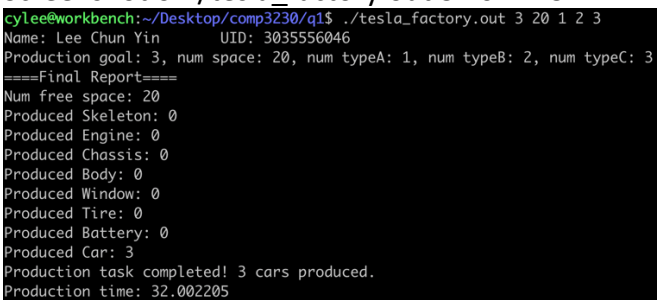
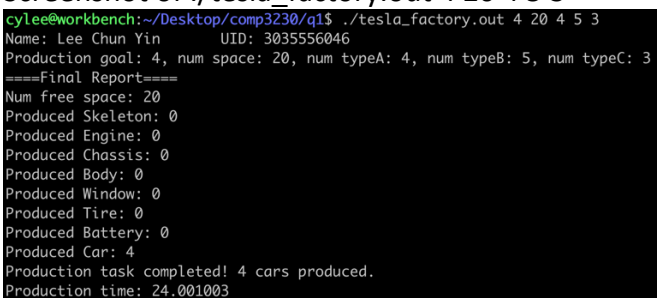
Q1:

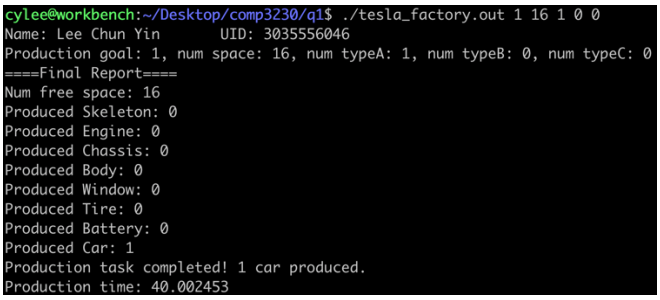
Screenshot of ./tesla_factory.out 3 20 1 2 3



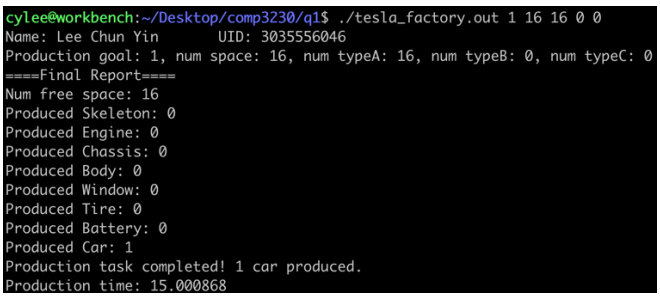
Screenshot of ./tesla_factory.out 4 20 4 5 3

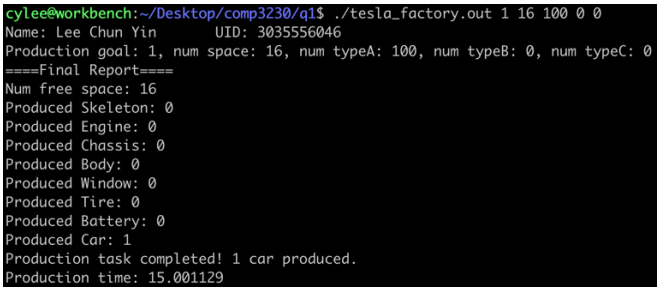


We need num_spaces>=16 (17 – 1 since we don't need to store the car) in order to ensure sufficient storage spaces, thus 16 storage spaces are sufficient for sure because 1 skeleton, 1 engine, and 1 chassis can make 1 car body and 7 windows, 1 body, 4 tires, and 1 battery pack can make 1 car. Also, note that the production time is 40 which is the same as the expected production time of robot A.



It is not always true that the number of robots increases, the production time decreases.





In general, adding more robots may decrease the production time while increasing number of cars may increase the production time. However, if we add excess number of robots, the production may increase slightly because there are more overheads for large number of robots. For different type of robots, we don't know whether the time for making each component is the least.

Q2:

Problem: The production process is executed in a way that may cause deadlock.
Solution: (Deadlock prevention strategy)
In order to solve the deadlock problem, we can change the order of jobs put in jobQ. The solving algorithm is to first put body and its raw materials into jobQ and then put car and its raw materials into jobQ. The algorithm ensures enough spaces for storage because after making a raw material (component) of body/car, their space will be released. Therefore, we can free up one space for another raw material (component) to enter the jobQ and ensure that the algorithm is deadlock free. Also, we add a semaphore to ensure only one thread call the dequeue function.

Q3:
To increase the speed of execution, I use the autonomous thread model. First, create twelve semaphores in task struct and initialize it in the main function. The semaphore sem is a binary semaphore which controls the execution flow of the program in order to prevent getting the head of the queue from multiple threads. The six semaphores sem0, sem1, sem2, sem3, sem4 and sem5 are the number of components that are currently required. The sem6 semaphore controls the resources and have the semaphore value of number of spaces minus one. The minus one indicates that we need to allocate one space for body. The semaphores semA, semB and semC indicates the number of robots that are not working on jobs and the initial value of each semaphore is the number of each type of robots. The last semaphore sem_body is to avoid deadlock in the production process. Then, according to the type of robots and raw materials, the program will try to match the job to the robots with least production time (with some constraints). For type A robots, the production time of chassis, window and battery is the least compared to other robots. Therefore, if the head of the queue is those components of the car, the program will allow the thread to start making them. Similarly, for type B robots, the production time of body is the least compared to other robots and for type C robots, the production time of skeleton, engine, tire and car is the least compared to other robots. In general, if the production time of the component-robot pair is not the least, the matching will continue. However, in order to avoid repeating the matching procedure, when the semaphore values of the other robot(s) are zero, the program will also allow the thread to start making them. The idea behind is simple, take

production time of battery as an example. We know that type A takes least amount of time to make battery. However, if the semaphore value of type A robot is zero, is it worth to wait for at most three more seconds for the semaphore value to turn positive and wait for the released space which the time may not be deterministic? If type B robot is free at this moment, it takes only four seconds to finish. During production, the value of the semaphore of the type of robot will be decreased by one. The value of the semaphore sem will be increased by one after dequeue. The above algorithm applies for every types of robots.

	Type A	Type B	Type C	MIN	MAX
SKELETON	5	4	3	**3**	**5**
ENGINE	4	5	3	**3**	**5**
CHASSIS	3	4	4	**3**	**4**
WINDOW	1	2	3	**1**	**3**
TIRE	2	2	1	**1**	**2**
BATTERY	3	4	4	**3**	**4**
BODY	4	3	6	**3**	**6**
CAR	6	5	4	**4**	**6**
Total	40	47	49	**30**	**59**

Table 1: Production Time

```c
typedef struct Task_t {
  // Feel free to add more stuff if needed
  Queue jobQ;
  sem_t sem;
  sem_t sem0;
  sem_t sem1;
  sem_t sem2;
  sem_t sem3;
  sem_t sem4;
  sem_t sem5;
  sem_t sem6;
  sem_t semA;
  sem_t semB;
  sem_t semC;
  sem_t sem_body;
} Task_t;
```

```c
  /* Prepare task */
  Task task = calloc(1, sizeof(Task_t));
  task->jobQ = queueCreate(num_cars * 17);
  sem_init(&task->sem, 0, 1);
  sem_init(&task->sem0, 0, 0);
  sem_init(&task->sem1, 0, 0);
  sem_init(&task->sem2, 0, 0);
  sem_init(&task->sem3, 0, 0);
  sem_init(&task->sem4, 0, 0);
  sem_init(&task->sem5, 0, 0);
  sem_init(&task->sem6, 0, num_spaces - 1);
  sem_init(&task->semA, 0, num_typeA);
  sem_init(&task->semB, 0, num_typeB);
  sem_init(&task->semC, 0, num_typeC);
  sem_init(&task->sem_body, 0, num_spaces - 1);
  for (int k = 0; k < num_cars; k++){

    //Store raw materials of body
    int i = 6;
    queueEnqueue(task->jobQ, &i);
    for(int j = 0; j < 3; ++j){
      queueEnqueue(task->jobQ, &j);
    }

    //Store raw materials of car
    i = 7;
    queueEnqueue(task->jobQ, &i);
    for(int j = 3; j < 6; ++j){
      if(j == WINDOW) {
        for(int l = 0; l < 7; l++) queueEnqueue(task->jobQ, &j);
      } else if(j == TIRE) {
        for(int l = 0; l < 4; l++) queueEnqueue(task->jobQ, &j);
      } else queueEnqueue(task->jobQ, &j);
    }
  }
  /* Prepare task end*/
```

```c
while (!queueIsEmpty(task->jobQ)) {
  sem_wait(&task->sem);
  if(queueIsEmpty(task->jobQ)){
    sem_post(&task->sem);
    continue;
  }
  sem_getvalue(&task->semA, &numA);
  sem_getvalue(&task->semB, &numB);
  sem_getvalue(&task->semC, &numC);

  head = *queueFront(task->jobQ);
  if(type == 'A'){
    if(head == 2 || head == 3 || head == 4 || numB == 0 || numC == 0){
      sem_wait(&task->semA);
      queueDequeueFront(task->jobQ, &jobID);
      sem_post(&task->sem);
#ifdef DEBUG
    debug_printf(__func__, "Robot%c[%d]: working on job %d...\n",
        RobotTypeToChar(robot->robotType), robot->id, jobID);
#endif
      revisedSimpleWork(jobID, robot);
      sem_post(&task->semA);
    }else{
      sem_post(&task->sem);
    }
  }else if(type == 'B'){
    if(head == 6 || numA == 0 || numC == 0){
      sem_wait(&task->semB);
      queueDequeueFront(task->jobQ, &jobID);
      sem_post(&task->sem);
#ifdef DEBUG
    debug_printf(__func__, "Robot%c[%d]: working on job %d...\n",
        RobotTypeToChar(robot->robotType), robot->id, jobID);
#endif
      revisedSimpleWork(jobID, robot);
      sem_post(&task->semB);
    }else{
      sem_post(&task->sem);
    }
  }else{
    if(head == 0 || head == 1 || head == 5 || head == 7 || numA == 0 || numB == 0){
      sem_wait(&task->semC);
      queueDequeueFront(task->jobQ, &jobID);
      sem_post(&task->sem);
#ifdef DEBUG
    debug_printf(__func__, "Robot%c[%d]: working on job %d...\n",
        RobotTypeToChar(robot->robotType), robot->id, jobID);
#endif
      revisedSimpleWork(jobID, robot);
      sem_post(&task->semC);
    }else{
      sem_post(&task->sem);
    }
  }
}
pthread_exit(NULL);
```

Scalability analysis:
There are three independent variables that is production goal, number of spaces and number of robots. First, fix the spaces and robots. The production time increases linearly. Depending on other two variables, the increment in production time varies. For fewer number of spaces and number of robots, the production time increment will be larger. Note that the production time will not converge.

```
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory_debug.out 2 16 9 1 7
Name: Lee Chun Yin       UID: 3035556046
Production goal: 2, num space: 16, num typeA: 9, num typeB: 1, num typeC: 7
====Final Report====
Num free space: 16
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 16.002277
```

```
====Final Report====
Num free space: 16
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 4
Production task completed! 4 cars produced.
Production time: 23.008333
```

```
====Final Report====
Num free space: 16
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 6
Production task completed! 6 cars produced.
Production time: 30.004437
```

```
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory_debug.out 2 7 1 2 3
Name: Lee Chun Yin       UID: 3035556046
Production goal: 2, num space: 7, num typeA: 1, num typeB: 2, num typeC: 3
====Final Report====
Num free space: 7
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 27.003300
```

```
====Final Report====
Num free space: 7
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 3
Production task completed! 3 cars produced.
Production time: 36.031176
```

```
====Final Report====
Num free space: 7
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 4
Production task completed! 4 cars produced.
Production time: 45.022745
```

Then, fix the number of production goals and robots. For storage space less than 17, it is considered insufficient and thus the production time will be higher. If the storage space is much higher than sufficient, the production time will decrease but converge. In this case, it is 15 seconds.

```
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory_debug.out 2 2 4 4 4
Name: Lee Chun Yin       UID: 3035556046
Production goal: 2, num space: 2, num typeA: 4, num typeB: 4, num typeC: 4
====Final Report====
Num free space: 2
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 58.275157
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory.out 2 17 4 4 4
Name: Lee Chun Yin       UID: 3035556046
Production goal: 2, num space: 17, num typeA: 4, num typeB: 4, num typeC: 4
====Final Report====
Num free space: 17
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 18.043139
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory_debug.out 2 30 4 4 4
Name: Lee Chun Yin       UID: 3035556046
Production goal: 2, num space: 30, num typeA: 4, num typeB: 4, num typeC: 4
====Final Report====
Num free space: 30
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 17.002147
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory_debug.out 2 400 4 4 4
Name: Lee Chun Yin       UID: 3035556046
Production goal: 2, num space: 400, num typeA: 4, num typeB: 4, num typeC: 4
====Final Report====
Num free space: 400
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 15.022004
```

Lastly, we fix the production goal and the number of spaces. For low number of robots, the production time will be much higher since the program have to wait for the component to finish. However, the production time have an upper bound and converge if the storage space is insufficient, that is the robots need to wait for the storage space.

```
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory_debug.out 2 4 1 1 0
Name: Lee Chun Yin      UID: 3035556046
Production goal: 2, num space: 4, num typeA: 1, num typeB: 1, num typeC: 0
====Final Report====
Num free space: 4
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 94.135805
cylee@workbench:~/Desktop/comp3230/q3$ ./tesla_factory_debug.out 2 4 15 15 15
Name: Lee Chun Yin      UID: 3035556046
Production goal: 2, num space: 4, num typeA: 15, num typeB: 15, num typeC: 15
====Final Report====
Num free space: 4
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 27.043209
```

When the robots need to wait for the space, then it will affect the performance. The performance difference can be caused by the server, the overhead and the threads are not deterministic. The reason behind the deadlock problem is that the threads are not deterministic. Some threads run much faster than expected and it is not a decent case in implementation causing deadlock. For example, if the storage space is two, skeleton and engine entering the storage will cause deadlock. In order to avoid deadlock, we need to use multiple semaphores which degrade the performance.