This is a multi-threaded program that simulates a car assembly line. The program reads the number of each type of car (Type A, Type B, Type C, and Type D) and the maximum number of simulation days from a file called "input.txt". It also reads the limits for each car part, such as the number of chassis, paint, tires, seats, engines, and top covers that are available.

```
37 ▼ int main() {
38
39     FILE *input_file = fopen("input.txt", "r");
40 ▼   if (input_file == NULL) {
41       perror("ERROR");
42       return 1;
43     }
44     fscanf(input_file, "%d %d %d %d %d", &numOfTypeA, &numOfTypeB, &numOfTypeC,
45           &numOfTypeD, &maxSimulationDay);
46     fscanf(input_file, "%d %d %d %d %d %d", &chassisLimit, &paintLimit,
47           &tireLimit, &seatLimit, &engineLimit, &topCoverLimit);
48     fclose(input_file);
49
50     sem_init(&sem_chassis, 0, chassisLimit);
51     sem_init(&sem_tires, 0, tireLimit);
52     sem_init(&sem_seats, 0, seatLimit);
53     sem_init(&sem_engine, 0, engineLimit);
54     sem_init(&sem_top_cover, 0, topCoverLimit);
55     sem_init(&sem_paint, 0, paintLimit);
56
```

The program then uses semaphores to control access to the shared resources, such as the car parts. For example, the sem_chassis semaphore controls access to the chassis and the sem_tires semaphore controls access to the tires. The semaphores are initialized with the limits read from the input file.

The program also creates a struct called "Car" which contains information about each car being assembled, such as the car's ID, whether its chassis, tires, seats, engine, and top cover are on, whether the car is done, and a lock to control access to the car's data.

```
▼ struct Car {
    pthread_mutex_t lock;
    int id;
    int chassisOn;
    int tiresOn;
    int seatsOn;
    int engineOn;
    int topCoverOn;
    int paintOn;
    int isDone;
  };
  struct Car cars[BUFFER];

  int numOfTypeA, numOfTypeB, numOfTypeC, numOfTypeD, maxSimulationDay,
      currentSimulationDay;
  int chassisLimit, paintLimit, tireLimit, seatLimit, engineLimit, topCoverLimit;

  sem_t sem_chassis, sem_top_cover, sem_paint, sem_tires, sem_seats, sem_engine;
```

The program creates four types of threads, one for each type of car, and each thread represents a technician working on the assembly line. The threads are created using the pthread_create() function and are passed a function to call when they are running. For example, the typeA threads are passed the typeACallFunc function.

The typeACallFunc, typeBCallFunc, typeCCallFunc, typeDCallFunc are callback functions which will be executed when the thread starts to run. Each of these functions simulate the operations performed by a technician on the car. They check which car parts are needed for their specific type of car and use sem_wait() and sem_post() functions to acquire and release the necessary resources.

```
138 ▼ void *typeACallFunc(void *arg) {
139
140     int id = *((int *)arg);
141     int carID;
142
143 ▼   while (1) {
144
145 ▼     for (carID = 0; carID < BUFFER; carID++) {
146 ▼       if (pthread_mutex_trylock(&cars[carID].lock) == 0) {
147 ▼         if (cars[carID].chassisOn && !cars[carID].tiresOn) {
148
149           sem_wait(&sem_tires);
150           usleep(300);
151
152           cars[carID].tiresOn = 1;
153           printf("Type A - %d       %d      tires       %d\n",
154                  id, carID + 1, currentSimulationDay);
155         }
156       }
157
158 ▼     if (cars[carID].topCoverOn && !cars[carID].paintOn) {
159
160         sem_wait(&sem_paint);
161         usleep(300);
162         cars[carID].paintOn = 1;
163         cars[carID].isDone = 1;
164
165         printf("Type A - %d     %d      paint       %d\n",
166                id, carID + 1, currentSimulationDay);
167       }
168       pthread_mutex_unlock(&cars[carID].lock);
169     }
170 ▼   if (carID == BUFFER) {
171       continue;
172     }
173   }
174 }
```

```
177 ▼ void *typeBCallFunc(void *arg) {
178
179     int id = *((int *)arg);
180     int carID;
181
182 ▼   while (1) {
183
184 ▼     for (carID = 0; carID < BUFFER; carID++) {
185 ▼       if (pthread_mutex_trylock(&cars[carID].lock) == 0) {
186           if (!cars[carID].chassisOn &&
187 ▼             currentSimulationDay != maxSimulationDay) {
188
189             sem_wait(&sem_chassis);
190             usleep(300);
191             cars[carID].chassisOn = 1;
192             printf("Type B - %d       %d      chassis     %d\n",
193                    id, carID + 1, currentSimulationDay);
194           }
195           pthread_mutex_unlock(&cars[carID].lock);
196         }
197       }
198 ▼     if (carID == BUFFER) {
199         continue;
200       }
201     }
202   }
```

The main function then enters a loop that runs for the maximum number of simulation days. On each iteration of the loop, the current simulation day is incremented and the threads are allowed to run. The program outputs the technician ID, car ID, operation, and simulation day for each operation performed on a car.

In summary, this program simulates a car assembly line using multi-threading and semaphores to coordinate access to shared resources between the different threads. The program reads input from a file, initializes semaphores, creates threads, and simulates the assembly of cars over a specified number of days.

SAMPLE RUN:

```
DAY 1
Type B - 0       1       chassis     1
Type B - 0       2       chassis     1
Type B - 0       3       chassis     1
Type B - 0       4       chassis     1
Type C - 3       2       seats       1
Type C - 3       4       seats       1
Type D - 1       1       engine      1
Type C - 2       3       seats       1
Type D - 1       2       engine      1
Type D - 1       3       engine      1
Type D - 1       4       engine      1
Type A - 1       1       tires       1
Type A - 1       2       tires       1
Type A - 1       3       tires       1
Type A - 1       4       tires       1
Type C - 3       1       seats       1
Type D - 0       1       top cover   1
Type D - 0       2       top cover   1
Type D - 0       3       top cover   1
Type D - 0       4       top cover   1
DAY 2
Type B - 0       5       chassis     2
Type B - 2       6       chassis     2
Type A - 0       1       paint       2
Type A - 0       2       paint       2
Type C - 0       5       seats       2
Type A - 0       3       paint       2
Type A - 0       4       paint       2
Type A - 0       5       tires       2
Type C - 0       6       seats       2
Type D - 0       5       engine      2
Type D - 0       5       top cover   2
Type D - 0       6       engine      2
Type A - 1       5       paint       2
Type A - 1       6       tires       2
Type D - 1       6       top cover   2
DAY 3
Type B - 1       5       chassis     3
Type B - 0       6       chassis     3
Type A - 0       6       paint       3
DAY 4
Type B - 2       7       chassis     4
Type B - 1       7       chassis     4
```