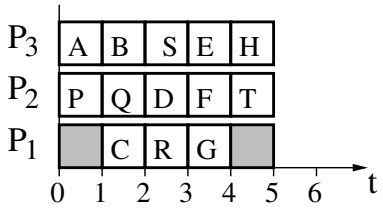
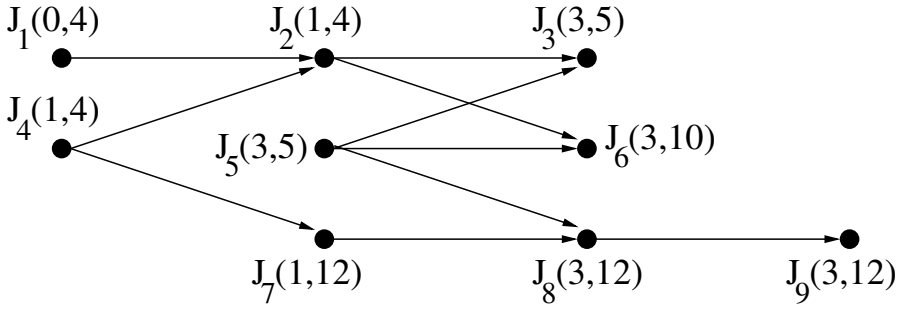
– Lösung zur Praktikumsaufgabe 5 –

Thema: *Scheduling*

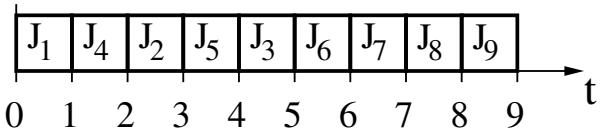
1. Der genutzte Algorithmus ist im Prinzip das Priorisierungsschema, das in Aufgabenstellung 2.c) verbal formuliert wurde. (Priorität bestimmt durch den längsten Pfad zu einem Job ohne Nachfolger; je länger dieser Pfad, desto größer die Priorität). Resultierender Schedule:



2. a) Effektive Bereitzeiten und Deadlines:



b) Gültiger Schedule nach EDF:

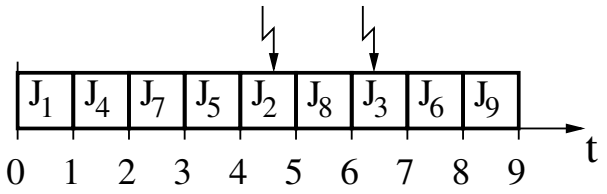


c) Den resultierenden Schedule zeigt die folgende Abbildung;

Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
Priorität	2	1	0	3	2	0	2	1	0

Tabelle 1: Prioritätsverteilung für die Jobs aus Aufgabe 2c)

J_2 und J_3 verpassen ihre Deadline.



3. a) Die folgende Tabelle gibt die Werte für die Slack Time der Tasks zu allen relevanten Systemzeitpunkten an (*non-strict* LST):

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
A	1	-	1	0	1	0	1	0	**-1**
B	1	-	1	0	1	-	1	0	1
C	1	-	1	-	1	0	1	-	1
D	2	1	1	1	1	1	0	0	2
E	2	1	1	1	0	0	0	0	2

Es resultiert der folgende Schedule (nach dynamischem LST):

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7
P_1	A	D	D	D	D	A	D	D
P_2	B	E	E	A	E	E	E	E
P_3	C	-	C	B	B	C	C	B

Wenn die Slack Times identisch sind, wurden nach der minimalen Anzahl vorzeitiger Beendigungen und Migrationen optimiert.

Zu $t = 8$ verpaßt ein Job seine Deadline (zu $t = 1$ wurde 1 Quantum Prozessorzeit verschenkt, welches ganz am Ende fehlt).

Ein gültiger Schedule ist z.B.:

	t=0	t=1	t=2	t=3	t=4
P_1	D	D	D	E	...
P_2	E	B	E	B	...
P_3	A	C	A	C	...

Das Verfahren LST ist optimal für mehrere Prozessoren, wenn die Deadlines und Bereitzeiten aller Jobs identisch sind.

4. Eine Beispielimplementierung für Linux, die keinen Anspruch auf Eleganz oder Effizienz legt, zeigt das Listing. Der ggT wird über die rekursive Variante des euklidischen Algorithmus implementiert; das kgV wird auf den ggT zurückgeführt.

Listing 1: Programm zur Ermittlung geeigneter Framegrößen (`framesize.c`)

```
/*
   framesize.c
   computes viable framesizes for a given set of tasks
   cf. Jane Liu: Real-Time Systems, Pearson, 2002, pp. 88ff. (section
       5.3.2)

   Limitations:
   o period t_p and framesize must be integer
   o Tasks are numbered starting from 1, one per line, Tx(t_phi, t_p,
       t_e, t_d)
   o Example: T1(0,4,1,4)
*/


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

#define MAXTASKS 100
#define MAXLINE   80

#ifndef DEBUG
#undef DEBUG
#endif

struct periodic_task {
  double t_phi;  /* phase of task */
  unsigned long t_p;    /* period */
  double t_e;    /* execution time */
  double t_d;    /* deadline */
};

/* Prototypes */

void read_taskset(void);
void dump_taskset(void);
unsigned long gcd_euclid(unsigned long, unsigned long);
unsigned long lcm(unsigned long, unsigned long);
unsigned long get_hyperperiod(void);
unsigned long minimum_framesize (void);
int test_framesize(unsigned long);
double get_load(void);

/* Globals */

struct periodic_task tasks[MAXTASKS];
unsigned int numTasks;
```

```
void read_taskset(void)
{
  char linebuf[MAXLINE], *aktpos;
  long index;

  while(!feof(stdin)) {
    if (!fgets(linebuf, MAXLINE-1, stdin)) {
      continue;
    }
    aktpos = strchr(linebuf, 'T');
    if (aktpos==NULL) {
      continue;        /* skip lines without 'T' */
    }
    aktpos = strtok(aktpos+1, "(");        /* find start of index */
    /* T1 soll am Index 0 stehen */
    index = strtol(aktpos, NULL, 10) - 1;
    aktpos = strtok(NULL, ",");        /* find 1st parameter */
    tasks[index].t_phi = strtod(aktpos, NULL);
    aktpos = strtok(NULL, ",");        /* find 2nd parameter ... */
    tasks[index].t_p = strtoul(aktpos, NULL, 10);
    aktpos = strtok(NULL, ",");
    tasks[index].t_e = strtod(aktpos, NULL);
    aktpos = strtok(NULL, ",");
    tasks[index].t_d = strtod(aktpos, NULL);
    numTasks++;
  }
  return;
}

void dump_taskset(void)
{
  int c = 0;

  for (c=0; c<numTasks; c++) {
    printf("T%d ( %.2f, %ld, %.2f, %.2f )\n", c+1, tasks[c].t_phi,
      tasks[c].t_p, tasks[c].t_e, tasks[c].t_d);
  }
}

unsigned long gcd_euclid(unsigned long a, unsigned long b)
{
  if (b==0) {
    return a;
  }
  else {
    return gcd_euclid(b, (a % b)) ;
  }
}

unsigned long lcm(unsigned long a, unsigned long b)
{
  return a / gcd_euclid(a, b) * b;
}

unsigned long get_hyperperiod(void)
```

```
{
  int c;
  unsigned long temp_h;

  temp_h = tasks[0].t_p;
  for (c=1; c<numTasks; c++) {
    temp_h = lcm(temp_h, tasks[c].t_p);
  }
  return temp_h;
}

double get_load(void)
{
  double load = 0.0;
  int c;

  for(c=0; c<numTasks; c++) {
    load += (double) tasks[c].t_e / (double) tasks[c].t_p;
  }
  return load;
}

/* computes the minumum frame size, i.e. INT(max(t_e,i)+1)  */

unsigned long minimum_framesize(void)
{
  int c;
  double maximum = 0.0;

  for (c=0; c<numTasks; c++) {
    if (tasks[c].t_e > maximum) {
      maximum = tasks[c].t_e;
    }
  }
  return (unsigned long) ceil(maximum);
}

int test_framesize(unsigned long f)
{
  int result = 1, c;

  for(c=0; (c<numTasks)&&(result==1); c++) {
    if ((double) (2*f - gcd_euclid(f, tasks[c].t_p)) > tasks[c].t_d) {
      result = 0;
    }
  }
  return result;
}

int main(int argc, char *argv[])
{
  unsigned long H;   /* hyper period */
  unsigned long act_f;

  read_taskset();
```

```
#ifdef DEBUG
  dump_taskset();
#endif

  H = get_hyperperiod();
  printf("H = %ld,  min f = %ld, load = %.2f\n", H, minimum_framesize
      (),
    get_load() );

  printf("Valid framesizes = { ");
  for (act_f = minimum_framesize(); act_f <= H/2; act_f++) {
    if ( (H % act_f) != 0 ) {   /* act_f | H ? */
#ifdef DEBUG
      printf("Framesize %ld does not divide H\n", act_f);
#endif
      continue;
    }
    /* 2f - gcd(f, t_p,i) <= t_d,i for all i ? */
    if (!test_framesize(act_f)) {
#ifdef DEBUG
      printf("Constraint 5.3 failed for f=%ld\n", act_f);
#endif
      continue;
    }
    printf("%ld ", act_f);
  }
  printf("}\n");

  return 0;
}
```

a) $f \in \{2, 3, 6\}$

b) $f = 8$

5. a) Das erste *Frame Size Constraint* erfordert:

$$f \geq 3$$

Die zweite Bedingung erfordert, dass f ein Teiler der Hyperperiode H ist.

$$H = \text{kgV}(4, 6) = 12 \rightsquigarrow f \in \{2, 3, 4, 6, 12\} \quad .$$

Die dritte Bedingung legt $\forall i$ fest:

$$2f - \gcd(t_{p,i}, f) \leq t_{d,i}$$

1. $f = 2$:

$$4 - \gcd(4,2) \leq 4$$
$$4 - \gcd(6,2) \leq 6 \quad \rightsquigarrow f = 2 \text{ ist gültig.}$$

2. $f = 3$:

$$6 - \gcd(4,3) \not\leq 4$$
$$6 - \gcd(6,3) \leq 6 \quad \rightsquigarrow f = 3 \text{ ist ungültig.}$$

3. $f = 4$:

$$8 - \gcd(4,4) \leq 4$$
$$8 - \gcd(6,4) \leq 6 \quad \rightsquigarrow f = 4 \text{ ist gültig.}$$

4. $f = 6$:

$$12 - \gcd(4,6) \not\leq 4$$
$$12 - \gcd(6,6) \leq 6 \quad \rightsquigarrow f = 6 \text{ ist ungültig.}$$

5. $f = 12$:

$$24 - \gcd(4,12) \not\leq 4$$
$$24 - \gcd(6,12) \not\leq 6 \quad \rightsquigarrow f = 12 \text{ ist ungültig.}$$

$\rightsquigarrow f = 4$. Für eine Hyperperiode sind also 3 Frames f_1, f_2, f_3 nötig. Innerhalb von H wird Task T_1 dreimal und Task T_2 zweimal aktiviert. $\Rightarrow$ Dekomposition in die Jobs $J_{i,j}(t_r, t_e, t_d)$:

$$J_{1,1}(0,3,4) \quad J_{1,2}(4,3,8) \quad J_{1,3}(8,3,12) \quad J_{2,1}(0,1.5,6) \quad J_{2,2}(6,1.5,12)$$

b) Das Netzwerk hat die in Abbildung 1 gezeigte Gestalt.

Der maximale Fluß durch diesen Graphen ist $\varphi = 11$, da beide Jobs von T_2 nicht im mittleren Frame f_2 ausgeführt werden dürfen. Da der maximale Fluß kleiner ist als die Summe der Ausführungszeiten der beteiligten Jobs ($\sum t_e = 12$), repräsentiert dieser Graph keinen brauchbaren Schedule. Es ist für $f = 4$ kein Schedule angebbar.

Abhilfe verspricht das *Job Slicing* (vgl. Vorlesungsskript). Wir dekomponieren die längste Task (T_1) in zwei Subtasks $T_{1A}(4,2)$ und $T_{1B}(4,1)$. Damit ermöglichen wir eine Framegröße von $f = 2$. Die Hyperperiode bleibt unverändert bei $H = 12$, also befinden sich nun 6 Frames in ihr. Es ergibt sich folgende Dekomposition von Jobs $J_{i,j}(t_r, t_e, t_d)$ in H:

$$J_{1A,1}(0,2,4) \quad J_{1A,2}(4,2,8) \quad J_{1A,3}(8,2,12)$$
$$J_{1B,1}(0,1,1) \quad J_{1B,2}(4,1,8) \quad J_{1B,3}(8,1,12)$$
$$J_{2,1}(0,1.5,6) \quad J_{2,2}(6,1.5,12)$$

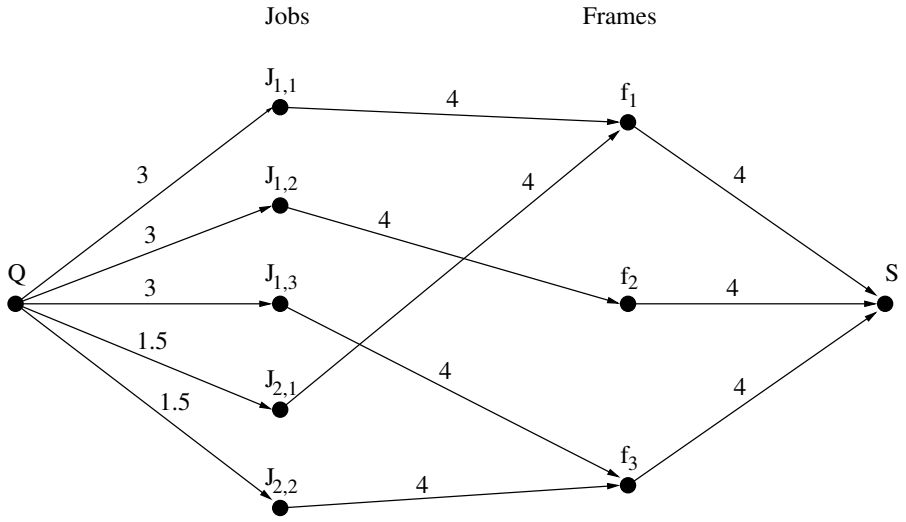


Abbildung 1: Netzwerk für $T_1(4,3)$, $T_2(6,1.5)$ sowie $f = 4$

Diese Jobs sind auf die folgenden 6 Frames zu verteilen:

$$f_1[0,2) \quad f_2[2,4) \quad f_3[4,6) \quad f_4[6,8) \quad f_5[8,10) \quad f_6[10,12)$$

Der Network Flow-Graph wird ein rechter Drahtverhau, wie Abbildung 2 verdeutlicht.

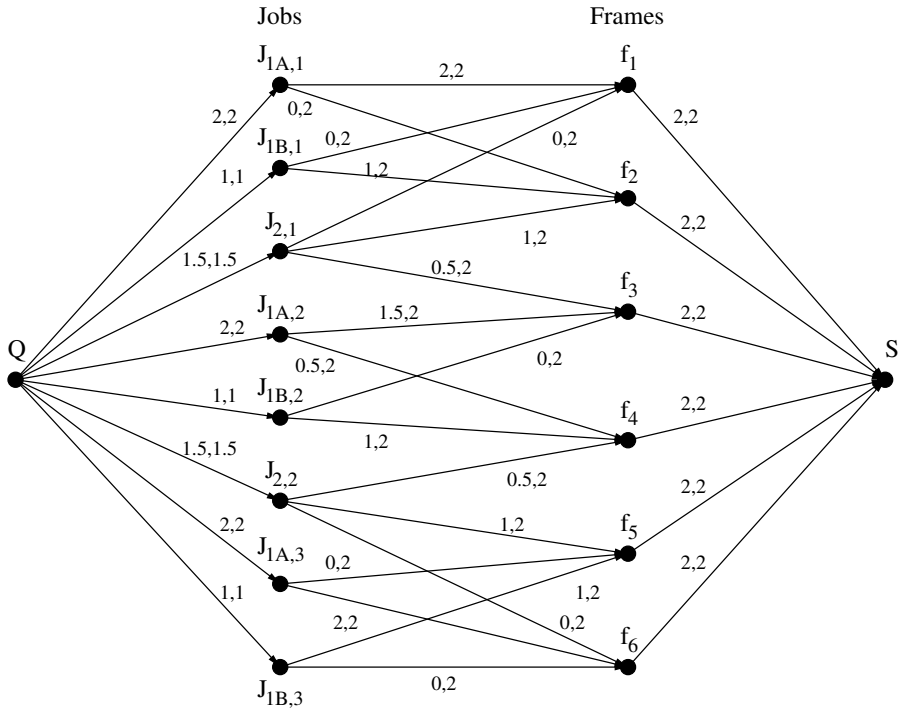


Abbildung 2: Network Flow Graph mit maximalem Fluß für $T_1(4,3)$, $T_2(6,1.5)$ sowie $f = 2$

Er zeigt einen (möglichen) maximalen Fluß, dessen Summe so groß ist, wie die akkumulierten Abarbeitungszeiten der Jobs; d.h., er repräsentiert einen brauchbaren Schedule (Abbildung 3).

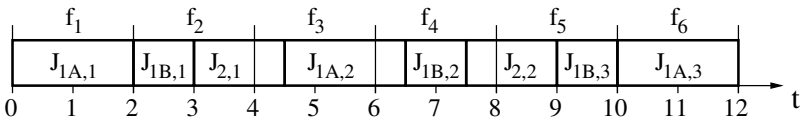


Abbildung 3: Abgeleiteter brauchbarer Schedule für $T_1(4, 3)$, $T_2(6, 1.5)$ sowie $f = 2$