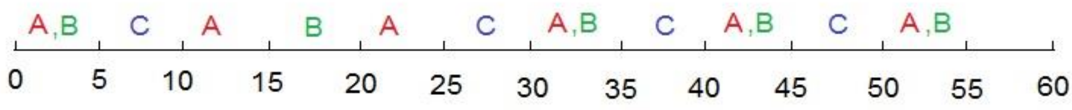
Answers to Review Problems

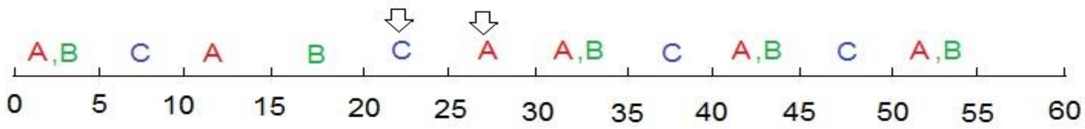
[1] a) Major cycle length will be 60 ms. Frame length f should satisfy $5 \le f \le 10$ and divide 60, which leaves the values 5, 6, and 10.

The condition $2f - gcd\,(f, T_i) \le d_i$ for all tasks will be satisfied by f=5 and 6 only. Taking f=5, we can construct schedule as shown below:

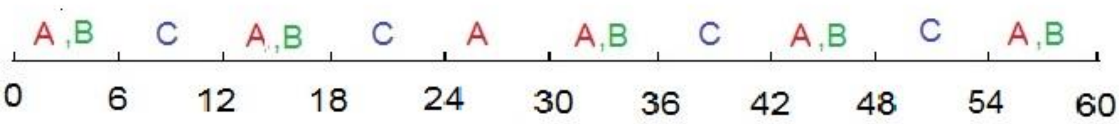


b) The condition
$2 \times 5 - \gcd(5{,}15) = 5 \le 10$

is still valid. However, the second execution of C in the above schedule will miss the new deadline. Thus schedule need to be modified as follows:



N.B. If we choose f=6, we can construct a schedule in part (a) as shown below



However, in part (b) we cannot modify the schedule as the complete frame for both A and C in the first period will be the first frame, and they cannot run together in the same frame.

---

[2] The major cycle length is 180 ms.

a) f=10 is larger than all execution times and less than all deadlines, and it divides 180. Further:

$2x10 - \gcd(10{,}30) = 10 \le 30$
$2x10 - \gcd(10{,}18) = 18 \le 18$
$2x10 - \gcd(10{,}20) = 10 \le 20$

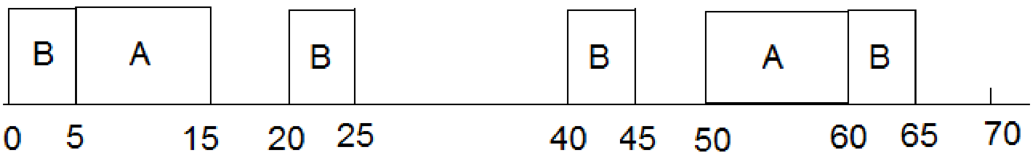
b) A possible execution schedule is shown below:

0-10	B,C	90-100	A,B
10-20	A	100-110	C
20-30	B,C	110-120	B
30-40	A	120-130	C
40-50	B,C	130-140	B,A
50-60		140-150	C
60-70	B,C	150-160	B,A
70-80	A	160-170	C
80-90	B,C	170-180	B

c) The time t=200 will be at the start of third frame in the second cycle from 180 to 360. System can insert the task in the first 4 ms in which the CPU is idle, which would be from 216 to 220, which the earliest time at which can be executed without perturbing the periodic tasks.

---

[3] a) Using Liu and Layland theorems, we first observe that $\sum u_i = 0.8071$, which is more than the threshold of the sufficient condition for three tasks. Using the critical instant theorem:



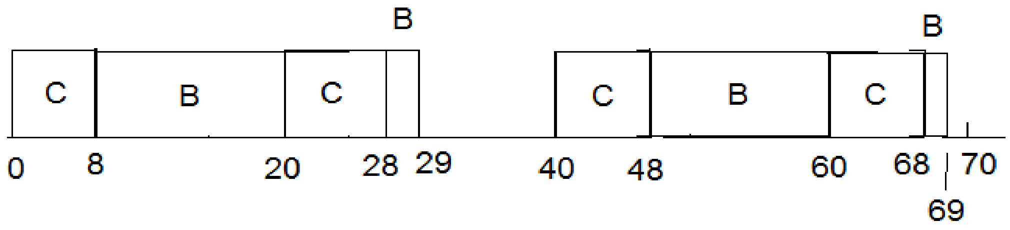
Thus, third task has 30 ms of idle CPU time before its deadline. Since it needs only 25 ms of execution, thus tasks are RM schedulable.

b) For the condition $\sum u_i \leq 1$ to remain satisfied, task of third period can be reduced to 45.46 ms.

c) If the first task remains of second priority and its execution time increases to 16 ms, third task would miss its deadline. Thus first task, and since it is a soft real-time task, would be given the least priority. If its execution time increases to 16, it will miss its deadline (this occurs with low probability) while the other two hard real-time tasks will not miss their deadlines.

---

[4] a) Again using the critical instance theorem, the tasks are RM schedulable.



b) Since $\sum u_i = 0.8964 \leq 1$, the tasks are EDF schedulable,

c) The CPU utilization of the three tasks are 0.171, 0.325. and 0.4. Arranging the utilizations of all tasks in non-decreasing order:

0.71, 0.55, 0.47, 0.4, 0.325. 0.21, 0.171, 0.16

Then applying the first-fit algorithm:

Processor 1 0.71, 0.21
Processor 2 0.55, 0.4,
Processor 3 0.47, 0.325, 0.171
Processor 4 0.16

Thus, the minimum number of required processors is 4.

---

[5] a) Priority order will be A-B-C

$R_A = 4 \leq 15$

$R_B = 5 + \left\lceil \frac{R_B}{15} \right\rceil \times 4$ which converges at $R_B = 9 \leq 20$

$R_C = 10 + \left\lceil \frac{R_C}{15} \right\rceil \times 4 + \left\lceil \frac{R_C}{20} \right\rceil \times 5$ which converges at $R_C = 28 \leq 30$

Thus tasks will always meet their deadlines.

b)
$R_A = 4 + 4 = 8 \leq 15$

$R_B = 5 + 4 + \left\lceil \frac{R_B}{15} \right\rceil \times 4$ which converges at $R_B = 13 \leq 20$

$R_C = 10 + \left\lceil \frac{R_C}{15} \right\rceil \times 4 + \left\lceil \frac{R_C}{20} \right\rceil \times 5$ which converges at $R_C = 28 \leq 30$

Thus tasks will still meet their deadlines.

c) Worst-case response times will be the same, and thus third task will miss its deadline in the worst case.

d) Priority order will be A-C-B

$R_A = 4 \ \leq 15$

$R_C = 10 + \left\lceil \frac{R_C}{15} \right\rceil \times 4$ which converges at $R_C = 14 \ \leq 20$

$R_B = 5 + \left\lceil \frac{R_B}{15} \right\rceil \times 4 + \left\lceil \frac{R_B}{30} \right\rceil \times 10$ which converges at $R_B = 23 \ \leq 24$

Thus, using dead-line monotonic scheduling deadlines will not be missed.

---

[6]
a) 3
b) 3
c) zero
d)
Process 1 operates m=2, print A, n = 1
Process 2 operates m=2, print BC, n = 1
Process 2 operates m=2, print B, n = 0 and pre-empted
Process 1 operates m=1, print A, n = 1
Process 3 operates m=1, print D, n = 0
Process 2 operates again m=1, print C, n = 1
Process 1 operates m=0, print A, n = 2
Process 2 operates m=0, print BC, n = 2
Process 3 operates m=0, print D, n = 1
Process 3 operates m=0, print A, n = 0

Thus, sequence is possible.

---

[7] We use a semaphore m initially equal to zero

```
Thread 1      Thread 2     ….    Thread n       Required Thread
….            ……                 …..            down (m);
up (m);       up (m);            up (m);        down (m);
                                                …..
                                                down (m);    // n times
                                                // continue
```

Note that there is no need to use more than one semaphore.

---

[8] a) Using two semaphores m and n initially equal to zero

```
Process P1                 Process P2                 Process P3
…..                        …..                        …..
for (i=0; i < 50; i++)     for (j=0; j < 50; j++)     for (k=0; k < 50; k++)
{A[i]= fn1(i);             {B[j]= fn2(j);             {down (m); down(n);
  up(m);}                   up(n);}                    C[k]= A[k]+B[k];}
  …..                      …..                        …..
```

b) No, if we use m only

```
Process P1                 Process P2                 Process P3
…..                        …..                        …..
for (i=0; i < 50; i++)     for (j=0; j < 50; j++)     for (k=0; k < 50; k++)
{A[i]= fn1(i);             {B[j]= fn2(j);             {down (m); down(m);
  up(m);}                   up(m);}                    C[k]= A[k]+B[k];}
  …..                      …..                        …..
```

P3 may operate after two iterations of P1 while P2 did not operate.

c) Using two semaphores m and n initially equal to zero

```
Process P1                 Process P2                 Process P3
…..                        …..                        …..
for (i=0; i < 50; i++)     for (j=0; j < 50; j++)     down (m); down(n);
A[i]= fn1(i);              B[j]= fn2(j);               for (k=0; k < 50; k++)
up(m);                     up(m);                        C[k]= A[k]+B[k];
  …..                      …..                        …..
```

i.e up( ) and down ( ) operations are out of the loops.
