[s]-1⌋
```

```
func loadPage = ( snr s, nat p ) nat : /* Finde die erste freie Kachel oder Verdränge LRU */
    ⌈ var nat u,lru,mp := uc+1,-1,-1;
    for i:= 0 to mmp do if mp=-1 ∧   ¬memUsed[i] then mp:=i fi;
    if memUsed[i] ∧ memUsedDate[i] < u then u,lru:=memUsedDate[i],i fi od;
    if mp=-1 then /* Keine freie Kachel, dann schreiben der Least Recent Used Kachel */
    nat lrus = memSegmentPage[lru][0]; nat lrup := memSegmentPage[lru][1];
    for i:=0 to mc do file[ st[lurs][lurp][1] ] [i]:=mem[lru][i] od;
    st[lurs][lurp][0]:=-1; mp:=lru fi;
    /* Lesen */
    for i:= 0 to mc do mem[mp][i]:=file[st[s][p][1]][i] od;
    memSegmentPage[mp][0]:=s;memSegmentPage[mp][1]:=p;
    memUsed[mp]:=true; st[s][p][0]:=mp; mp ⌋
```

c) Vorteile der Segmentierung: Eine einfache Trennung von Bereichen wird erreicht. Eine Speicherverschiebung ist nicht notwendig.

Nachteil der Segmentierung: Es existiert ein Mehraufwand bei jedem Speicherzugriff.

Vorteile des Seitenaustauschverfahrens: Ein großer virtueller Speicher ist vorhanden.

Nachteil des Seitenaustauschverfahrens: Ein schlechte Strategie kann zu ständigem Lesen und Schreiben von Seiten auf dem Hintergrundspeicher führen („Seitenflattern“).

Aufgabe 38 (P) Segmentierung und Seitenaustauschverfahren — Java

```
import java.io.RandomAccessFile;
import java.io.IOException;
import java.util.Vector;

public class VirtualMemory
{
  private final static int  UNDEFINED = -1;

  private Vector  segments = new Vector();
  private int     pageSize;

  private byte    memoryPages[][];
  private RandomAccessFile  file;

  private boolean isFilePageUsed[];
  private boolean isMemoryPageUsed[];

  private long    memoryPageUsedDate[];
  private int     memoryPageToPageTableEntry[][];
  private long    usageCount = 0;

  public VirtualMemory( int pageSize, int memoryPageCount,
                        int filePageCount, String pageFileName )
  {
     this.pageSize = pageSize;
```

```
   memoryPages = new byte[memoryPageCount][];
   for( int p = 0 ; p < memoryPages.length ; p++ )
      { memoryPages[p]=new byte[pageSize]; }

   isFilePageUsed   = new boolean[ filePageCount   ];
   isMemoryPageUsed = new boolean[ memoryPageCount ];

   memoryPageUsedDate         = new long[ memoryPageCount ];
   memoryPageToPageTableEntry = new int [ memoryPageCount ][];

   try { file = new RandomAccessFile( pageFileName,"rw" ); }
   catch( IOException e )
   {
       throw new RuntimeException( e.getMessage() );
   }
}

synchronized public int newSegment()
{
   // reuse first free slot
   for ( int i = 0 ;  i < segments.size() ; i++ )
   {
      if ( segments.elementAt(i) == null ) return i;
   }

   // add page table for segment
   segments.addElement( new Vector() );

   return segments.size()-1 ;
}

synchronized public void removeSegment( int segment )
{
   setSegmentSize( segment, 0 );
   segments.setElementAt( null, segment );
}

synchronized public void setSegmentSize( int segment, int size )
{
   Vector pageTable = (Vector)segments.elementAt( segment );
   int currentPageCount = pageTable.size();
   int pageCount        = ((size-1)/pageSize)+1;

   if ( pageCount > currentPageCount )
   {
      for (  int i = currentPageCount ; i < pageCount ; i++)
      { addPage(pageTable ); }
   }
   else if ( pageCount < currentPageCount )
   {
```

```
      for ( int i = currentPageCount ; i > pageCount ; i-- )
          { removePage( pageTable ); }
   }
}

private void addPage( Vector pageTable )
{
   int freeFilePage = UNDEFINED;

   for ( int filePage = 0; filePage < isFilePageUsed.length ; filePage++ )
   {
      if( !isFilePageUsed[filePage] )
      {
         freeFilePage = filePage;
         isFilePageUsed[filePage]=true;
         break;
      }
   }

   if ( freeFilePage==UNDEFINED )
      throw new OutOfMemoryError("no free file page found");

   try
   {
      file.seek( freeFilePage * pageSize );
      file.write( new byte[pageSize] );

      // entry is an int array with at [0]=memory page index
      // , [1]=file page index
      pageTable.addElement( new int[]{UNDEFINED,freeFilePage} );

      System.out.println( "Added page " + (pageTable.size()-1)
      + " in file page " + freeFilePage );
   }
   catch( IOException e )
   {
      throw new RuntimeException( e.getMessage() );
   }
}

private void removePage( Vector pageTable )
{
   int pageTableEntry[] = (int[])pageTable.lastElement();
   pageTable.removeElementAt( pageTable.size()-1 );

   int filePage = pageTableEntry[1];
   isFilePageUsed[filePage]=false;

   int memoryPage = pageTableEntry[0];
   if ( memoryPage != UNDEFINED )
   {
```

```
      isMemoryPageUsed[memoryPage]=false;
   }

   System.out.println( "Removed page " + pageTable.size() + " as file page "
      + filePage + " and memory page " + memoryPage );

}

synchronized public byte getValue( int segment, int offset )
{
   return memoryPages[getMemoryPage(segment,offset/pageSize)][offset%pageSize];
}

synchronized public void setValue( int segment, int offset, byte  b )
{
   memoryPages[getMemoryPage(segment,offset/pageSize)][offset%pageSize]=b;
}

private int getMemoryPage( int segment, int page  )
{
   Vector pageTable = (Vector)segments.elementAt(segment);
   int pageTableEntry[]=(int[])pageTable.elementAt( page );

   int memoryPage = pageTableEntry[0];
   if ( memoryPage == UNDEFINED  )
   {
      loadPage( pageTableEntry );
      memoryPage =  pageTableEntry[0];
   }

   usageCount++;
   memoryPageUsedDate[memoryPage]=usageCount;

   System.out.println( "Segment " + segment + " uses page " + page
      + " in memory page " + memoryPage + " at " + usageCount );

   return memoryPage;
}

private void loadPage( int[] pageTableEntry )
{
    int memoryPageToReplace = selectMemoryPageToReplace();

    try
    {
      if( isMemoryPageUsed[ memoryPageToReplace] )
      {
        write( memoryPageToReplace );
      }
      read( memoryPageToReplace, pageTableEntry );
    }
```

```
    catch( IOException e )
    {
       throw new RuntimeException( e.getMessage() );
    }
}

private int selectMemoryPageToReplace()
{
    long leastRecentUsedDate = usageCount+1;
    int memoryPageToReplace = UNDEFINED;

    // first free slot, or least recent used
    for( int p = 0 ; p < isMemoryPageUsed.length ; p++ )
    {
       if ( !isMemoryPageUsed[p] )
       {
         memoryPageToReplace = p;
         break;
       }
       else if ( memoryPageUsedDate[p] < leastRecentUsedDate )
       {
          leastRecentUsedDate = memoryPageUsedDate[p];
          memoryPageToReplace = p;
       }
    }
    return memoryPageToReplace;
}

private void write( int memoryPage  ) throws IOException
{
    int pageTableEntry[] = memoryPageToPageTableEntry[memoryPage];
    int filePage = pageTableEntry[1];

    System.out.println( "Write memory page " + memoryPage
              + " to file page " + filePage );

    file.seek( filePage * pageSize );
    file.write( memoryPages[memoryPage] );

    pageTableEntry[0]=UNDEFINED; // remove memory page index
    memoryPageToPageTableEntry[memoryPage]=null;
    isMemoryPageUsed[memoryPage]=false;
}

private void read( int memoryPage, int[] pageTableEntry ) throws IOException
{
   int filePage = pageTableEntry[1];

   System.out.println( "Load file page " + filePage
      + " to memory page " + memoryPage );
```

```
      file.seek( filePage * pageSize );
      file.read( memoryPages[memoryPage] );

      memoryPageToPageTableEntry[memoryPage]=pageTableEntry;
      isMemoryPageUsed[memoryPage]=true;
      pageTableEntry[0]=memoryPage;
   }

   public static void main( String arguments[] )
   {
      VirtualMemory memory = new VirtualMemory( 100, 3, 6, "pageFile" );
      printAction(1);
      int s0 = memory.newSegment(); memory.setSegmentSize( s0, 100 );
      memory.setValue( s0,99,(byte)0x01);

      printAction(2);
      int s1 = memory.newSegment(); memory.setSegmentSize( s1, 101 );
      memory.setValue( s1,100,(byte)0x03);

      printAction(3);
      memory.setSegmentSize( s0, 101 );
      memory.setValue( s0, 100, (byte)0x03 );

      printAction(4);
      memory.setSegmentSize( s1, 100 );

      printAction(5);
      memory.setSegmentSize( s0, 201 );
      byte b = (byte)0x05; System.out.println("Value:" + b );

      memory.setValue( s0, 200, b );
      b = memory.getValue( s0, 100 );
      b = memory.getValue( s0, 0 );

      printAction(6);
      memory.setSegmentSize( s1, 101 );
      memory.setValue( s1, 100, (byte)0x06 );

      printAction(7);
      System.out.println("Value:" + memory.getValue( s0, 200 ));
   }

   static void printAction( int action )
         { System.out.println("\nAction: (" + action + ")" ); }
}

Ausgabenprotokoll:

Action: (1)
Added page 0 in file page 0
Load file page 0 to memory page 0
```

```
Segment 0 uses page 0 in memory page 0 at 1

Action: (2)
Added page 0 in file page 1
Added page 1 in file page 2
Load file page 2 to memory page 1
Segment 1 uses page 1 in memory page 1 at 2

Action: (3)
Added page 1 in file page 3
Load file page 3 to memory page 2
Segment 0 uses page 1 in memory page 2 at 3

Action: (4)
Removed page 1 as file page 2 and memory page 1

Action: (5)
Added page 2 in file page 2
Value:5
Load file page 2 to memory page 1
Segment 0 uses page 2 in memory page 1 at 4
Segment 0 uses page 1 in memory page 2 at 5
Segment 0 uses page 0 in memory page 0 at 6

Action: (6)
Added page 1 in file page 4
Write memory page 1 to file page 2
Load file page 4 to memory page 1
Segment 1 uses page 1 in memory page 1 at 7

Action: (7)
Write memory page 2 to file page 3
Load file page 2 to memory page 2
Segment 0 uses page 2 in memory page 2 at 8
Value:5


Ausgabenprotokoll:

[V/0/0]load file page:0 replace memory page:0
[1/0/1][:/0/2][0/0/3]  >V1:0<
[V/0/100]load file page:1 replace memory page:1
[1/0/101][:/0/102][1/0/103]  >V1:1<
[V/0/200]load file page:2 replace memory page:2
[1/0/201][:/0/202][2/0/203]  >V1:2<
----------------
[V/0/0][1/0/1][:/0/2][0/0/3][ /0/4]  >V1:0<
[V/0/100][1/0/101][:/0/102][1/0/103][ /0/104]  >V1:1<
[V/0/200][1/0/201][:/0/202][2/0/203][ /0/204]  >V1:2<
[V/1/0]load file page:3 replace memory page:0
[2/1/1][:/1/2][0/1/3]  >V2:0<
```

```
[V/1/100]load file page:4 replace memory page:1
[2/1/101][:/1/102][1/1/103]  >V2:1<
[V/1/200]load file page:5 replace memory page:2
[2/1/201][:/1/202][2/1/203]  >V2:2<
----------------
[V/1/0][2/1/1][:/1/2][0/1/3][ /1/4]  >V2:0<
[V/1/100][2/1/101][:/1/102][1/1/103][ /1/104]  >V2:1<
[V/1/200][2/1/201][:/1/202][2/1/203][ /1/204]  >V2:2<
```