Main Function

We have started by defining necessary global variables, such as some mutexes, a car struct and an array that stores those cars.

Our main function starts by parsing a given input file in order to get the necessary variables.

```
// open input.txt
FILE *input = fopen("input.txt", "r");
if (input == NULL)
{
    printf("Error opening input.txt\n");
    return 1;
}

// read input.txt line by line
fscanf(input, "%d %d %d %d %d\n%d %d %d %d %d %d", &NUM_THREAD_A, &NUM_THREAD_B, &NUM_THREAD_C, &NUM_THREAD_D, &NUM_DAYS,
       &NUM_DAILY_CHASIS, &NUM_DAILY_TIRES, &NUM_DAILY_SEATS, &NUM_DAILY_ENGINES, &NUM_DAILY_TOPCOVERS, &NUM_DAILY_PAINTS);
// close input.txt
fclose(input);
```

We have tried to redirect the stdout to a file but it did not work properly so we have used the command line to see our output.

Then we determined the minimum size for the cars array.

```
// get the smallest amount of material
int min = NUM_DAILY_CHASIS;
if (NUM_DAILY_TIRES < min)
{
    min = NUM_DAILY_TIRES;
}
if (NUM_DAILY_SEATS < min)
{
    min = NUM_DAILY_SEATS;
}
if (NUM_DAILY_ENGINES < min)
{
    min = NUM_DAILY_ENGINES;
}
if (NUM_DAILY_TOPCOVERS < min)
{
    min = NUM_DAILY_TOPCOVERS;
}
if (NUM_DAILY_PAINTS < min)
{
    min = NUM_DAILY_PAINTS;
}

int arraySize = sizeof(car) * min;
cars = malloc(arraySize);
```

We have created thread arrays for each worker type and initialize them in a for loop.

```
for (int i = 0; i < NUM_THREAD_A; i++)
{
    pthread_create(&WORKERS_A[i], NULL, WORKER_A_WORK, &cars[0]);
}
for (int i = 0; i < NUM_THREAD_B; i++)
{
    pthread_create(&WORKERS_B[i], NULL, WORKER_B_WORK, &cars[0]);
}
for (int i = 0; i < NUM_THREAD_C; i++)
{
    pthread_create(&WORKERS_C[i], NULL, WORKER_C_WORK, &cars[0]);
}
for (int i = 0; i < NUM_THREAD_D; i++)
{
    pthread_create(&WORKERS_D[i], NULL, WORKER_D_WORK, &cars[0]);
}

// wait for days to finish
pthread_join(DAY_ITERATOR, NULL);

// join all threads
for (int i = 0; i < NUM_THREAD_A; i++)
{
    pthread_join(WORKERS_A[i], NULL);
}
for (int i = 0; i < NUM_THREAD_B; i++)
{
    pthread_join(WORKERS_B[i], NULL);
}
for (int i = 0; i < NUM_THREAD_C; i++)
{       You, 3 seconds ago • Uncommitted changes
    pthread_join(WORKERS_C[i], NULL);
}
for (int i = 0; i < NUM_THREAD_D; i++)
{
    pthread_join(WORKERS_D[i], NULL);
}
```

Here is an example of what a thread function looks like.

```
void *WORKER_A_WORK(void *args)
{
    car *c = (car *)args;
    place_tires(c);
    paint_car(c);
    return NULL;
}
```

In this case WORKER-A can place tires and paint cars. Here is the implementation of placing tires.

```
void place_tires(car *c)
{
    // tires wait for chasis to be ready
    // lock the car's mutex
    // wait for chasis lock
    while (c->tiresDone == 1 || c->isCarReady == 1)
    {
        // if the current car is locked then go to the next car
        c = &cars[c->CarID + 1];
    }
    sem_wait(&c->chasis);
    pthread_mutex_lock(&c->mutex);

    c->currentWorker.workerID = pthread_self();
    c->currentWorker.workerType = WORKER_A;
    c->currentWorker.currentPartName = TIRES;
    c->tiresDone = 1;
    // decrement daily tires remaining
    pthread_mutex_lock(&mutex_tires);
    NUM_DAILY_TIRES_REMAINING--;
    pthread_mutex_unlock(&mutex_tires);

    printf("%s-%u\t%d\t%s\t%d\n",
           c->currentWorker.workerType, c->currentWorker.workerID,
           c->CarID, c->currentWorker.currentPartName, CURRENT_DAY);

    // signal that chasis is ready  // signal that tires are ready
    sem_post(&c->tires);

    // unlock the car's mutex
    pthread_mutex_unlock(&c->mutex);
}
```

It first checks if a given car's tires are already placed or the car is ready, if so it continues with the next car. If not then it waits for the chassis to be done, after

that it locks the car's mutex, then it does its work, prints it , at the end it notifies the tires are done (mounting seats require it) and it unlocks the mutex.

Here is the implementation of the day simulator.

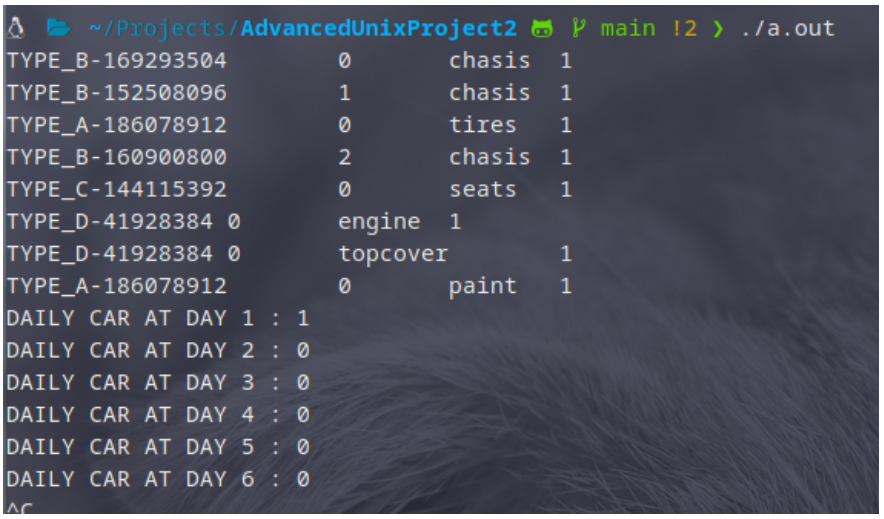
```
void *iterate_day(void *arg)
{
    for (CURRENT_DAY = 1; CURRENT_DAY <= NUM_DAYS; CURRENT_DAY++)
    {
        sleep(3);

        printf("DAILY CAR AT DAY %d : %d\n", CURRENT_DAY, DAILY_CAR);

        pthread_mutex_lock(&mutex_chasis);
        NUM_DAILY_CHASIS_REMAINING = NUM_DAILY_CHASIS;
        pthread_mutex_unlock(&mutex_chasis);

        pthread_mutex_lock(&mutex_engines);
        NUM_DAILY_ENGINES_REMAINING = NUM_DAILY_ENGINES;
        pthread_mutex_unlock(&mutex_engines);

        pthread_mutex_lock(&mutex_tires);
        NUM_DAILY_TIRES_REMAINING = NUM_DAILY_TIRES;
        pthread_mutex_unlock(&mutex_tires);

        pthread_mutex_lock(&mutex_seats);
        NUM_DAILY_SEATS_REMAINING = NUM_DAILY_SEATS;
        pthread_mutex_unlock(&mutex_seats);

        pthread_mutex_lock(&mutex_topcovers);
        NUM_DAILY_TOPCOVERS_REMAINING = NUM_DAILY_TOPCOVERS;
        pthread_mutex_unlock(&mutex_topcovers);

        pthread_mutex_lock(&mutex_paints);
        NUM_DAILY_PAINTS_REMAINING = NUM_DAILY_PAINTS;
        pthread_mutex_unlock(&mutex_paints);
        DAILY_CAR = 0;
    }
    pthread_exit(NULL);
}
```

It sleeps 3 seconds for each day and at the end of the day it resets the global material variables.

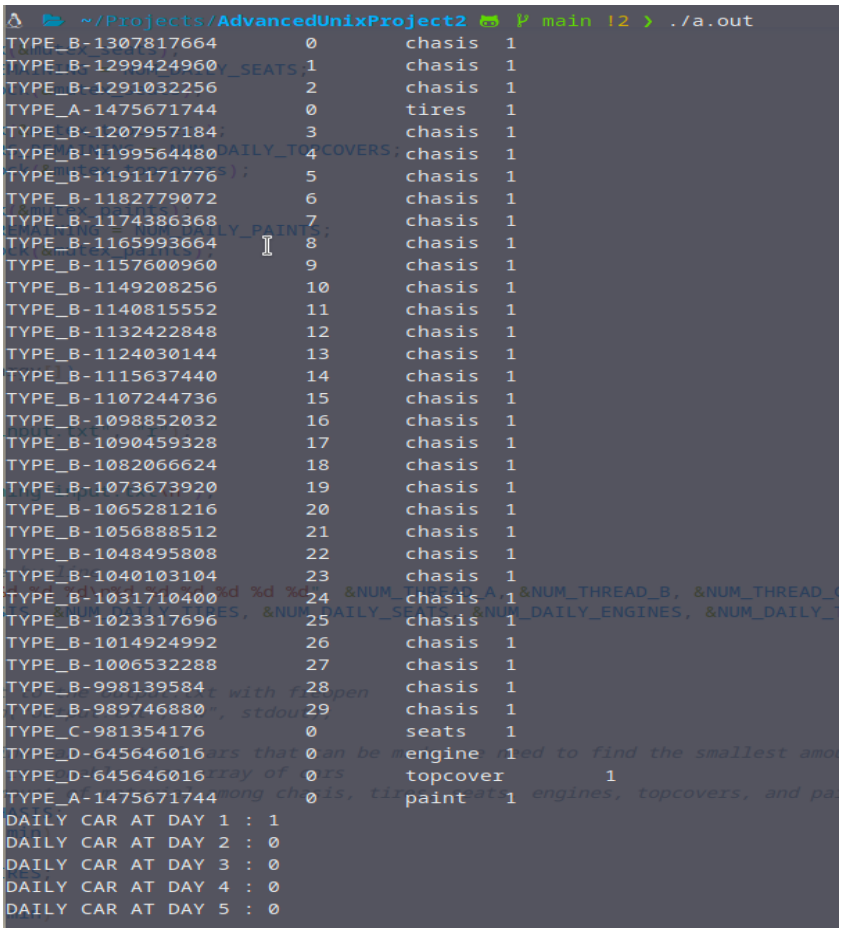
Sample Run

Disclaimer: Our project is not working fully. We have run into some problems.

Execution 1:



Execution 2: (After adding more threads)



It seems the threads cant pass after the chassis phase.