Systemprogrammierung

Mitschrift der Musterlösungen der Übungen

Vorlesung gehalten von Prof. Dr.-Ing. Stefan Kowalewski
im Wintersemester 2004/05 an der RWTH Aachen

Eine studentische Mitschrift von Alexander Schiffel
alexander.schiffel@rwth-aachen.de

27. Februar 2005

Es besteht kein Anspruch auf Vollständigkeit oder Richtigkeit

(Anmerkung : Übung 1+2 [Ausnahme: Quelltext Ü2, A3] und Übung 5, A1 sind eigene Lösungen)

Übung 1

Aufgabe 1

a)

```
#include <stdio.h>


main()
{
    float k;
    float celsius, fahrenheit;

    printf("Temperatur in Grad Celsius:");
    scanf("%f", &celsius);

    k = 9.0 / (float)5;

    fahrenheit = k * celsius + 32.0;

    printf("ergibt in Fahrenheit: %6.2f\n", fahrenheit);
}
```

b)

```
#include <stdio.h>

float umrechnen (float a,float k); // Funktionsdeklaration

int main( int argc, char *argv[] )
{
    int i;
    float k=9.0 / (float)5; //Muss man ja nicht jedesmal neu berechnen ...
    float ausg=0;
    printf("Hier sehen sie die Tabelle der Grad Celsius von -30 bis 100
      und die entsprechenden Fahrenheitwerte :\n");
    for (i=-30;i<=100;i=i+10){
        ausg=umrechnen((float)i,k);
        printf("%d Celsius ergibt in Fahrenheit: %6.2f\n", i,ausg);
    }
}


float umrechnen(float a,float k)
{
    float fahrenheit;
    fahrenheit = k * a + 32.0;
    return fahrenheit;
}
```

Aufgabe 2

a)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

# define ALPHABET 36
# define PWLAENGE 10
# define TEXTLAENGE 100

int main() {
    char pw[PWLAENGE+1];
```

```
char text[TEXTLAENGE+1];
char neu[TEXTLAENGE+1];
char choice[2];
int i=0, x=0, y=0, z=0;
int counter=0;

// 'tabelle' legt die zugehörigen Indizes zur Umwandlung fest
char tabelle[36] = {'A','B','C','D','E','F','G','H','I','J','K','L',
             'M','N','O','P','Q','R','S','T','U','V','W','X',
       'Y','Z','0','1','2','3','4','5','6','7','8','9'};

printf("Bitte geben Sie ein Passwort ein (max. %d Zeichen) :\n",
  PWLAENGE);
gets(pw);
strupr(pw); // umwandeln in Großbuchstaben
printf("Bitte geben Sie den Text ein (max. %d Zeichen) :\n",
  TEXTLAENGE);
gets(text);
strupr(text); // umwandeln in Großbuchstaben
printf("PW: %s Text: %s \n", pw, text);

printf("Wollen Sie verschluesseln [v] oder entschluesseln [e]? \n");
gets(choice);

// Auswahlmenü
switch(choice[0]) {
    case 'v': // verschlüsseln
        counter=0;
        i=0;
        for(x=0; x<strlen(text); x++) {
            if(text[x]==' ') continue; // Leerzeichen überspringen
            for(y=0; y<36; y++) { //nach Zeichen in der Tabelle suchen
                if(tabelle[y]==text[x]) {
                    for(z=0; z<36; z++) {
                        if(tabelle[z]==pw[i]) {
                           // Berechnung des Chiffrecodes
                           neu[counter]=tabelle[(y+z)%36];
                           break;
                        }
                    }
                    break;
                }
            }
            i=(i+1)%strlen(pw); // nächster Buchstabe des Codewortes
            counter++;
        }
        neu[counter]='\0';
        printf("Das kodierte Wort ist: %s", neu);
        break;

    case 'e': // entschlüsseln
        counter=0;
        i=0;
        for(x=0; x<strlen(text); x++) {
            for(y=0; y<36; y++) { //nach Zeichen in der Tabelle suchen
                if(tabelle[y]==text[x]) {
                    for(z=0; z<36; z++) {
                        if(tabelle[z]==pw[i]) {
                           // Berechnung des Originalwortes
                           neu[counter]=tabelle[(y-z+36)%36];
                           break;
                        }
                    }
                    break;
                }
            }
```

```
                i=(i+1)%strlen(pw); //nächster Buchstaben des Codewortes
                counter++;
            }
            neu[counter]='\0';
            printf("Das dekodierte Wort ist: %s", neu);
            break;

    }
}
```

Aufgabe 3

a)

In der Funktion Diff() werden a und b neu initialisiert und ausgegeben. Die Differenz wird jedoch zwischen den übergegebenen Werten, die in dieser Funktion die Bezeichnung p1 bzw. p2 haben, gebildet.
Zurückgegeben wird dann eben dieser Wert „d", der in der Adresse der Variable c aus der main-Prozedur gespeichert wird.
Die Ausgabe |main| gibt die Werte der lokalen Variablen a und b wieder, da diese globalen gegenüber Vorrang haben.
In der Ausgabe von |summe| hingegen gibt es keine lokalen Variablen a und b und daher werden die Werte der globalen Variablen ausgegeben.
|main| : a= 5, b= 6
|summe| : a= 1, b= 2, a+b= 11
|diff| : a= 4, b= 3, a-b= -1

- main : a= 5, b= 6
  Ausgabe der main-Prozedur. Aktuelle Werte für a und b sind die in main neu deklarierten. Die globalen Variablen a und b sind versteckt.
- summe : a= 1, b= 2
  Ausgabe der Prozedur „summe". Globale Variablen a und b sind sichtbar
- a+b= 11
  Ergebnis der Addition der in „main" lokalen Variablen a und b
- diff : a= 4, b= 3
  Die lokalen Variablen a und b in „diff" verdecken die globalen

b)

```
#include <stdio.h>

int anz = 10;
int max = 0;
int array[10] = {4, 6, 2, 0, 9, 1, 5, 7, 8, 3};

void exchange(int i1, int i2)
{
    int tmp;
    if(array[i1] > array[i2])
    {
        if (array[i1] > max)
            max = array[i1];
            tmp = array[i1];
            array[i1] = array[i2];
            array[i2] = tmp;
    }
    else
    {
        if(array[i2] > max)
            max = array[i2];
    }
}
```

```
main()
{
    int i, j;

    for(i=0; i<anz; i++)
    {
        for(j=i+1; j<anz; j++)
            exchange(i, j);
    }

    printf("Die Zahlen in sortierter Reihenfolge:");
    for(i=0; i<anz; i++)
        printf(" %d", array[i]);

    printf("\nDas Maximum: %d\n", max);
}
```

c)

```
#include <stdio.h>

int exchange(int i1, int i2,int *array,int max,int anz)
{
    int tmp;
    if(array[i1] > array[i2])
    {
        if (array[i1] > max)
            max = array[i1];
            tmp = array[i1];
            array[i1] = array[i2];
            array[i2] = tmp;
    }
    else
    {
        if(array[i2] > max)
            max = array[i2];
    }
    return max;
}


main()
{
    int anz = 10;
    int max = 0;
    int array[10] = {4, 6, 2, 0, 9, 1, 5, 7, 8, 3};

    int i, j;

    for(i=0; i<anz; i++)
    {
        for(j=i+1; j<anz; j++)
            max=exchange(i, j,array,max,anz);
    }

    printf("Die Zahlen in sortierter Reihenfolge:");
    for(i=0; i<anz; i++)
        printf(" %d", array[i]);

    printf("\nDas Maximum: %d\n", max);
}
```

Übung 2

Aufgabe 1

a)

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int quadGleichung(double a, double b, double c, double *l1, double *l2);

int main() {
    double l1,l2;
    float a,b,c;
    int anz=0;
    printf("Zu loesen ist die Gleichung ax^2+bx+c=0, a!=0\n");
    printf("Bitte geben Sie den Koeffizienten a ein: ");
    scanf("%f",&a);
    printf("Bitte geben Sie den Koeffizienten b ein: ");
    scanf("%f",&b);
    printf("Bitte geben Sie den Koeffizienten c ein: ");
    scanf("%f",&c);
    printf("a = %f, b = %f, c = %f\n", a,b,c);
    anz=quadGleichung(a,b,c,&l1,&l2) ;
    printf("Anzahl der Nullstellen: %d\n", anz);
    if (anz == 2)printf("Es gibt zwei reelle Loesungen: %f und %f
                  \n",l1,l2);
    else if (anz==1)printf("Es gibt nur eine reelle Loesung: %f\n",l1);
    else printf("Keine reelle Loesung !!!\n");

}

int quadGleichung(double a, double b, double c, double *l1, double *l2) {
    double d = b*b - 4*a*c;
    if (d > 0.){ // Diskriminante > 0 => 2 reelle Loesungen
        *l1 = (((-1.)*b - sqrt(d))/ ((2.)*a));
        *l2 = (((-1.)*b + sqrt(d))/ ((2.)*a));
        return 2;
    }
    else if (d == 0.){ // Diskriminante > 0 => 1 Loesung
        *l1 = (((-1.)*b - sqrt(d))/ ((2.)*a));
        return 1;
     }
    else { // Diskriminante < 0 => keine Loesung
        return 0;
    }
}
```

Aufgabe 2

a) *Was leistet das Programm?*

Das Programm liest eine Datei zeichenweise ein, wandelt die Buchstaben in Großbuchstaben um und speichert diese in der als zweiten Parameter angegebenen Zahl ab.

```
#include <stdio.h>

//argc = Anzahl der Aufrufparameter
int main(int argc, char **argv)
{
    FILE    *f1, *f2;
    char    buf[100];
    int     i;

    if(argc != 3) // weniger oder mehr als 3 Aufrufparameter
    {
        printf("usage: u1_4a <inputfiles> <outputfile>!\n");
        exit(-1);
    }

    f1 = fopen(argv[1], "rt");
    if(f1 == NULL) // kein Lesezugriff auf f1 möglich
    {
        printf("Error: Can't open '%s' for read!\n", argv[1]);
        exit(-1);
    }

    f2 = fopen(argv[2], "wt");
    if(f2 == NULL) // kein Schreibzugriff auf f2 möglich
    {
        printf("Error: can't open '%s' for write!\n", argv[2]);
        exit(-1);
    }

    while(!feof(f1)) // zeichenweises Durcharbeiten von f1 bis Ende
                  erreicht
    {
        fgets(buf, 100, f1); // Einlesen von 100 Zeichen aus f1
        for(i=0; i<strlen(buf); i++)
            if(islower(buf[i])) // buf[i] Kleinbuchstabe?
                buf[i] += 'A'-'a'; // Umwandlung in Großbuchstaben
        fputs(buf, f2); // Speichern des Buchstabens in f2
    }

    fclose(f1); // Schließen von f1
    fclose(f2); // Schließen von f2
    printf("Programm Ende!\n");
}
```

b) *Welche 3 Grundtypen gibt es, ein File mit fopen() zu öffnen? Worin unterscheiden sich diese Typen?*

Möglichkeit 1 : 'r' : öffnet die Datei zum Lesen (read)
Möglichkeit 2 : 'w' : öffnet die Datei zum (Über-)Schreiben (write)
Möglichkeit 3 : 'a' : öffnet die Datei zum Schreiben (Anfügen) (append)

```
f2 = fopen(argv[2], "at");
```
Hängt die Daten an das Fileende an.

c) Der Zugriffstyp 'rb' ermöglicht es Binärdateien zu behandeln im Gegensatz zum Typ 'rt' (Textdateien).
Dies ist natürlich nur sinnvoll auf Systemen, die zwischen Text- und Binärdateien unterscheiden.

Man sieht keine offensichtliche Veränderung der Funktionalität des Programmes, unter Windows wird nur die Codierung der Zieldatei verändert.

Aufgabe 3

b) *Welchen Vorteil hat die dynamische Erfassung der Studentendaten in einer linearen Liste gegenüber der statischen Erfassung in einem Array?*

Der Speicherplatz des Arrays muss bereits am Anfang reserviert werden und ist somit statisch, dh nicht mehr veränderbar im Laufe des Programmes.
-> maximale Anzahl der Daten vorbestimmt.
-> zu große Dimensionierung des Arrays belegt unnötigen Speicher.
Bei Verwendung einer linearen Liste wird der benötigte Speicherplatz im Laufe des Programmes angepasst, dh bei Anfügen wird neuer Speicher reserviert und bei Entfernen wieder freigegeben.
-> maximale Anzahl der Daten nicht vorbestimmt.
-> keine Dimensionierung vor Ablauf des Programmes nötig.

c) *Was hat diese Form der Referenzierung für Vorteile gegenüber einer Kopie der Liste mit anderer Sortierung?*

Es wird kein zusätzlicher Speicherplatz für die sortierte Studentenliste benötigt, da die neue Liste nur Verweise auf die alte Liste enthält.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


typedef struct stud_type_ {             /* Struktur des Datensatzes:    */
  int               matnum;             /* Matrikelnummer, Vor-         */
  char              vorname[20];        /* und Nachname sind Eintraege. */
  char              nachname[20];       /* Die Datenbank ist eine       */
  struct stud_type_ *next_student;      /* einfach verkettete Liste     */
} stud_type;

// implement
typedef struct sort_type_ {
    struct stud_type_ *student;
    struct sort_type_ *next_entry;
} sort_type;
// end implement

stud_type *studenten_liste = NULL;      /* Initialisierung der Datenbank*/
// implement
sort_type *sortier_liste = NULL;        /* Siehe Aufgabe 1c             */
// end implement

int is_empty()                          /* Ist die Datenbank leer?      */
{
  return (studenten_liste==NULL);
}


void enqueue(int matnum,                /* Einfuegen eines Elementes    */
             char vorname[20],
             char nachname[20])
{
  stud_type *curr,
            *next,
            *neuer_student;

  neuer_student = malloc(sizeof(stud_type)); /* Speicher fuer Element   */
  strcpy(neuer_student->vorname,vorname);    /* anfordern und mit Daten */
```

```
  strcpy(neuer_student->nachname,nachname);  /* fuellen                  */
  neuer_student->matnum = matnum;

  curr = studenten_liste;              /* Neues Element in DB einfuegen */
  if ((curr==NULL)||(curr->matnum>neuer_student->matnum))
  {
    neuer_student->next_student = curr; /* Sonderfall: an erster Stelle */
    studenten_liste = neuer_student;    /* einfuegen                    */
  }
  else
  {
    next = curr->next_student;          /* Durchmustern der DB          */
    while ((next!=NULL) && (next->matnum<neuer_student->matnum))
    {
      curr = next;
      next = next->next_student;
    }
    neuer_student->next_student = next; /* Neues Element nach curr und  */
    curr->next_student=neuer_student;   /* vor next einfuegen           */
  }
}


int dequeue(int matnum)                 /* Löschen eines Elementes      */
{
  stud_type *curr,*next;

  if (is_empty(studenten_liste))
    return(0);                          /* Rückgabewert: Fehler!        */

  curr=studenten_liste;
  if (curr->matnum==matnum) {           /* Sonderfall: erstes Element   */
    studenten_liste=curr->next_student; /* loeschen                     */
    free(curr);                         /* Speicher freigeben           */
    return(1);                          /* Rückgabewert: OK.            */
  }

  next=curr->next_student;              /* Durchmustern der DB          */
  while ((next!=NULL)&&(next->matnum<matnum)) {
    curr=next;
    next=next->next_student;
  }

  if ((next==NULL)||(next->matnum!=matnum))
    return(0);                          /* Rückgabewert: Fehler!        */
  curr->next_student=next->next_student;  /* Element loeschen           */
  free(next);                           /* Speicher wieder freigeben    */
  return(1);                            /* Rückgabewert: OK.            */
}


int get_student(int matnum,             /* Auslesen eines Elementes     */
                char vorname[20],
                char nachname[20]) {
  stud_type *curr;

  curr=studenten_liste;                 /* Durchmustern der DB          */
  while ((curr!=NULL)&&(curr->matnum<matnum)) curr=curr->next_student;
  if ((curr==NULL)||(curr->matnum!=matnum))
    return(0);                          /* Rückgabewert: Fehler!        */
  else {
    strcpy(vorname,curr->vorname);
    strcpy(nachname,curr->nachname);
    return(1);                          /* Rückgabewert: OK.            */
  }
}
```

```
// implement
void create_sortier_liste() {
    stud_type *curr;

    curr = studenten_liste;
    while(curr != NULL) {
        enqueue_sort(curr);
        curr = curr->next_student;
    }
}

void enqueue_sort (stud_type * student) {
    sort_type *curr;
    sort_type *next;
    sort_type *new_element;

    new_element = malloc(sizeof(sort_type));
    new_element->student = student;

    curr = sortier_liste;
    if((curr == NULL) || strcmp(student->nachname,curr->student->nachname) <0){
        new_element->next_entry = curr;
        sortier_liste = new_element;
    }

    else {
        next = curr->next_entry;
        while((next!= NULL) && (strcmp(student->nachname, curr->student-
               >nachname) >0)) {
            curr = next;
            next = next->next_entry;
        }
        new_element->next_entry = next;
        curr->next_entry = new_element;
    }
}

void save(char *filename) {
    stud_type *curr = studenten_liste;
    FILE *f = fopen(filename, "wt");
    if( f == NULL )
        printf("Can´t open file %s for write \n", filename);
    else {
        while (curr != NULL) {
            fprintf(f, "%s %s %d \n", curr->vorname, curr->nachname,
                    curr->matnum);
            curr = curr->next_student;
        }
        fclose(f);
    }
}

void load(char *filename) {
    char vorname[20], nachname[20];
    int matnum;
    FILE *f = fopen (filename, "rt");
    if (f == NULL)
        printf("Can´t open file %s for read \n", filename);
    else{
        studenten_liste = NULL;
        while (!feof(f)) {
            fscanf(f, "%s %s %d \n", vorname, nachname, &matnum);
            enqueue(matnum, vorname, nachname);
        }
        fclose(f);
```

```
    }
}

main() {                                /* Test der Listenfunktionen     */
  int matnum;
  char vorname[20],nachname[20];
  stud_type *curr;

  printf(">>> Fuege neuen Studenten in die Liste ein: Bill Clinton [6666] \n");
  enqueue(6666,"Bill","Clinton");

  printf("> Fuege neuen Studenten in die Liste ein: Hillary Clinton [4711]\n");
  enqueue(4711,"Hillary","Clinton");

  printf("> Fuege neuen Studenten in die Liste ein: Newt Gingrich [9999]\n");
  enqueue(9999,"Newt","Gingrich");

  printf("> Test, ob die Matrikelnummer 0815 bereits erfasst wurde\n");
  if (get_student(815,vorname,nachname))
    printf("    Matrikelnummer %4i: %s %s\n",815,vorname,nachname);
  else
    printf("    Matrikelnummer %4i ist unbekannt\n",815);

  printf("> Fuege neuen Studenten in die Liste ein: Monica Lewinsky [0815]\n");
  enqueue(815,"Monica","Lewinsky");

  printf("> Loesche die Matrikelnummer 4711 ...\n");
  if (dequeue(4711))
    printf("    Matrikelnummer %4i geloescht\n",4711);
  else
    printf("    Matrikelnummer %4i war nicht erfasst\n",4711);

  printf("> Test ob die Studentenliste leer ist ...\n");
  if (is_empty())
    printf("    Die Studentenliste ist leer \n");
  else
    printf("    Die Studentenliste ist nicht leer \n");

  printf("> Test, ob die Matrikelnummer 6666 bereits erfasst wurde ...\n");
  if (get_student(6666,vorname,nachname))
    printf("    Matrikelnummer %4i: %s %s\n",6666,vorname,nachname);
  else
    printf("    Matrikelnummer %4i ist unbekannt\n",6666);

  printf("> Loesche die Matrikelnummer 9998 ...\n");
  if (dequeue(9998))
    printf("    Matrikelnummer %4i geloescht\n",9998);
  else
    printf("    Matrikelnummer %4i war nicht erfasst\n",9998);

  printf("> Loesche die Matrikelnummer 9999 ...\n");
  if (dequeue(9999))
    printf("    Matrikelnummer %4i geloescht\n",9999);
  else
    printf("    Matrikelnummer %4i war nicht erfasst\n",9999);

  printf("> Gebe alle erfassten Studenten aus ...\n");
  curr = studenten_liste;

  while(curr!=NULL)
  {
    printf("    Matrikelnummer %4i: %s %s\n",curr->matnum,curr->vorname,
           curr->nachname);
    curr=curr->next_student;
  }
}
```

Übung 3

Aufgabe 1

a)

Prozess : Einheit des Ressourcebesitzes & Ausführung
- eigenen (virtuellen) Arbeitsraum
- Kontrolle über E/A, Hauptspeicher
- Ausführung verfolgt immer einen Weg (= trace) durch ein Programm

Thread : Einheit der Ausführung
- Ausführung innerhalb eines Prozesses
- Bewegung im selben Arbeitsraum
- Zugriff auf alle Betriebsmittel im Prozess
- keine Schutzmechanismen

b)

- Ausnutzung der parallelen Architektur
- Ressourcenverwaltung
- gemeinsames Nutzen von Ressourcen (Vorteil für Kommunikation zwischen 2 Threads)
- Antwortzeit der Threads untereinander

c)

- Drucken im Hintergrund
- Rechtschreibüberprüfung
- Bearbeitung der Eingabe durch den User
- etc.

Vorteil: schnelle Interaktion mit dem Benutzer
Nachteil: Inkonsistenz und möglicher Datenverlust

Aufgabe 2

a)

Erzeuger	**Produkt**	**Lager**	**Verbraucher**
Anwendungsprogramm	Druckjob	Druckqueue	Drucker(-manager)
Anwendungsprogramm	Programmdaten	Festplattenpuffer	Festplatte
Musikprogramm	Tondaten	Digital-Sound-Puffer	Soundkarte

b)

Da es keine Variable gibt, die von beiden Prozessen verändert werden kann, treten keine Konsistenzprobleme auf.

Erzeuger : IN
Verbraucher : OUT

Mehr als ein Verbraucher/Erzeuger:
Hinzufügen einer weiteren Variable „counter"

Bsp:

Erzeuger 1	Erzeuger 2	IN	counter	OUT	
while counter < ... lese IN		7	4	3	
	while counter < ... lese IN				
	IN:= IN+ 1 mod MAX	8	4	3	
IN:= IN+ 1 mod MAX counter:= counter+ 1		8	4	3	
	counter:= counter+ 1	8	6	3	FEHLER!

Aufgabe 3

a)

1) progress requirement verletzt

t	p_0	p_1	turn
1.	crit.		0
	turn= 1		1
	remain...		1
	stirbt		1
		crit.	1
		turn= 0	0
		remain...	0
7.		while(turn!=1)	0

2) mutual exclusion verletzt

t	p_0	p_1	flag[0]	flag[1]
1.	flag[0]:= false			
2.		flag[1]:= false		
3.				
4.				
5.				
6.				
7.				
8.				

3) progress requirement verletzt

t	p_0	p_1	flag[0]	flag[1]
1.	flag[0]:= flase		F	
2.		flag[1]:= false	F	F
3.	flag[0]:= true		T	F
4.		flag[1]:= true	T	T
5.	while(flag[1])		T	T
6.		while(flag[0])	T	T

1.+2. = init

(Nebenbemerkung: *progress requirement verletzt ⇒ bounded waiting verletzt*
umgekehrt muss die Folgerung nicht gelten!)

b)

- first come, first served : mutual exclusion verletzt
  progress requirement verletzt

- sperren : progress requirement verletzt
- Ampeln : korrekte Lösung
- Vorrangrichtung : bounded waiting verletzt (progress requirement verletzt)

c)

```
Bool SensorA;     //true, falls Auto auf Seite A festgestellt wird
Bool SensorB;     //true, falls Auto auf Seite B festgestellt wird
Bool AmpelB;      //true, falls Ampel B grün ist
void toggle();    //schaltet grüne Ampel auf gelb-> rot, andere auf grün
const int Max;    //gibt an, nach wievielen Iterationen zwangsweise
                    umgeschaltet wird
int counter;

counter:= 0;
repeat
      if(SensorA AND !SensorB AND AmpelB) then {
            counter:= 0;
            toggle();
      }

      if(!SensorA AND SensorB AND !AmpelB) then {
            counter:= 0;
            toggle();
      }

      counter:= counter+ 1;

      if(counter == Max) then {
            counter:= 0;
            toggle();
      }
until false
```

Übung 4

Aufgabe 1

2) bounded waiting: wird nicht eingehalten
- P_i betritt UKB
- P_j betritt KB
- P_i findet `turn=j` und `flag[j]= true` und bleibt stehen (`noop`)
- P_j verlässt KB, stellt `flag[j]:= false`, durchläuft UKB stellt `flag[j]:= true`
- P_i findet `flag[j]= true` und muss warten

3) progress requirement: wird eingehalten
- sobald für einen Prozess `flag == true`, kann anderer Prozess nicht mehr `turn` ändern.

i. `turn= i` und P_i betritt als erster die `while`-Schleife, so gelangt P_i immer in seinen KB, da P_j `turn` nicht ändern kann

ii. `turn= j` und P_i betritt als erster die `while`-Schleife, so hat er seinen `flag` auf `true` gesetzt.
Entweder setzt P_i dann `turn= i` und betritt KB oder P_j überholt P_i und P_j gelangt in den KB

Aufgabe 2

a)

```
while (testinc(lock))
      noop;

critical section;

lock:= 0;
uncritical section;
```

b)

```
signal(S) {
      if(testinc(S) == -1)
           wecke ältesten Prozess aus Warteschlange S und lasse ihn in den KB
}

wait(S); {
      if(dectest(s) == -1)
           halte Prozess an und ordne ihn in S ein
}
```

Aufgabe 3

a)

```
receive(A, message A);
receive(B, message B);
```

b)

```
int childA, childB;
shared char *messageA= NULL;
shared char *messageB= NULL;

childA = fork();

if(childA != 0)
      childB= fork();

if(childA == 0)
      receive(A, messageA);
else if (childB == 0)
      receive(B, messageB);

else {
      while((messageA == NULL) && (messageB == NULL))
            noop;
}

kill(childA);
kill(childB);
```

c)

```
send(A, sentMessage);
receive(A, receivedMessage);

if(receivedMessage == sentMessage) {
      send(A, message);
}
```

Ansatz ist nicht korrekt!

- war Mailbox nicht leer, so wurde sentMessage ohne Bedeutung abgelegt
- erste Nachricht wurde aus Mailbox entfernt

Aufgabe 4

- Konflikte über globale Variablen
- Ablaufabhängigkeit von Threads
- Effizienzprobleme

Aufgabe 5

Nachrichten:

```
const wait= 0;
const signal= 1;

void wait() {
      int msg;
      send(sync, signal);
      receive(sync, msg);
}

void signal() {
      send(sync, wait);
}
```

```
void sync() {
      int quelle;
      int msg;
      int counter= 1;
      int dummy= 0;

      while(true) {
            receive(&quelle, &msg);
            if(msg == signal) {
                  if(counter <= 0)
                        enqueue(quelle);
                  else send(quelle, dummy);
                  counter--;
            }
            if(msg == wait) {
                  if (counter < 0) {
                        send(dequeue(), dummy);
                        counter++;
                  }
            }
      }
}
```

Übung 5

Aufgabe 1

a) Es werden für jede Tür benötigt:
- ein Sensor, ob jemand die Türklinke drückt (sprich die Tür öffnen will), jeweils von innen und außen
- ein Sensor, der sagt ob jemand im Zwischenraum ist
- ein Sensor, der bei einem Banküberfall durch Angestellte betätigt wird
- ein Relais zum Öffnen / Schließen der Tür.

b) Steuerungsprogramm:

Es gelte, dass man nicht im Innenraum die Richtung wechseln kann.

```
Bool Sensor_BankRobbery // TRUE wenn Banküberfall
Bool Sensor_Innenraum   // TRUE wenn jemand im Schleusenraum steht
Bool Sensor_T1_A        // TRUE wenn jemand von draußen die 1. Türklinke drückt
Bool Sensor_T1_B        // TRUE wenn jemand aus der Schleuse nach draußen will
Bool Sensor_T2_A        // TRUE wenn jemand aus der Schleuse in die Bank will
Bool Sensor_T2_B        // TRUE wenn jemand aus der Bank in die Schleuse will
Bool Tür_A              // TRUE wenn Tür nach draußen offen ist
Bool Tür_B              // TRUE wenn Tür zur Bank offen ist

void toggle()           // öffnet geschlossene Tür und schließt die andere

init (S, 1);            // Innenraumsemaphor

/**
 * Tür-A:
 **/
repeat

      /* Bei Banküberfall -> Türen sperren */
      while (Sensor_BankRobbery)
            noop;


      /* Möchte ein Kunde von außen herein, Tür nach außen ist geschlossen */
      if (Sensor_T1_A & !Tür_A ) then begin
            wait(S);    // warte bis Innenraum frei ist
            // Innenraum ist frei
            // Überprüfe, ob Tür noch verschlossen
            if(!Tür_A) then toggle();
            /* andernfalls war jemand im Innenraum und ist heraus gelassen
             * worden, also ist die Tür offen */
            toggle();
            // Kunde betritt Innenraum
            while(!Sensor_T2_A) // so lange Türklinke zur Bank nicht gedrückt
                  noop;
            toggle();
            signal(S);
      end;


      /*  Möchte ein Kunde in den Mittelraum von außen, die Tür ist offen */
      if(Sensor_T1_A & Tür_A) then begin
            wait(S);
            // Kunde betritt den freien Innenraum
```

```
            // warten bis er die andere Tür öffnen will
            while (!Sensor_T2_A)
                  noop;
            toggle();
            signal(S);
      end;

until FALSE;
```

Entsprechendes gilt für ein Programm Tür-B, wo jeweils die Sensoren vertauscht werden (A -> B im gesamten Programm).

c) Es wird ein weiterer Semaphor B für die Bank gebraucht, der mit der max. Anzahl an Personen n initialisiert wird.

Das modifizierte Programm für z.B. Tür_A ist dann:

```
init(B,n);

repeat

      /* Möchte ein Kunde von außen herein, die Tür nach außen ist zu */
      if (Sensor_T1_A & !Tür_A ) then begin
            wait(B); // warte bis Platz in der Bank ist
            wait(S); // warte bis Innenraum frei ist
            // Innenraum ist frei
            // überprüfe, ob Tür noch verschlossen ist
            if(!Tür_A) then toggle();
            /* Andernfalls war jemand im Innenraum und ist heraus gelassen
             * worden, also ist die Tür offen */
            toggle();
            // Kunde betritt Innenraum
            while(!Sensor_T2_A) // so lange Türklinke zur Bank nicht gedrückt
                  noop;
            toggle();
            signal(S);
      end;



      /* Möchte ein Kunde in den Mittelraum von außen, die Tür ist offen */
      if(Sensor_T1_A & Tür_A) then begin
            wait(B); // warte bis in der Bank Platz ist
            wait(S); // Warte bis Innenraum frei ist
            // Kunde betritt den freien Innenraum
            // warten bis er die andere Tür öffnen will
            while (!Sensor_T2_A)
                  noop;
            toggle();
            signal(S);
      end;

until FALSE;
```

Bei der Tür nach außen (Tür B) entfällt das 'wait(B);'
Dafür kommt vor jedem signal(S); ein signal(B);

Aufgabe 2

```
program automat {

	semaphore counter;
	semaphore druckerfrei;
	semaphore drucker;

	int einwurf;
	int geld;
	int wechsel;


	void geld() {
		while(true) {
			geld= 0;
			while(geld < preis) {
				einwurf= einwurf();
				for(int i= 1; i<= einwurf; i++)
					signal(counter);
				geld+= einwurf;
			}
			wechsel(geld- Preis);
			for(int i= 1; i<= geld- Preis; i++)
				wait(counter);
			geld= 0;
			wait(druckerfrei);
			signal(drucker);
		}
	}

	void drucker() {
		while(true) {
			signal(druckerfrei);
			wait(drucker);
			for(int i= 1; i<= Preis; i++)
				wait(counter);
			druckerKarte();
		}
	}


	void main() {
		init(counter, 0);
		init(druckerfrei, 0);
		init(drucker, 0);
		geld();
		drucker();
	}

}
```

Aufgabe 3

a)

```
n_WO entspricht wo_count
n_OW entspricht ow_count
n_climb entspricht climb_count


void ape_ow() {
      wait(mutex);
      if((dir == WO) OR ((dir == OW AND n_WO > 0))
            n_OW++;
            signal(mutex);
            wait(s_OW);
            wait(mutex);
      }

      dir= OW;
      n_climb++;
      signal(mutex);
      climb();
      wait(mutex);
      n_climb--;

      if(n_climb == 0) {
            dir= Empty;
            if(n_OW == 0)
                  for(; n_WO; n_WO—)
                        signal(s_WO);
            else {
                  n_WO--;
                  signal(s_WO);
            }
      }

      signal(mutex);
}
```

b)

```
mutex= 1;
s_WO= 0;
s_OW= 0;
```

Übung 6

Aufgabe 1

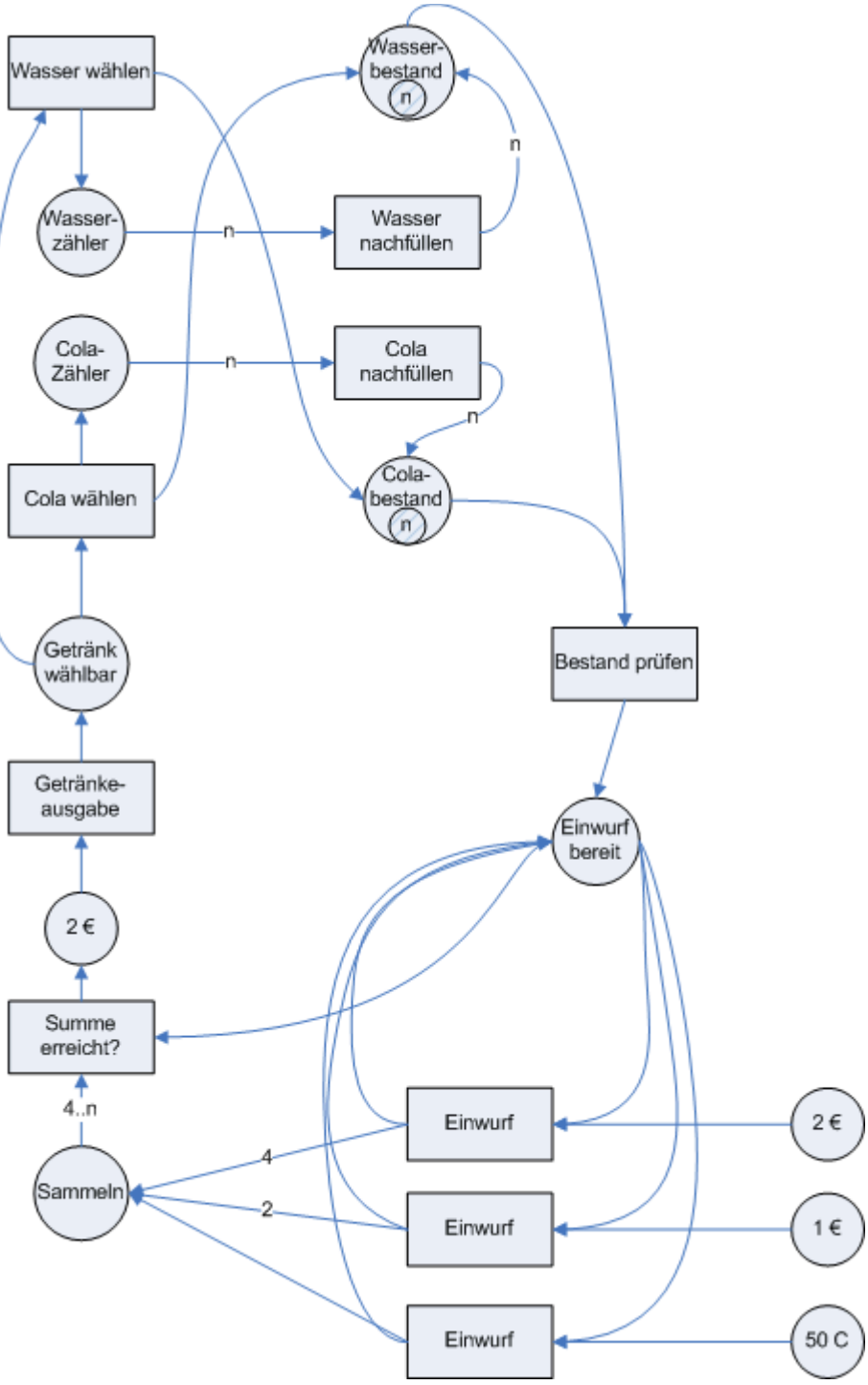
```
semaphore waiting[4];   // init(waiting,{0,0,0,0});
semaphore Kreuzung;     // init(Kreuzung, 1);
semaphore mutex;        // init(mutex, 1);

init nr_wait[4]= {0,0,0,0};

void auto(int pos) {
      int next= post+ 1;
      int prev= pos- 1;
      wait(mutex);
      nr_wait[pos]++;

      while((next != 4) && (nr_wait[next] > 0)) {
            signal(mutex);
            wait(waiting[pos]);
            wait(mutex);
      }

      signal(mutex);
      wait(Kreuzung);
      fahre();
      signal(Kreuzung);
      nr_wait[pos]--;

      if((prev != 0) && (nr_wait[prev] > 0)) {
            for(int i= 0; i < nr_wait[prev]; i++)
                  signal(waiting[prev]);
      }
}
```

Aufgabe 2

```
const Bridge_free= 0;
const OM= 1;
const MO= 2;

int current_direction;
int bridge_counter;
int o_count;
int m_count;

semaphore mutex;
semaphore s_om, s_mo;

void überquereBrücke(int Richtung) {
      if(Richtung == OM) {
            if((current_direction != Richtung) OR
              ((current_direction == Richtung) && (m_count > 0))) {
                  o_count++;
                  wait(s_om);
            }
            else
                  kein Warten
      }
      else {
            if((current_direction != Richtung) OR
              ((current_direction == Richtung) AND (o_count > 0))) {
```

```
                m_count++;
                wait(s_mo);
        }
    }

    current_direction= Richtung;

    if((Richtung == OM) && (o_count > 0))
        o_count--;

    if((Richtung == MO) && (o_count > 0))
        m_count--;

    bridge_counter++;
    go_over_the_bridge();
    bridge_counter--;

    if(bridge_counter == 0) {
        current_direction= Bridge_free;
        if((m_count == 0) && (o_count > 0)) {
            for(int i= 0; i < o_count; i++)
                signal(s_ow);
        }
        else
            signal(s_mo);
    else {
        current_direction= Bridge_free;
        if((o_count == 0) && m_count > 0) {
            for(int i= 0; i < m_count; i++)
                signal(s_mo);
        }
        else
            signal(s_om);
    }
}
```

Aufgabe 3

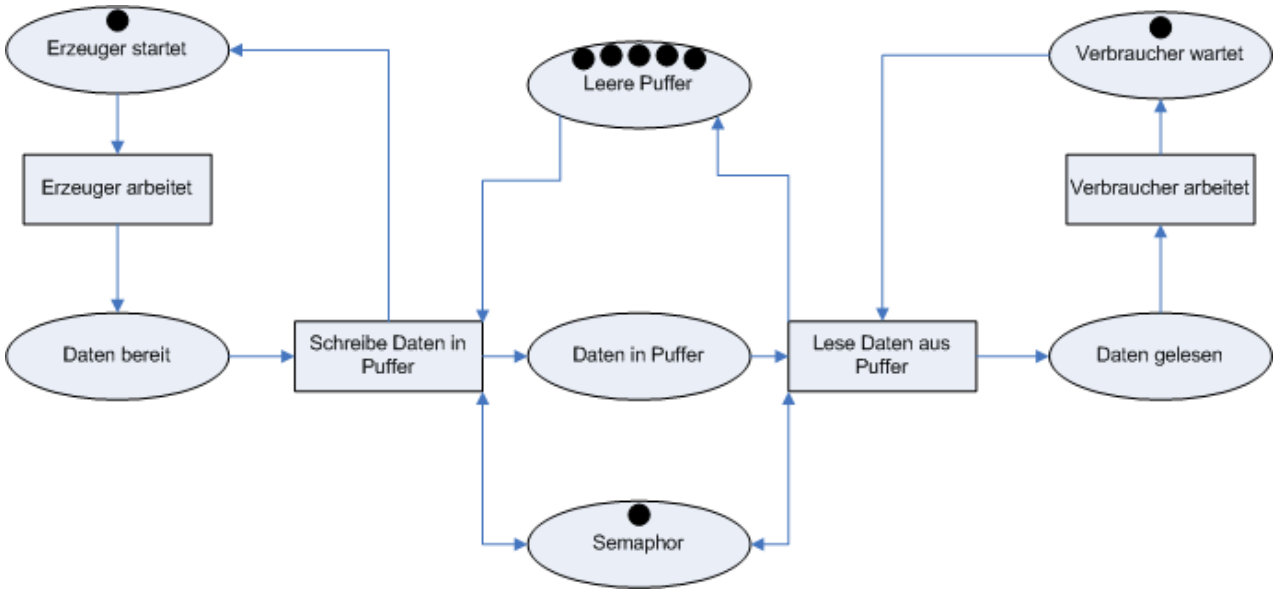

Wasser wählen
Wasser-bestand
n
Wasser-zähler
n
Wasser nachfüllen
n
Cola-Zähler
n
Cola nachfüllen
n
Cola wählen
Cola-bestand
n
Getränk wählbar
Bestand prüfen
Getränke-ausgabe
Einwurf bereit
2 €
Summe erreicht?
4..n
Einwurf
2 €
4
Sammeln
2
Einwurf
1 €
Einwurf
50 C

Übung 7

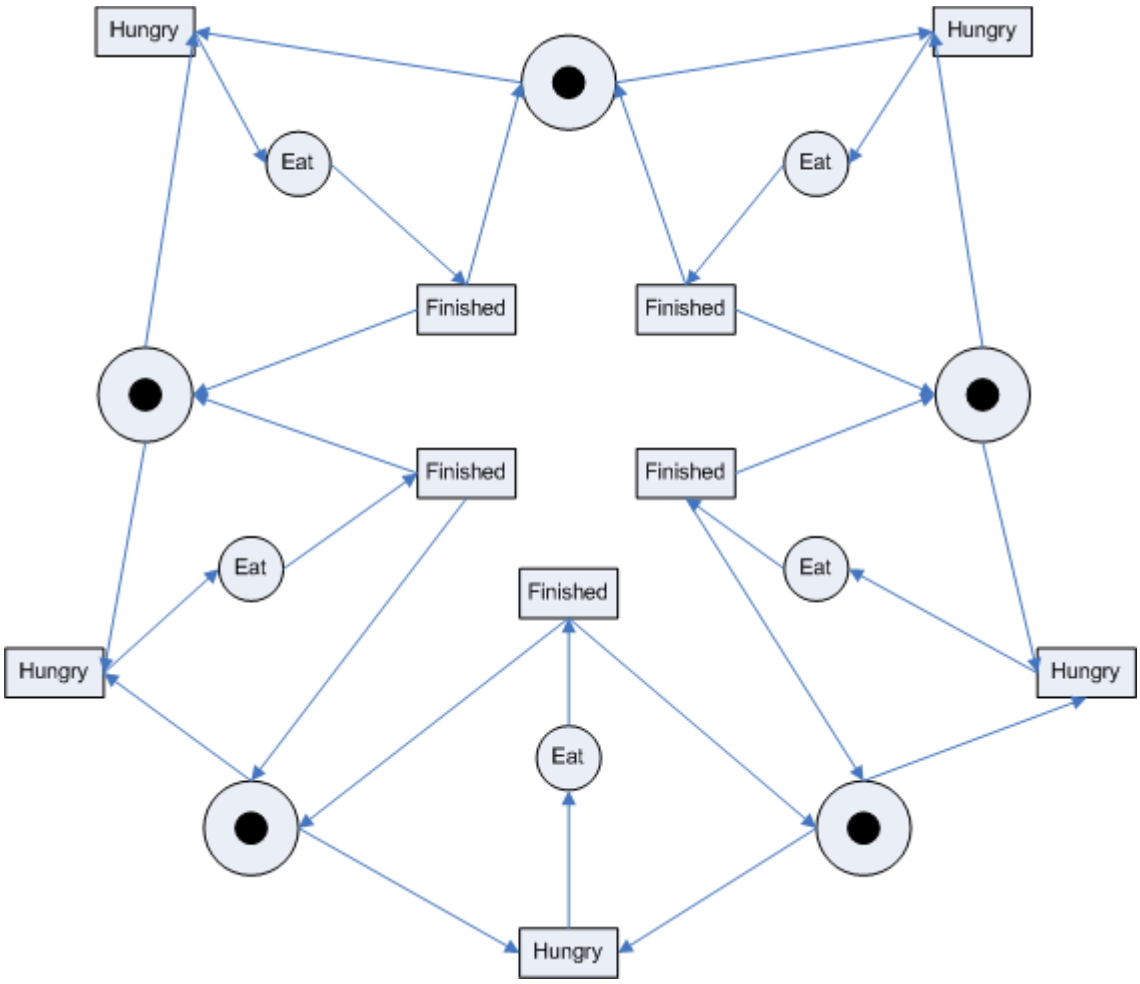
Aufgabe 1

```
Monitor FileCoordinator {

      int count;
      condition c;

      void acquire(int processID) {
            while((count + processID) >= n)
                  c.wait();
            count+= processID;
      }

      void release(int processID) {
            count-= processID;
            c.signal();
      }

      init {
            count= 0;
      }
}
```

Aufgabe 2



Aufgabe 3


Hungry
Hungry
Eat
Eat
Finished
Finished
Finished
Finished
Eat
Eat
Finished
Hungry
Hungry
Eat
Hungry

Übung 8

Aufgabe 1

```
Monitor SensorSystem
{
      condition c1, // sind weniger als 100 Elemente im Array?
      condition c2; // sind genau 100 Elemente im Array?
      int results [100];
      int numData;

      void saveResult (int x)
      {
            while (numData == 100) {
                  c2.signal ( );
                  c1.wait ( );
            }

            results [numData]= x;
            numData++;

      c1.signal();
      }

      void analyseData()
      {
            while (numData < 100) {
                  c2.wait();
            }
            Datenanalyse (results);
            numData= 0 ;

            c1.signal();
      }

      init {
            count= 0;
      }
}
```

Aufgabe 2

Bedingung 1: Die belegten bzw. angeforderten Betriebssmittel können nur exklusiv benutzt werden. Werden alle Betriebsmittel nicht exklusiv zugeteilt, so kann offenbar keine Verklemmung entstehen (genaugenommen gilt dies sogar, solange nur ein Betriebsmittel exklusiv genutzt wird). Andererseits ist ein unkontrollierter gemeinsamer Zugriff etwa auf einen Drucker nicht sinnvoll. Daher wird bisweilen Spooling vorgeschlagen, um ”exklusive“ Betriebsmittel logisch gemeinsam nutzen zu können. Spooling funktioniert jedoch nicht bei allen Betriebsmitteln gleichermaßen (z.B. CPU) und zudem kann der Wettbewerb um Plattenplatz für das Spooling seinerseits auch zu Verklemmungen führen. Die grundsätzliche Idee, die Betriebsmittelanforderungen auf möglichst wenige Prozesse zu reduzieren, ist jedoch vielfach anwendbar.

Bedingung 2: Prozesse sind bereits im Besitz von Betriebssmitteln, während sie weitere anfordern. Eine Möglichkeit, diese ”Haltebedingung“ auszuhebeln, besteht darin, dass Prozesse alle ihre Betriebsmittel atomar anfordern. Das grundlegende Problem besteht wiederum darin, dass viele Prozesse im vorhinein nicht exakt abschätzen können, welche und wieviele Betriebsmittel sie benötigen (andernfalls könnte der Bankers-Algorithmus eingesetzt

werden). Zudem werden die Betriebsmittel bei einem solchen Ansatz im allgemeinen vergleichsweise schlecht ausgenutzt. Eine Variante dieses Ansatzes besteht darin, dass ein Prozess vor der Anforderung eines weiteren Betriebsmittels zunächst alle seine Betriebsmittel freigibt und anschließend alle benötigten erneut atomar anfordert. Dies erfordert jedoch oftmals einen enormen Mehraufwand, etwa wenn kritische Abschnitte freigegeben und vorher zunächst in einen konsistenten Zustand überführt werden müssen.

Bedingung 3: Belegte Betriebsmittel können nicht zwangsweise Prozessen entzogen werden. Zwangsweises Entziehen von Betriebssmitteln (Preemption) ist ebenfalls im allgemeinen wenig erfolgversprechend. Einem Prozess beispielsweise den Drucker inmitten eines laufenden Druckjobs zu entziehen, erscheint wenig sinnvoll.

Bedingung 4: Es existiert eine zirkuläre Wartesituation zwischen den Prozessen. Die zirkuläre Wartebedingung kann auf zahlreiche Arten durchbrochen werden. Eine Möglichkeit besteht darin, dass jeder Prozess zu einem Zeitpunkt nur maximal ein Betriebsmittel besitzen darf. Praxistauglich ist dieser Ansatz jedoch nur in seltenen Fällen. Alternativ kann man alle Betriebsmittel einer totalen Ordnung unterwerfen, wobei Prozesse Betriebsmittel nur aufsteigend nach dieser Ordnung anfordern/belegen dürfen. Auf diese Weise werden Zyklen im Betriebsmittelbelegungsgraph ausgeschlossen. Es ist jedoch vielfach schwierig, eine für alle Anwendungen geeignete Totalordnung der Betriebsmittel zu finden.

Aufgabe 3

a) Zur Beantwortung der Frage muß geklärt werden, ob die aktuelle Ressourcenbelegung eines Prozesses (Zeilen von $H(t)$) zusammen mit den noch zu stellenden Anforderungen des Prozesses (Zeilen von $Q(t)$) den Maximalvorrat (Vektor M) nicht überschreitet, d.h. also, es muß geprüft werden, ob für alle $i \in \{1,\ldots,6\}$ gilt: $(H(t)+Q(t))_i \leq M$, wobei (hier und im folgenden) mit $(A)_i$ die i -te Zeile der Tabelle A bezeichnet werde. Hier also:

$H(t)+Q(t)$ =

	BM_1	BM_2	BM_3	BM_4
P_1	2	11	20	6
P_2	5	12	18	6
P_3	6	15	12	4
P_4	2	7	6	3
P_5	3	15	22	4
P_6	4	8	18	2

$H(t)+Q(t) \leq (8,15,24,6)$ für alle Zeilen

Die Anforderungen sind also realisierbar.

b) Restvorrat zum Zeitpunkt t:

$$V(t) = M - \sum_{i=6}^{6} (H(t))_i = (2,4,7,4)$$

1. Alle sechs Zeilen von H sind unmarkiert, da es kein $i \in \{1,\ldots,6\}$ gibt mit $(H(t))i = 0$. Es ist $W = V(t) = (2,4,7,4)$.

2. Suche Zeile i mit $Q_i \leq W$. Die vierte Zeile erfüllt dieses Kriterium. Die Anforderungen von P_4 können also erfüllt werden.

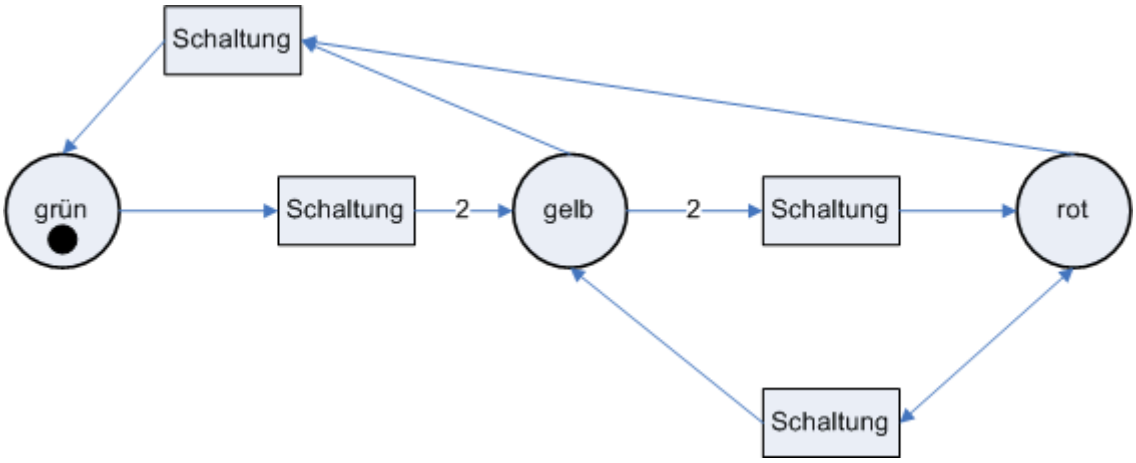
3. Annahme: P_4 kann zu Ende geführt werden. Nach seiner Beendigung gibt er seine Betriebsmittel wieder frei:

$$W = W + (H(t))_4 = (2,4,7,4) + (0,3,0,1) = (2,7,7,5)$$

⇒ Markierung der vierten Zeile.

4. Danach können keine Anforderungen mehr erfüllt werden, da es keine Zeile i in Q mehr gibt mit $Q_i \leq W$. Da noch unmarkierte Zeilen in H vorhanden sind, liegt ein Deadlock vor.

Aufgabe 4



Übung 9

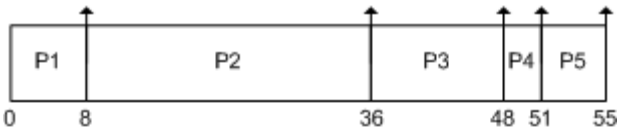
Aufgabe 1

```
Monitor Boxengasse {
      int         willFahren,
                  willArbeiten,
                  arbeiter;

      condition   fahrtEnde,
                  arbeitEnde;


      void fahrtBeginn(void) {
            willFahren++;
            if(arbeiter > 0)
                  arbeitEnde.wait();
      }

      void fahrtEnde(void) {
            willFahren--;
            while(willArbeiten > 0)
                  fahrtEnde.signal();
      }

      void arbeitBeginn(void) {
            willArbeiten++;
            if(willFahren > 0)
                  fahrtEnde.wait();
            arbeiter++;
            willArbeiten—-;
      }

      void arbeitEnde(void) {
            arbeiter--;
            if(arbeiter == 0)
                  arbeitEnde.signal();

      }

      init {
            willFahren= 0;
            willArbeiten= 0
            arbeiter= 0;
            init(fahrtEnde, 1);
            init(arbeitEnde, 1);
      }
```

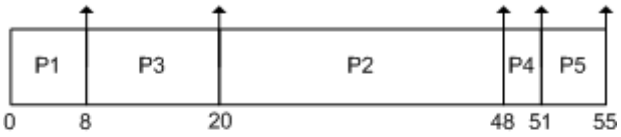
Aufgabe 2

a)

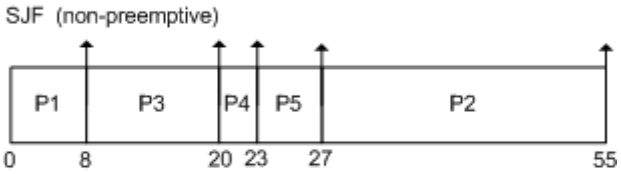
FIFO (non-preemptive)



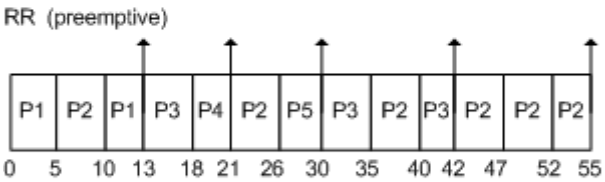
HPF (non-preemptive)



SJF (non-preemptive)



RR (preemptive)



HPF (preemptive)

SJF (preemptive)

b)

FIFO

Prozess	Ankunftszeit	Abgangszeit	Verweilzeit	Bedienzeit	Wartezeit
1	0	8	8	8	0
2	3	36	33	28	5
3	7	48	41	12	25
4	9	51	42	3	39
5	15	55	40	4	36
Mittelwerte			32,8		21,8

HPF

Prozess	Ankunftszeit	Abgangszeit	Verweilzeit	Bedienzeit	Wartezeit
1	0	8	8	8	0
2	3	48	45	28	17
3	7	20	13	12	1
4	9	51	42	3	39
5	15	55	40	4	36
Mittelwerte			29,6		18,6

SJF

Prozess	Ankunftszeit	Abgangszeit	Verweilzeit	Bedienzeit	Wartezeit
1	0	8	8	8	0
2	3	55	52	28	24
3	7	20	13	12	1
4	9	23	14	3	11
5	15	27	12	4	8
Mittelwerte			19,8		8,8

RR (Zeitscheibenwert 5)

Prozess	Ankunftszeit	Abgangszeit	Verweilzeit	Bedienzeit	Wartezeit
1	0	13	13	8	5
2	3	55	52	28	24
3	7	42	35	12	23
4	9	21	12	3	9
5	15	30	15	4	11
Mittelwerte			25,4		14,4

HPF (preemptiv)

Prozess	Ankunftszeit	Abgangszeit	Verweilzeit	Bedienzeit	Wartezeit
1	0	55	55	8	47
2	3	43	40	28	12
3	7	19	12	12	0
4	9	46	37	3	34
5	15	50	35	4	31
Mittelwerte			35,8		24,8

SJF (preemptiv)

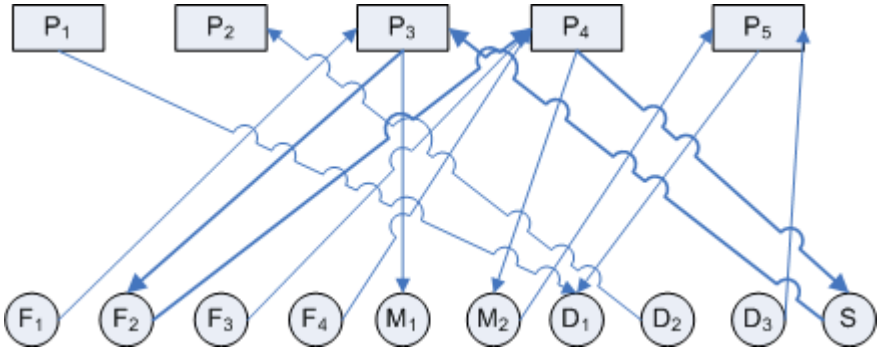
Prozess	Ankunftszeit	Abgangszeit	Verweilzeit	Bedienzeit	Wartezeit
1	0	8	8	8	0
2	3	55	52	28	24
3	7	27	20	12	8
4	9	12	3	3	0
5	15	13	4	4	0
Mittelwerte			17,4		6,4

Aufgabe 3

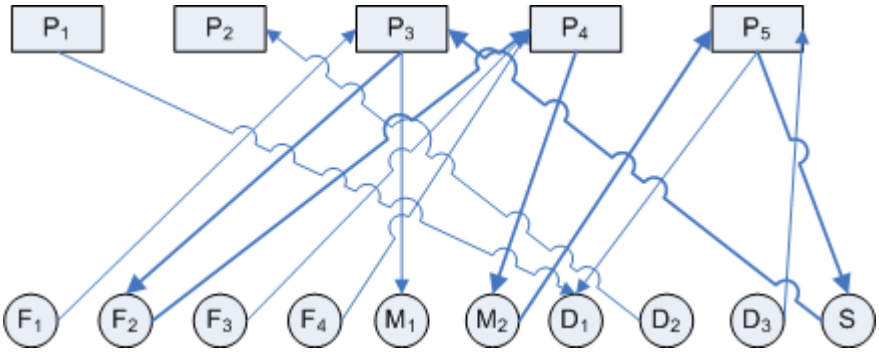
b) nein, weil von Prozess A keine Geräte benutzt werden

c) nein, weil Prozess B nur einen Drucker verwendet

d) Deadlock



e) Deadlock



Übung 10

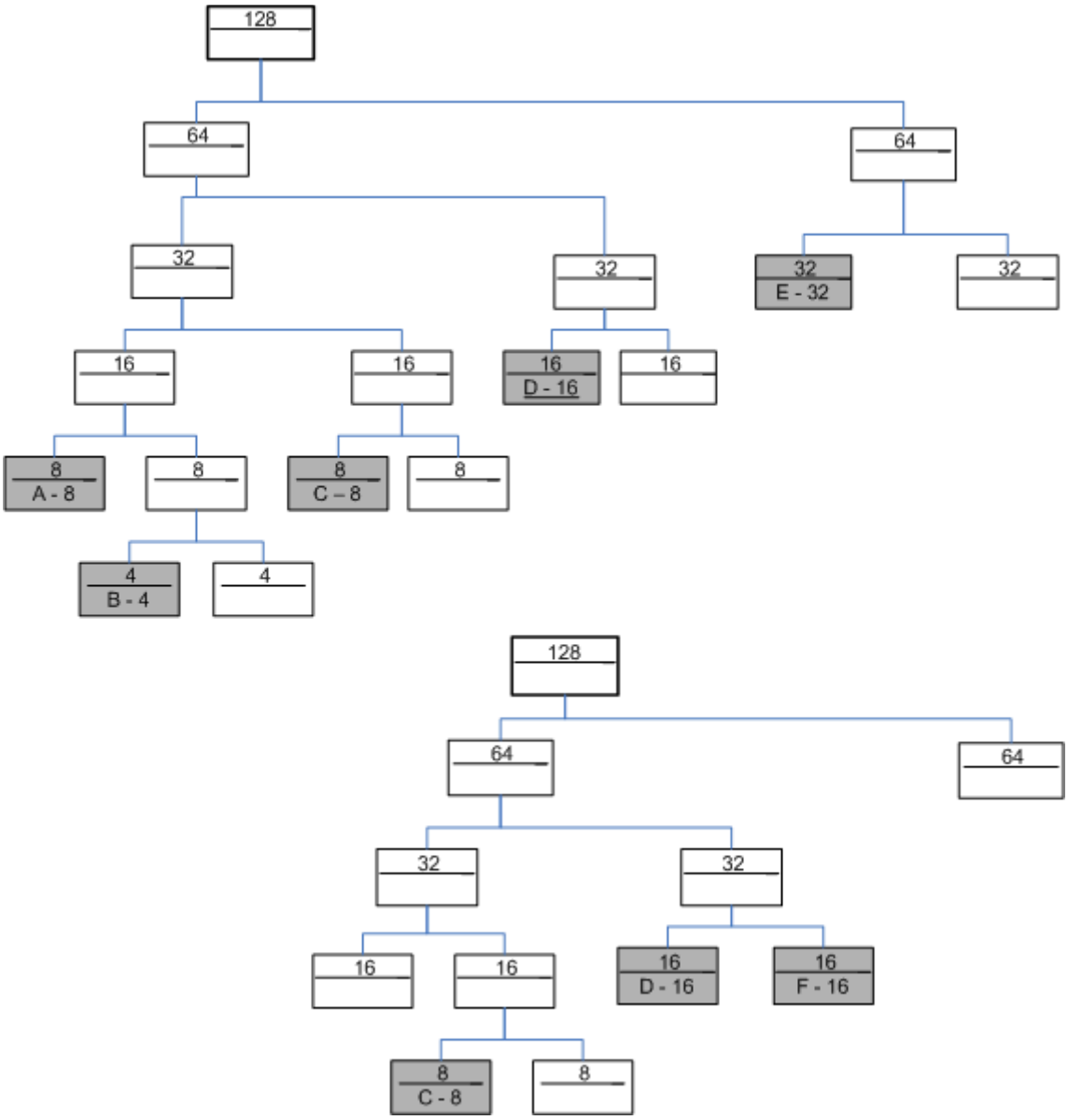
Aufgabe 1

t=0: A(4), B(7)

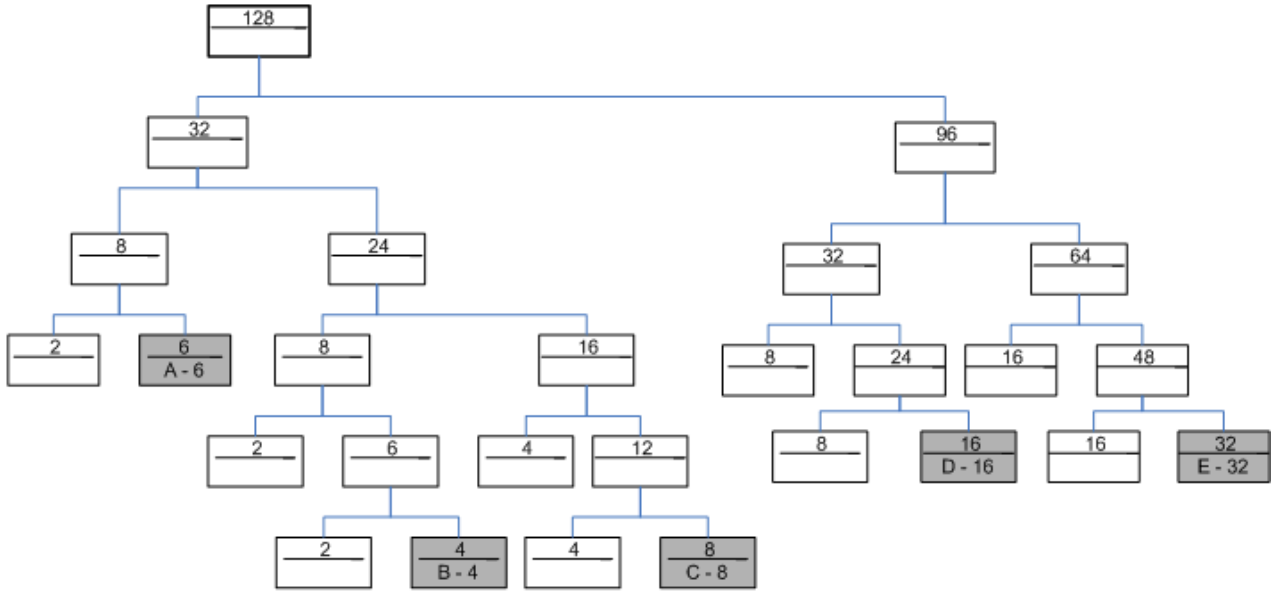
t	Klasse 0	Klasse 1	Klasse 2	Klasse 3	Event
1	A(3)	B(7)			C(1)
2	C(0)	B(7), A(3)			D(4)
3		B(6), A(3), D(4)			
4		B(5), A(3), D(4)			
5		B(4), A(3), D(4)			E(20)
6		B(3), A(3), D(4)		E(20)	
7		A(2), D(4)	B(3)	E(20)	
8		A(1), D(4)	B(3)	E(20)	
9		A(0), D4	B(3)	E(20)	
10		D(3)	B(3)	E(20)	F(3)
11		D(2), F(3)	B(3)	E(20)	
12		D(1), F(3)	B(3)	E(20)	
13		D(0), F(3)	B(3)	E(20)	G(2)
14	G(1)	F(3)	B(3)	E(20)	
15		F(2), G(1)	B(3)	E(20)	
16		F(1), G(1), H(3)	B(3)	E(20)	H(3)
17		F(0), G(1), H(3), I(3)	B(3)	E(20)	I(3)
18		G(0), H(3), I(3)	B(3)	E(20)	
19		H(2), I(3)	B(3)	E(20)	
20		**H(1), I(3)**	**B(3)**	**E(20)**	

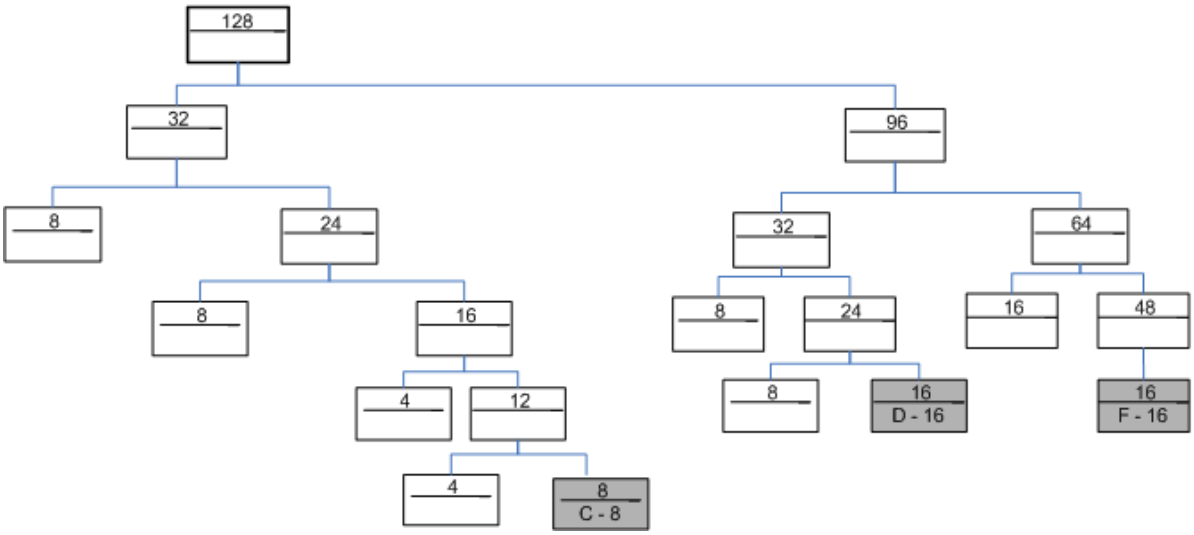
Aufgabe 2

a)


128
64
64
32
32
32
E - 32
32
16
16
16
D - 16
16
8
A - 8
8
8
C – 8
8
4
B - 4
4
128
64
64
32
32
16
16
16
D - 16
16
F - 16
8
C - 8
8

b)





Aufgabe 3

a)

$$\sum_{i=0}^{4} l_i = 1220$$

b)

Segment	Basis	Länge	Anfang phys. Speicher	Ende phys. Speicher
0	1410	365	1410	1774
1	400	70	400	469
2	500	120	500	619
3	630	515	630	1144
4	1145	150	1145	1294

kleinste Speicheradresse: 400
größte Speicheradresse: 1774

c)

1. 762 = 630 + 132 entspricht (3, 132)
2. 1145 = 1145 + 0 entspricht (4, 0)
3. 1146 = 1145 + 1 entspricht (4, 1)
4. 485 => nicht im System vorhanden, weil außerhalb des Speichers

Aufgabe 4

a) FIFO

	1	**5**	**2**	**4**	**5**	**3**	**4**	**5**	**2**	**4**	**2**	**1**	**4**	**3**	**Summe**
S1	1	5	2	4	4	3	3	5	2	4	4	1	1	3	
S2	-	1	5	2	5	4	4	3	5	2	2	4	4	1	
S3	-	-	1	5	2	2	2	4	3	5	5	2	2	4	
	x	**x**	**x**	**x**	**x**			**x**	**x**	**x**		**x**		**x**	**10**

b) LRU

	1	**5**	**2**	**4**	**5**	**3**	**4**	**5**	**2**	**4**	**2**	**1**	**4**	**3**	**Summe**
S1	1	5	2	4	5	3	4	5	2	4	2	1	4	3	
S2	-	1	5	2	4	5	3	4	5	2	4	2	1	4	
S3	-	-	1	5	2	4	5	3	4	5	2	4	2	1	
	x	**x**	**x**	**x**		**x**			**x**			**x**		**x**	**8**

c) LIMB

	1	**5**	**2**	**4**	**5**	**3**	**4**	**5**	**2**	**4**	**2**	**1**	**4**	**3**	**Summe**
S1			1	1	5	5	5	5	5	5	5	1	1	1	
S2		1	5	5	1	1	1	1	1	1	1	5	5	5	
S3	1	5	2	4	4	3	4	4	2	4	2	2	4	3	
	x	**x**	**x**	**x**		**x**	**x**		**x**	**x**	**x**		**x**	**x**	**11**

d) Second Chance

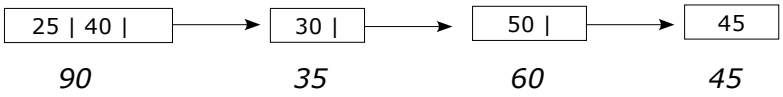
	1	**5**	**2**	**4**	**5**	**3**	**4**	**5**	**2**	**4**	**2**	**1**	**4**	**3**	**Summe**
S1	1	5	2	4	4	3	3	3	2	2	(2)	1	1	3	
S2		1	5	2	2	4	(4)	(4)	4	(4)	(4)	2	2	1	
S3			1	5	(5)	5	5	(5)	5	5	5	4	(4)	4	
	x	**x**	**x**	**x**		**x**			**x**			**x**		**x**	**8**

e) Optimale Verdrängung

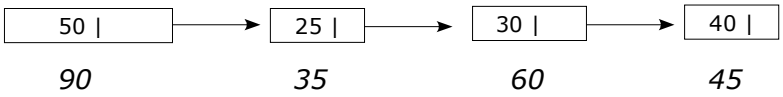
	1	**5**	**2**	**4**	**5**	**3**	**4**	**5**	**2**	**4**	**2**	**1**	**4**	**3**	**Summe**
S1	1	5	2	2	2	3	3	3	3	3	3	3	3	3	
S2	-	1	5	5	5	5	5	5	2	2	2	1	1	1	
S3	-	-	1	4	4	4	4	4	4	4	4	4	4	4	
	x	**x**	**x**	**x**		**x**			**x**			**x**			**7**

Aufgabe 5

a) First-Fit

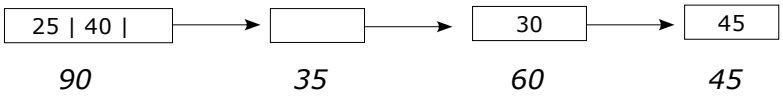


b) Best-Fit



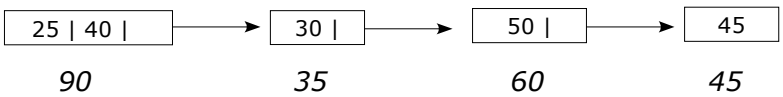
45 kann nicht mehr eingefügt werden

c) Worst-Fit



50 kann nicht mehr eingefügt werden

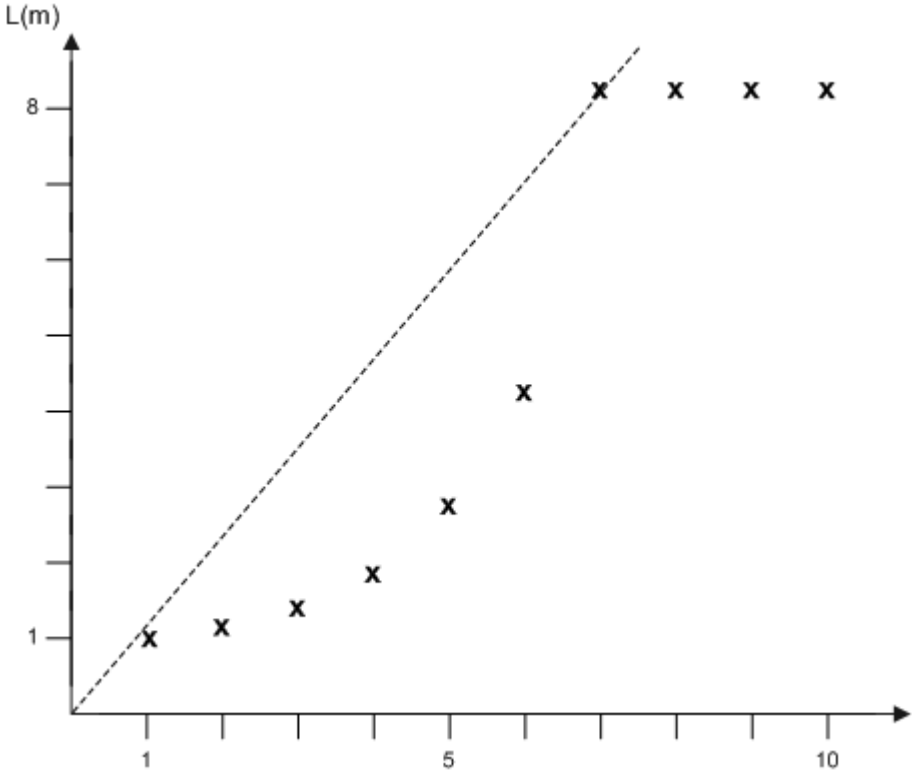
d) Next-Fit



Übung 11

Aufgabe 1

a)



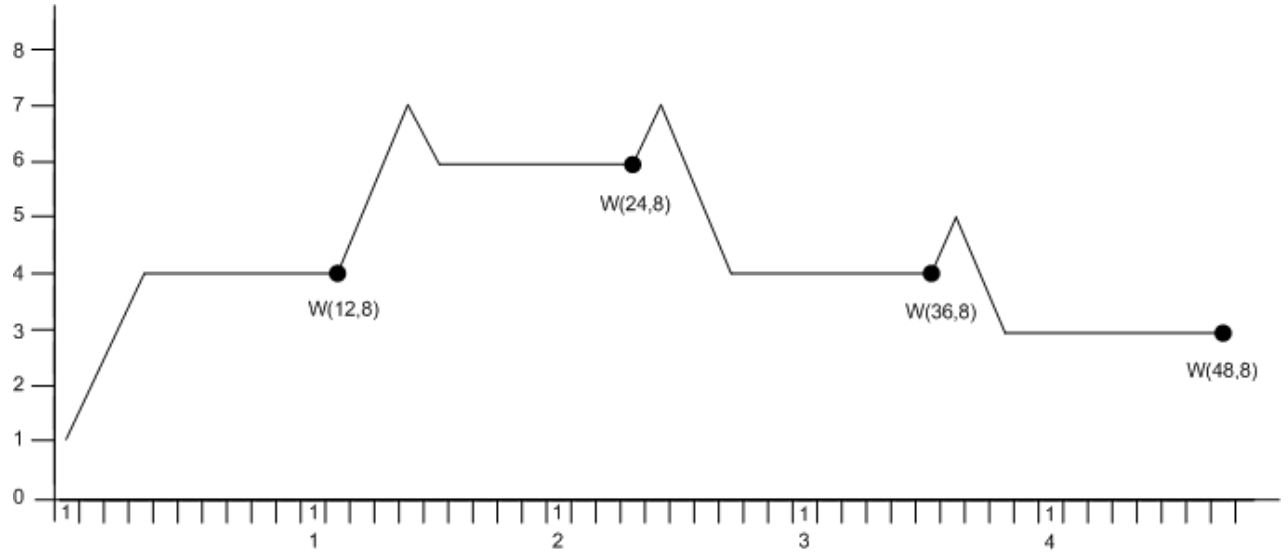
b) $primäres\ Knie: m_{opt} = 7$

c) nicht anwendbar

Aufgabe 2

a) Ist h zu klein gewählt, befinden sich nicht alle Seiten im Working Set
Ist h zu klein gewählt, sind zu viele inaktive Seiten im Working Set

b) + c)



$$W(12,8)=\{1,2,3,4\}$$
$$W(24,8)=\{2,3,4,5,6\}$$
$$W(36,8)=\{3,4,7,8\}$$
$$W(48,8)=\{1,7,8\}$$

d) Trashing: Zahl der Seitenfehler so groß, dass Systemdurchsatz indiskutabel wird.

Aufgabe 3

a)

$$RF(t_1,8)= \cup_{j=t_1 h+1}^{9} r_j=\{1,7,9,2,3\}$$

$$RF(t_2,8)= \cup_{j=t_2-h+1}^{16} r_j=\{7,3,1,8,0\}$$

$$VF(t_1,8)= \cup_{j=t_1+1}^{t_1+h} r_j=\{3,1,7,8,0,5\}$$

$$VF(t_2,8)= \cup_{j=t_2+1}^{t_2+h} r_j=\{5,6,8,0\}$$

b) Bei WS und VOPT treten jeweils 3 Seitenfehler auf

c)

VOPT

						2	2	2																
					9	9	9	9	2			1	1						6	6	6	6			
			1	7	7	7	7	7	7	7	7	7	7	1	8	8	0	5	5	5	5	5	6	0	
		3	3	3	3	3	3	3	3	3	3	3	3	7	1	1	8	8	8	8	8	8	5	6	6
t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
	-	3	1	7	9	2	3	9	2	7	3	1	3	7	8	1	0	5	6	8	6	8	5	0	6

WS

																	6							
																5	5	6						
						2	2	2				1	1			0	0	0	5	6	6				
					9	9	9	9	2	2	2	2	2	1	8	8	8	8	0	5	5	6			
				7	7	7	7	7	9	9	9	9	9	2	1	1	1	1	8	0	0	5	6		
			1	1	1	1	1	1	7	7	7	7	7	7	7	7	7	7	1	8	8	0	5	0	
		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	7	1	1	8	8	6	6
t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
	-	3	1	7	9	2	3	9	2	7	3	1	3	7	8	1	0	5	6	8	6	8	5	0	6

d) h = R / U

R: Kosten pro Seitenfehler R= 12
U: Kosten für das Halten einer Seite pro Zeiteinheit U= 4

h = 12 / 4 = 3

e) *hierfür existiert keine Musterlösung*