1. MILP Linearization

(1) $\alpha + \beta + \gamma\ 6{=}\ 2 \iff \alpha + \beta - \gamma \leq 1 \land \alpha - \beta + \gamma \leq 1 \land -\alpha + \beta + \gamma \leq 1$

α	β	γ	LHS	α + β − γ ≤ 1	α - β + γ ≤ 1	-α + β + γ ≤ 1	RHS	LHS=RHS?
0	0	0	T	T	T	T	T	YES
0	0	1	T	T	T	T	T	YES
0	1	0	T	T	T	T	T	YES
0	1	1	F	T	T	F	F	YES
1	0	0	T	T	T	T	T	YES
1	0	1	F	T	F	T	F	YES
1	1	0	F	F	T	T	F	YES
1	1	1	T	T	T	T	T	YES

因為在所有情況下 LHS=RHS，所以他們是 equivalent。

(2) $\alpha\beta = \gamma \iff \alpha + \beta - 1 \leq \gamma \land \gamma \leq \alpha \land \gamma \leq \beta$

α	β	γ	LHS	α + β − 1 ≤ γ	γ ≤ α	γ ≤ β	RHS	LHS=RHS?
0	0	0	T	T	T	T	T	YES
0	0	1	F	T	F	F	F	YES
0	1	0	T	T	T	T	T	YES
0	1	1	F	T	F	T	F	YES
1	0	0	T	T	T	T	T	YES
1	0	1	F	T	T	F	F	YES
1	1	0	F	F	T	T	F	YES
1	1	1	T	T	T	T	T	YES

因為在所有情況下 LHS=RHS，所以他們是 equivalent。

(3) $\beta x = y \iff 0 \leq y \leq x \land x - M(1 - \beta) \leq y \land y \leq M\beta$

為了確保 M 足夠大可以滿足上述公式，參考題目附的 table 可以發現，當 β 為 0 時，M 必須滿足的條件為: M >= x，由於變數 x <= 2020，所以 M 必須至少大於等於 2020；而當 β 為 1 時，M 必須滿足的條件為: M >= x = y，所以 M 仍然必須大於等於 2020，所以結論是我們取 2020 為 M 的值。

2. Signal Packing

(1)

YES, the new design is better

因為將兩個併在一起可以 save header，充分的運用剩餘空間，讓傳輸更有效率。

(2)

NO

因為 μ2 和 μ’0 的 sender 並非在同一個 ECU 上。

(3)

YES

因為 μ’0 和 μ3 為同一個 sender，將他們 merge 之後，可以一次傳輸 32bits，讓傳輸效率更加提升。

3. Simulated Annealing for Priority Assignment

(1) Priorities of messages

```
13
4
0
6
2
3
7
8
1
9
12
11
10
5
15
14
16
```

(2) objective value

```
205.72
```

(3)Code segment

將 input 讀進來且寫入變數的部分:

```
import time
import numpy as np
import random
import math

messages = []
with open('input.dat', 'r', encoding='UTF-8') as file:
    numbers = float(file.readline())
    tau = float(file.readline())
    for data in file.readlines():
        data = data.strip()
        data = data.split()
        messages.append(data)
# print(numbers, tau, messages)
```

計算 worst response time 和所有 messages 加總的部分

```python
def worstwaitingtime(number):
    num = number
    blocking_list = []
    for element in messages:
        if int(element[0]) >= int(messages[num][0]):
            blocking_list.append(element[1])
    block = float(max(blocking_list))
    # print(block)

    LHS = block
    Q = block
    RHS = 0
    while True:
        RHS = 0

        for i in range(0,int(messages[num][0])):
            a = 0
            for element in messages:
                if int(element[0]) == i:
                    break
                a += 1
            RHS += np.ceil((Q+tau)/float(messages[a][2]))*float(messages[a][1])
            # print(RHS)
        RHS += block
        # print(RHS)
        if LHS == RHS:
            break
        LHS = RHS
        Q = RHS
    return RHS+float(messages[num][1])

def total():
    obj = 0
    for i in range(17):
        a = worstwaitingtime(i)
        if a > float(messages[i][2]):
            obj += 1000  # penality of breaking restrictions
        obj += a
    return(obj)
```

Simulated annealing 的部分:

```python
if __name__ == "__main__":
    start = time.time()
    T = 100 # initial temp
    S = total() # initial solution
    output = []
    for i in range(17):
        output.append(messages[i][0])
    threshold = 0.01 # 機率的臨界值
    a = 0
    while(T>0.01):
        s1 = random.randint(0,16) # pick a random nieghbor
        s2 = random.randint(0,16)
        messages[s1][0],messages[s2][0] = messages[s2][0],messages[s1][0] # 交換priority
        cost = total()
        # if cost <= S:
        if cost <= S or math.exp((S-cost)/T) > threshold:
            S = cost
            output = []
            for i in range(17):
                output.append(messages[i][0]) # 保留最佳解時的priority分配
        T *= 0.999 # r=0.999
```

將結果寫出的部分:

```python
with open('output1.txt', 'w', newline='') as f:
    for i in range(17):
        f.writelines(str(output[i])+'\n')
with open('output2.txt', 'w', newline='') as f:
    f.write(str(S))
end = time.time()

print("runtime:", end-start, "secs")
```