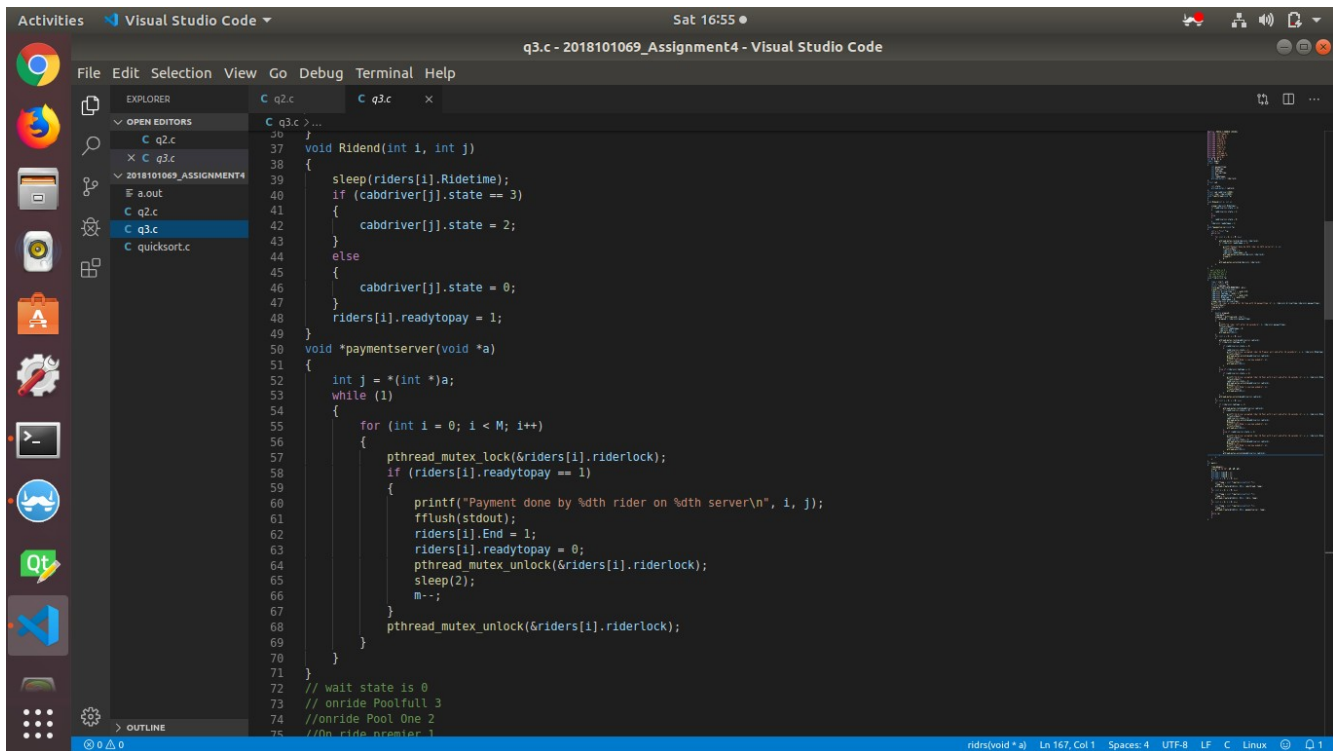


Report for question 3

This code snippet shows function for threads of payment server and function Ridend()



The screenshot shows the Visual Studio Code editor with a C file named q3.c. The code defines two functions: `Ridend` and `paymentserver`. The `Ridend` function takes a rider index `i` and a cab index `j` as arguments. It first sleeps for a duration specified by `riders[i].Ridetime`. Then, it checks the state of the cab `cabdriver[j]`. If the state is 3, it sets the state to 2. Otherwise, it sets the state to 0. After this, it sets `riders[i].readytopay = 1`. The `paymentserver` function takes a pointer to a `void *` argument. It declares an integer `j` and enters a `while (1)` loop. Inside the loop, it iterates over a range of `i` from 0 to `M-1`. For each `i`, it locks a mutex `riders[i].riderlock`. It then checks if `riders[i].readytopay == 1`. If true, it prints a message, flushes the output, sets `riders[i].End = 1`, resets `riders[i].readytopay = 0`, unlocks the mutex, sleeps for 2 seconds, and decrements a counter `m`. Finally, it unlocks the mutex. The `paymentserver` function ends with a `while` loop that waits for the state to be 0, and comments indicating the state of the pool.

```
36 void Ridend(int i, int j)
37 {
38     sleep(riders[i].Ridetime);
39     if (cabdriver[j].state == 3)
40     {
41         cabdriver[j].state = 2;
42     }
43     else
44     {
45         cabdriver[j].state = 0;
46     }
47     riders[i].readytopay = 1;
48 }
49
50 void *paymentserver(void *a)
51 {
52     int j = *(int *)a;
53     while (1)
54     {
55         for (int i = 0; i < M; i++)
56         {
57             pthread_mutex_lock(&riders[i].riderlock);
58             if (riders[i].readytopay == 1)
59             {
60                 printf("Payment done by %dth rider on %dth server\n", i, j);
61                 fflush(stdout);
62                 riders[i].End = 1;
63                 riders[i].readytopay = 0;
64                 pthread_mutex_unlock(&riders[i].riderlock);
65                 sleep(2);
66                 m--;
67             }
68             pthread_mutex_unlock(&riders[i].riderlock);
69         }
70     }
71 }
72 // wait state is 0
73 // onride Poolfull 3
74 //onride Pool One 2
75 //On ride greater 1
```

Threads of payment server tries to find rider which is ready to pay.

Function Ridend() waits until ride is over for passenger I and changes state of passenger I and the cab j.

This snippet shows riders thread whose function is to create threads for rider and find available cabs and leave if max wait time exceeds.

```
171 // cab pool size
172 #define CABS 10
173 // rider pool size
174 #define RIDERS 10
175 // payment server size
176 #define PAYMENTS 10
177 // mutex for cabs
178 pthread_mutex_t cabs_mutex = PTHREAD_MUTEX_INITIALIZER;
179 // mutex for riders
180 pthread_mutex_t riders_mutex = PTHREAD_MUTEX_INITIALIZER;
181 // mutex for payments
182 pthread_mutex_t payments_mutex = PTHREAD_MUTEX_INITIALIZER;
183 // condition variable for cabs
184 pthread_cond_t cabs_cv = PTHREAD_COND_INITIALIZER;
185 // condition variable for riders
186 pthread_cond_t riders_cv = PTHREAD_COND_INITIALIZER;
187 // condition variable for payments
188 pthread_cond_t payments_cv = PTHREAD_COND_INITIALIZER;
189 // array of cabs
190 int cabs[CABS];
191 // array of riders
192 int riders[RIDERS];
193 // array of payments
194 int payments[PAYMENTS];
195 // main function
196 int main()
197 {
198     time_t start, end;
199     printf("Starting program\n");
200     time(&start);
201     // create cabs
202     for (int i = 0; i < CABS; i++)
203     {
204         cabs[i] = rand() % 1000;
205         pthread_create(&tid1[i], NULL, cabthread, &i);
206     }
207     // create riders
208     for (int i = 0; i < RIDERS; i++)
209     {
210         riders[i] = rand() % 1000;
211         pthread_create(&tid2[i], NULL, riderthread, &i);
212     }
213     // create payments
214     for (int i = 0; i < PAYMENTS; i++)
215     {
216         payments[i] = rand() % 1000;
217         pthread_create(&tid3[i], NULL, paymentserver, &i);
218     }
219     // wait for all threads to finish
220     for (int i = 0; i < CABS; i++)
221         pthread_join(tid1[i], NULL);
222     for (int i = 0; i < RIDERS; i++)
223         pthread_join(tid2[i], NULL);
224     for (int i = 0; i < PAYMENTS; i++)
225         pthread_join(tid3[i], NULL);
226     time(&end);
227     printf("Program finished\n");
228     return 0;
229 }
```

This snippet shows Main() function which creates threads for cab ,riders and payment servers. It also contains a while loop() which will end when all riders gets cab or leaves.

```
171 // cab pool size
172 #define CABS 10
173 // rider pool size
174 #define RIDERS 10
175 // payment server size
176 #define PAYMENTS 10
177 // mutex for cabs
178 pthread_mutex_t cabs_mutex = PTHREAD_MUTEX_INITIALIZER;
179 // mutex for riders
180 pthread_mutex_t riders_mutex = PTHREAD_MUTEX_INITIALIZER;
181 // mutex for payments
182 pthread_mutex_t payments_mutex = PTHREAD_MUTEX_INITIALIZER;
183 // condition variable for cabs
184 pthread_cond_t cabs_cv = PTHREAD_COND_INITIALIZER;
185 // condition variable for riders
186 pthread_cond_t riders_cv = PTHREAD_COND_INITIALIZER;
187 // condition variable for payments
188 pthread_cond_t payments_cv = PTHREAD_COND_INITIALIZER;
189 // array of cabs
190 int cabs[CABS];
191 // array of riders
192 int riders[RIDERS];
193 // array of payments
194 int payments[PAYMENTS];
195 // main function
196 int main()
197 {
198     time(&begin);
199     scanf("%d %d %d", &N, &M, &K);
200     m = M;
201     pthread_t tid1[M + 1];
202     pthread_t tid2[N + 1];
203     pthread_t tid3[K + 1];
204     for (int i = 0; i < N; i++)
205     {
206         int *temp = (int *)malloc(sizeof(int *));
207         *temp = i;
208         pthread_create(&tid1[i], NULL, cabstthread, temp);
209     }
210     for (int i = 0; i < M; i++)
211     {
212         int *temp = (int *)malloc(sizeof(int *));
213         *temp = i;
214         pthread_create(&tid2[i], NULL, ridrs, temp);
215     }
216     for (int i = 0; i < K; i++)
217     {
218         int *temp = (int *)malloc(sizeof(int *));
219         *temp = i;
220         pthread_create(&tid3[i], NULL, paymentserver, temp);
221     }
222     while (m)
223     {
224     }
225 }
```