Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Bakhshi	Student ID:	20605126
Other name(s):	Sohail/Sohailharoon		
Unit name:	Operating Systems	Unit ID:	COMP2006
Lecturer / unit coordinator:	Sie Teng Soh	Tutor:	Jordan
Date of submission:	3/4/23	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: ______________________ Date of signature: 3/4/23

(By submitting this form, you indicate that you agree with all the above text.)

Operating Systems Assignment Report

Name: Sohail Bakhshi
ID: 20605126

Table of Contents

Source Code

Main.c

```c
//Name: Sohail Bakhshi
//ID: 20605126
// 2023 OS Assignment
// main.c contains the main function which contains threads running the customer and teller functions
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "linkedlist.h"
#include "functions.h"
#include "teller.h"
#include "customer.h"



int main(int argc, char *argv[]){
    int queueSize = atoi(argv[1]);
    int customerTime = atoi(argv[2]);
    int withdrawTime = atoi(argv[3]);
    int depositTime = atoi(argv[4]);
    int informationTime = atoi(argv[5]);
    int startHour, startMin, startSec;
    if (argc != 6){
        printf("----------------------------\nError: To run enter ./main m tc tw td ti\nQueue Size (m)\nTime for customers to enter queue (tc)\nTime to withdraw (tw)\nTime to deposit(td)\nTime to get information(ti)\n----------------------------\n");
    }
    else{
        resetLog(); //reset the log file each time i run the program
        pthread_t customerThread; //customer thread
        pthread_t teller1; //teller thread
        pthread_t teller2; //teller thread
        pthread_t teller3; //teller thread
        pthread_t teller4; //teller thread
        linkedlist * queue = createLinkedList(queueSize); //intialising fifo linked list / queue
        customerArgs customerArguments = {queue,customerTime}; // parameters for my customer function used in a struct to pass to customer thread
        getTime(&startHour, &startMin, &startSec);//get the start time



        // parameters for my teller function used in a struct to pass to teller thread
        tellerArgs teller1Args = {queue, 1,startHour,startMin,startSec,withdrawTime,depositTime,informationTime};
        tellerArgs teller2Args = {queue, 2,startHour,startMin,startSec,withdrawTime,depositTime,informationTime};
        tellerArgs teller3Args = {queue, 3,startHour,startMin,startSec,withdrawTime,depositTime,informationTime};
        tellerArgs teller4Args = {queue, 4,startHour,startMin,startSec,withdrawTime,depositTime,informationTime};

        pthread_mutex_init(&tellerMutex, NULL); // mutex initialised
        pthread_cond_init(&tellerCond, NULL); // condition initialised

        //create the threads
        pthread_create(&customerThread,NULL,(void*)customer,(void*)&customerArguments);
        pthread_create(&teller1,NULL,(void*)teller,(void*)&teller1Args);
        pthread_create(&teller2,NULL,(void*)teller,(void*)&teller2Args);
        pthread_create(&teller3,NULL,(void*)teller,(void*)&teller3Args);
        pthread_create(&teller4,NULL,(void*)teller,(void*)&teller4Args);

        //join the threads
        pthread_join(customerThread,NULL);
        pthread_join(teller1,NULL);
        pthread_join(teller2,NULL);
        pthread_join(teller3,NULL);
        pthread_join(teller4,NULL);

        freeLinkedList(queue); //free allocated memory for the queue
        pthread_mutex_destroy(&tellerMutex); // destroy mutex
        pthread_cond_destroy(&tellerCond); // destroy condition

    }




    return 0;
}
```

Teller.c

```
//file for my teller function
//Name: Sohail Bakhshi
//ID: 20605126
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "linkedlist.h"
#include "functions.h"
#include "teller.h"
#include "customer.h"

int activeTellers= 4;
int tellerServedCount[4] = {0,0,0,0};
int totalServed=0;



void teller(void *arguments){
    tellerArgs * data = arguments;
    customerInfo * customer;
    int servedCount=0;

    while(1)
    {
        pthread_mutex_lock(&tellerMutex); //use a lock to only allow 1 thread into the critical section

        while(data->queue->count ==0 && terminateTellers ==0) // if theres no customers in the queue and the
tellers havent been told to terminate then the teller threads must wait
        {
            pthread_cond_wait(&tellerCond, &tellerMutex);
        }

        if(data->queue->count >0){
            customer = deleteFirst(data->queue); // remove customer from queue to be served
            servedCount ++; //count the serve amount
            tellerServedCount[data->tellerNo-1]++;// this will be used later so that the final teller that terminates can output the teller statistics in order with the serve count
            pthread_mutex_unlock(&tellerMutex); // unlock the mutex so the next thread can execute and serve
            pthread_cond_signal(&tellerCond);  // signal to the customer thread that a customer has been served/removed from queue
            log_response_time(customer,data);

            if (strncmp(&customer->service, "W", 1) == 0) // using a library function to compare two strings if the strings match it will return 0 this helped me with seperating the customers based off their service type
            {
                log_completion_time(customer,data,data->withdrawTime);
            }
            else if (strncmp(&customer->service, "I", 1) == 0)
            {
                log_completion_time(customer,data,data->informationTime);
            }
            else if (strncmp(&customer->service, "D", 1) == 0)
            {
                log_completion_time(customer,data,data->depositTime);
            }

            pthread_mutex_lock(&tellerMutex); // used this lock to stop multiple threads from freeing at once which was causing segmentation faults because they were freeing memory that had already been freed.
            free(customer);
            pthread_mutex_unlock(&tellerMutex);//unlock the mutex
            pthread_cond_signal(&tellerCond); // signal to the customer that it has been freed

            }

        if (terminateTellers==1 && data->queue->count ==0)
        {
            pthread_mutex_unlock(&tellerMutex);//unlock the mutex so all threads are in here
            pthread_mutex_lock(&tellerMutex); //lock the mutex so that 1 thread executes this section of code at a time
            totalServed+= servedCount;
            FILE *logFp = fopen("r_log","a");
            terminate_r_log(logFp,customer,data,servedCount);
            activeTellers --; // done this because each teller is terminating they should subtract from 4 so that the last teller can output the teller statistics
            if (activeTellers == 0) // last teller returns all teller stats to the file
            {
                fprintf(logFp,"\nTeller Statistics\n");
                for (int i = 0; i < 4; i++) {
                    fprintf(logFp, "Teller-%d Serves %d customers\n", i+1, tellerServedCount[i]);
                }
                fprintf(logFp,"\nTotal number of customers: %d customers. \n",totalServed);
                fclose(logFp);
            }
            pthread_mutex_unlock(&tellerMutex); // unlock so the next thread can execute

            break;
        }
    }
}
```

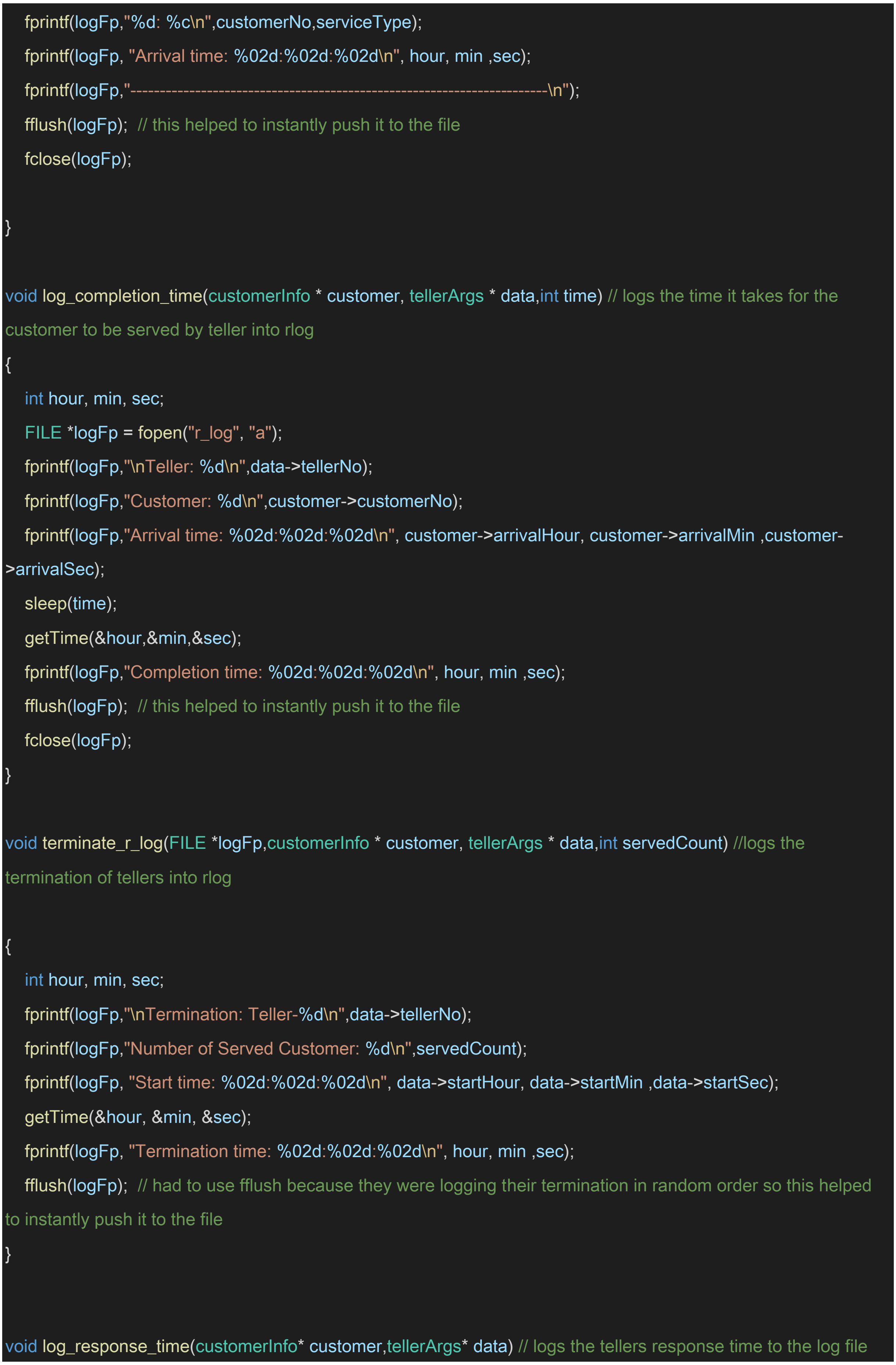
Customer.c

```
//file for my customer function
//Name: Sohail Bakhshi
//ID: 20605126
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "linkedlist.h"
#include "functions.h"
#include "teller.h"
#include "customer.h"


void customer(void *arguments)
{
    customerArgs *data = arguments;
    FILE *fpCustomerFile = fopen("c_file", "r"); // open file for reading
    int customerFile;
    while(1){
        pthread_mutex_lock(&tellerMutex); // lock
        while (data->queue->count == data->queue->queueSize)
        {
            pthread_cond_wait(&tellerCond, &tellerMutex); // make the the customer thread wait if the queue is full
        }
        if (data->queue->count != data->queue->queueSize)
        {
            pthread_mutex_unlock(&tellerMutex); //when the queue isnt full unlock the mutex and allow the thread to continue adding customers
            customerInfo *customers = (customerInfo*)(malloc(sizeof(customerInfo)));
            customerFile = fscanf(fpCustomerFile, "%d %c\n", &customers->customerNo, &customers->service);

            if (customerFile == EOF) // if it reaches the end of the file then it breaks because theres no more customers to serve
            {
                free(customers);  // free memory allocated for customers
                terminateTellers = 1; // terminate all tellers
                pthread_cond_broadcast(&tellerCond); //broadcast to all tellers saying that it has reached the end of the file and they should terminate
                break;
            }
            insertLast(data->queue, customers);
            getTime(&customers->arrivalHour,&customers->arrivalMin,&customers->arrivalSec);
            customer_r_log(customers->customerNo, customers->service,customers->arrivalHour, customers->arrivalMin, customers->arrivalSec);
            pthread_cond_broadcast(&tellerCond); // broadcast to all teller threads so all threads are awakened and not just 1 if i were to use signal
            sleep(data->customerTime); // simulates the time for customers to enter the queue
        }
    }
    fclose(fpCustomerFile);
}
```

Functions.c

```
//file for my additional functions
//Name: Sohail Bakhshi
```

```c
//ID: 20605126
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "linkedlist.h"
#include "functions.h"



void getTime(int *hour, int *min, int * sec) //got a little help from stack overflow to figure out how to use time but i did not copy //Antrromet. 2011. “Get the Current Time in C.” Stack Overflow. February 28, 2011. https://stackoverflow.com/questions/5141960/get-the-current-time-in-c.
{
    time_t timeNow;
    struct tm * timeinfo;
    time(&timeNow);
    timeinfo = localtime(&timeNow);
    *hour =timeinfo->tm_hour;
    *min = timeinfo->tm_min;
    *sec = timeinfo->tm_sec;
}

void resetLog() // resets the log file every time i run the program because it was annoying constantly manually deleting it
{
    FILE *logFp = fopen("r_log", "w");
    if (logFp == NULL) {
        printf("Error");
    }
    fclose(logFp);
}

void customer_r_log(int customerNo, char serviceType, int hour, int min, int sec) //logs when the customers arrive
{

    FILE *logFp = fopen("r_log","a");
    fprintf(logFp,"\n------------------------------------------------------------------------\n");
```



```c
    fprintf(logFp,"%d: %c\n",customerNo,serviceType);
    fprintf(logFp, "Arrival time: %02d:%02d:%02d\n", hour, min ,sec);
    fprintf(logFp,"------------------------------------------------------------------------\n");
    fflush(logFp);  // this helped to instantly push it to the file
    fclose(logFp);

}

void log_completion_time(customerInfo * customer, tellerArgs * data,int time) // logs the time it takes for the customer to be served by teller into rlog
{
    int hour, min, sec;
    FILE *logFp = fopen("r_log", "a");
    fprintf(logFp,"\nTeller: %d\n",data->tellerNo);
    fprintf(logFp,"Customer: %d\n",customer->customerNo);
    fprintf(logFp,"Arrival time: %02d:%02d:%02d\n", customer->arrivalHour, customer->arrivalMin ,customer->arrivalSec);
    sleep(time);
    getTime(&hour,&min,&sec);
    fprintf(logFp,"Completion time: %02d:%02d:%02d\n", hour, min ,sec);
    fflush(logFp);  // this helped to instantly push it to the file
    fclose(logFp);
}

void terminate_r_log(FILE *logFp,customerInfo * customer, tellerArgs * data,int servedCount) //logs the termination of tellers into rlog

{
    int hour, min, sec;
    fprintf(logFp,"\nTermination: Teller-%d\n",data->tellerNo);
    fprintf(logFp,"Number of Served Customer: %d\n",servedCount);
    fprintf(logFp, "Start time: %02d:%02d:%02d\n", data->startHour, data->startMin ,data->startSec);
    getTime(&hour, &min, &sec);
    fprintf(logFp, "Termination time: %02d:%02d:%02d\n", hour, min ,sec);
    fflush(logFp);  // had to use fflush because they were logging their termination in random order so this helped to instantly push it to the file
}


void log_response_time(customerInfo* customer,tellerArgs* data) // logs the tellers response time to the log file
```

```
{
    int hour, min, sec;
    FILE *logFp = fopen("r_log", "a");
    fprintf(logFp,"\nTeller: %d\n",data->tellerNo);
    fprintf(logFp,"Customer: %d\n",customer->customerNo);
    getTime(&hour,&min,&sec);
    fprintf(logFp, "Arrival time: %02d:%02d:%02d\n",customer->arrivalHour, customer->arrivalMin ,customer->arrivalSec);
    fprintf(logFp, "Response time: %02d:%02d:%02d\n", hour, min ,sec);
    fflush(logFp);  // this helped to instantly push it to the file
    fclose(logFp);

}
```

Linkedlist.c

```
//updated linked list that i created in practical 4 COMP2002 Unix Systems Programming which was also inspired and created based of the content from COMP1000 Unix and C programming
//used as my c_queue
//Name: Sohail Bakhshi
//ID: 20605126
#include <stdio.h>
#include <stdlib.h>
#include "linkedlist.h"



linkedlist * createLinkedList(int queueSize)//create linked list data structure which i will be using as my queue
{
    linkedlist * list;
    list = (linkedlist*)malloc(sizeof(linkedlist));
    list->head = NULL;
    list->count = 0;
    list->queueSize = queueSize;
    return list;
}

void insertLast(linkedlist *list, void * value) //adding it to the end makes sense because every node will be in order of arrival meaning the first one will be at the front of the queue
{
    node * newNode = (node*)malloc(sizeof(node));
    node * currNode = list->head;
    newNode->value = value;
    newNode->next = NULL;
    if (list->count == list->queueSize) //if the number of customers reach the queue limit
    {
        printf("Queue is full of customers\n");
    }
    if (list->head == NULL) //if the queue is empty set the head to the new node/customer
    {
        list->head = newNode;
    }
    else // if theres already customers then iterate through the list and search for the last node and add the new node behind that node
    {
        while(currNode->next != NULL)
        {
            currNode = currNode->next;
        }
        currNode->next = newNode;
    }
    list->count++;
}
void* deleteFirst(linkedlist *list) //fifo queue removes the first in so this is needed to remove the first customer
{
    void *data = NULL;
    node *temp = list->head;

    if (list->head ==NULL) // if the linked list /queue is empty
    {
        printf("Theres no customer in the queue\n");
    }
    else if(temp->next ==NULL) // if theres only one customer in the queue
    {
        data = temp->value; //set data to the node thats getting deleted so it can be returned
        list->head = NULL; //set the head to null then free the memory allocated
        free(temp);
        temp = NULL;
    }
    else //if theres multiple customers in the queue
    {
        data = temp->value; //set data to the node thats getting deleted so it can be returned
        list->head = list->head->next; //unlink the original head value and set the head to the next value thus deleting the original
        free(temp);
        temp = NULL;
    }
    list->count--;
    return data;

}
//recursion used to free allocated memory in the linked list
void freeNode(node * listnode)
{
    if(listnode != NULL){
        freeNode(listnode->next);
        free(listnode);
    }


}
void freeLinkedList(linkedlist *list){
    freeNode(list->head);
    free(list);
}
```

README

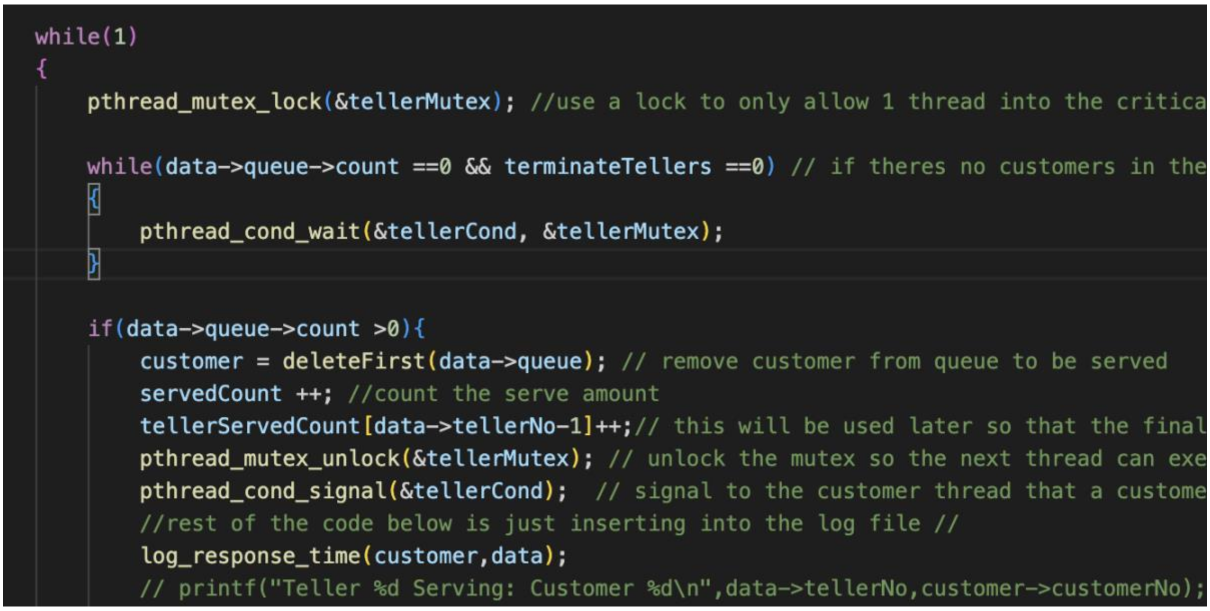
Usage Information:

To run the program, you must compile it first by entering 'make' into the terminal which will create a file called main.
Once compiled to run enter ./main m tc tw td ti where m is the queue size tc is the time for customers to enter the queue tw being withdraw time, td being deposit time and ti being information time.

Discussion

For this assignment we had to simulate a teller and customer scenario where customers would be loaded into a queue while the tellers serve and remove the customers from the queue. To achieve this, I used multiple/4 threads for my tellers and 1 thread for my customer. These threads are used to run functions called teller and customer and within these functions the main idea of my code is that in the teller function the teller removes 1 customer from the linked list/ queue to serve while the customer function adds customers to the queue until all customers are served. Now this scenario is bound to have some problems if I were to code it without the consideration of trying to achieve synchronisation. For example, we could incur a race condition where 2 threads serve the same customer twice this means that 2 threads have entered the critical section at the same time which is what we need to prevent. There could also be a scenario where the teller serves (consumes) more customers than what the producer (customer) produces also known as the producer consumer problem. For this reason, I have utilised mutexes and conditions in order to achieve mutual exclusion and synchronization within my code.

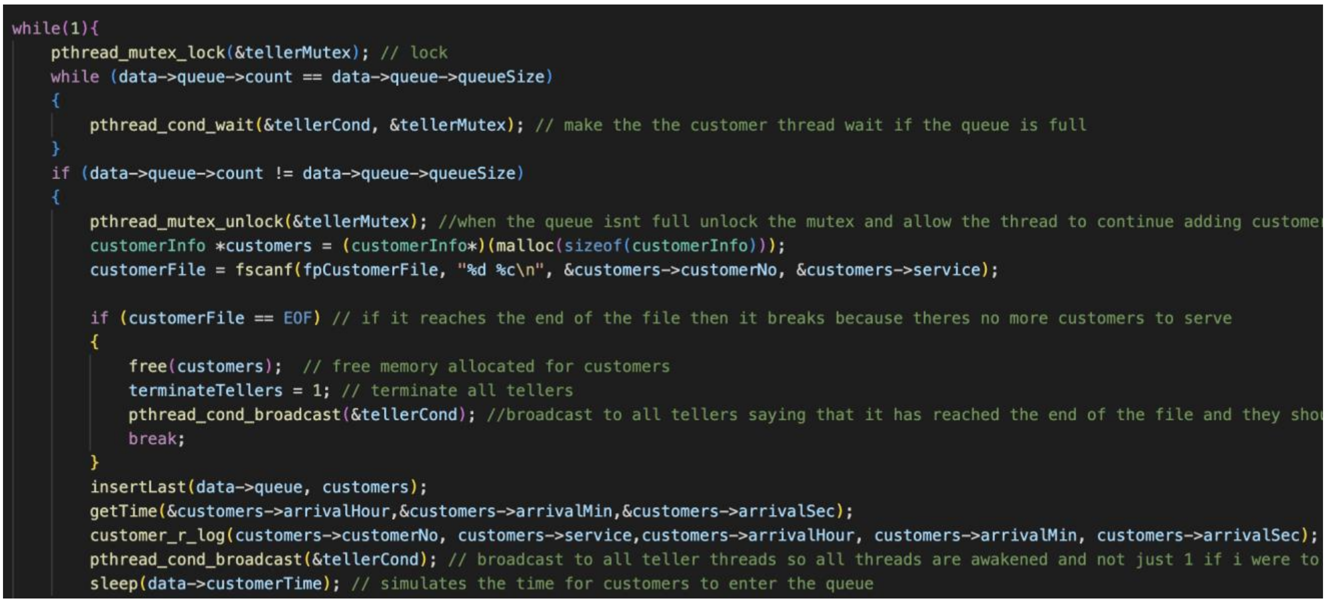
To further elucidate, within my teller function, I have used a mutex lock at the very beginning of my infinite while loop as can be seen below.



```
while(1)
{
    pthread_mutex_lock(&tellerMutex); //use a lock to only allow 1 thread into the critica

    while(data->queue->count ==0 && terminateTellers ==0) // if theres no customers in the
    {
        pthread_cond_wait(&tellerCond, &tellerMutex);
    }

    if(data->queue->count >0){
        customer = deleteFirst(data->queue); // remove customer from queue to be served
        servedCount ++; //count the serve amount
        tellerServedCount[data->tellerNo-1]++;// this will be used later so that the final
        pthread_mutex_unlock(&tellerMutex); // unlock the mutex so the next thread can exe
        pthread_cond_signal(&tellerCond);  // signal to the customer thread that a custome
        //rest of the code below is just inserting into the log file //
        log_response_time(customer,data);
        // printf("Teller %d Serving: Customer %d\n",data->tellerNo,customer->customerNo);
```

This mutex lock will allow only 1 teller thread to enter the critical section at a time. Once the customer has been removed from the queue the thread will unlock and signal to the customer that a customer has been removed from the queue and then the next teller thread will be allowed to enter its critical section. Thus, shows I am preventing a race condition and satisfying mutual exclusion.

To prevent the producer consumer problem, I have used a condition (pthread_cond_wait) were If there are no customers in the queue then the threads/tellers will wait until the customer broadcasts to all threads that there is a customer to be served. This can also be seen in my customer function where I prevent customers from being added if the queue is full. The discussion of the customer function is illustrated below.
Within the customer function I've done it in a similar way to what I did in the teller function but since there is only 1 thread running this function it doesn't really result in any race conditions. Below is a part of my customer function.



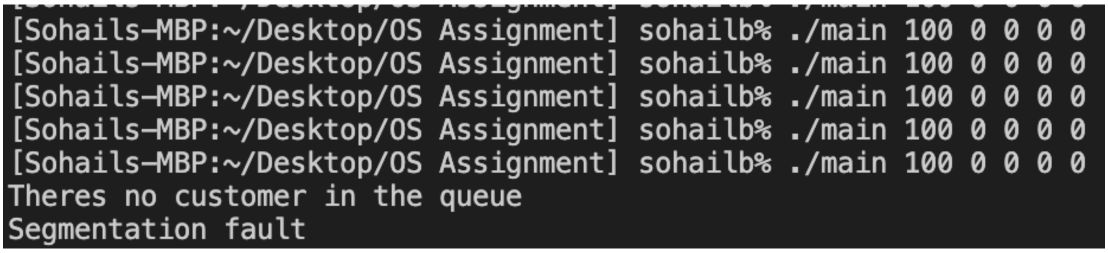
```
while(1){
    pthread_mutex_lock(&tellerMutex); // lock
    while (data->queue->count == data->queue->queueSize)
    {
        pthread_cond_wait(&tellerCond, &tellerMutex); // make the the customer thread wait if the queue is full
    }
    if (data->queue->count != data->queue->queueSize)
    {
        pthread_mutex_unlock(&tellerMutex); //when the queue isnt full unlock the mutex and allow the thread to continue adding custome
        customerInfo *customers = (customerInfo*)(malloc(sizeof(customerInfo)));
        customerFile = fscanf(fpCustomerFile, "%d %c\n", &customers->customerNo, &customers->service);

        if (customerFile == EOF) // if it reaches the end of the file then it breaks because theres no more customers to serve
        {
            free(customers);  // free memory allocated for customers
            terminateTellers = 1; // terminate all tellers
            pthread_cond_broadcast(&tellerCond); //broadcast to all tellers saying that it has reached the end of the file and they sho
            break;
        }
        insertLast(data->queue, customers);
        getTime(&customers->arrivalHour,&customers->arrivalMin,&customers->arrivalSec);
        customer_r_log(customers->customerNo, customers->service,customers->arrivalHour, customers->arrivalMin, customers->arrivalSec);
        pthread_cond_broadcast(&tellerCond); // broadcast to all teller threads so all threads are awakened and not just 1 if i were to
        sleep(data->customerTime); // simulates the time for customers to enter the queue
```

But since there could be a situation where there are too many customers in the queue and queue limit has reached then the customer should not add any more customers to the queue. For this reason, I have added a mutex lock at the beginning of the infinite loop where the lock will unlock if there is space in the queue. If not, the thread will wait until the teller has served and signalled to the customer thread that there is space to add more customers. And at the end of the if statement the thread will broadcast to all the tellers saying theres customers in the queue. Thus, preventing the producer consumer problem.

Tests

Issues:

So, in very rare cases when I run my code very quickly by spamming it and setting the time for the tellers to serve and the customers arrival time all to 0 (as in ./main 100 0 0 0 0) sometimes I get a segmentation fault but it happens like maybe 1/20 times and I'm not too sure what the reason for this is.



```
[Sohails-MBP:~/Desktop/OS Assignment] sohailb% ./main 100 0 0 0 0
[Sohails-MBP:~/Desktop/OS Assignment] sohailb% ./main 100 0 0 0 0
[Sohails-MBP:~/Desktop/OS Assignment] sohailb% ./main 100 0 0 0 0
[Sohails-MBP:~/Desktop/OS Assignment] sohailb% ./main 100 0 0 0 0
[Sohails-MBP:~/Desktop/OS Assignment] sohailb% ./main 100 0 0 0 0
Theres no customer in the queue
Segmentation fault
```

However, the good news is that as long as the sleep times are greater than 0 then my program will not fault at all. For example, something like ./main 10 1 2 3 4 would never fault. I also think that this fault could be caused due to how my computer runs threads on the CPU as I found I barely got any segmentation faults when running it on curtins VMware.

Sample inputs and outputs

Example 1:

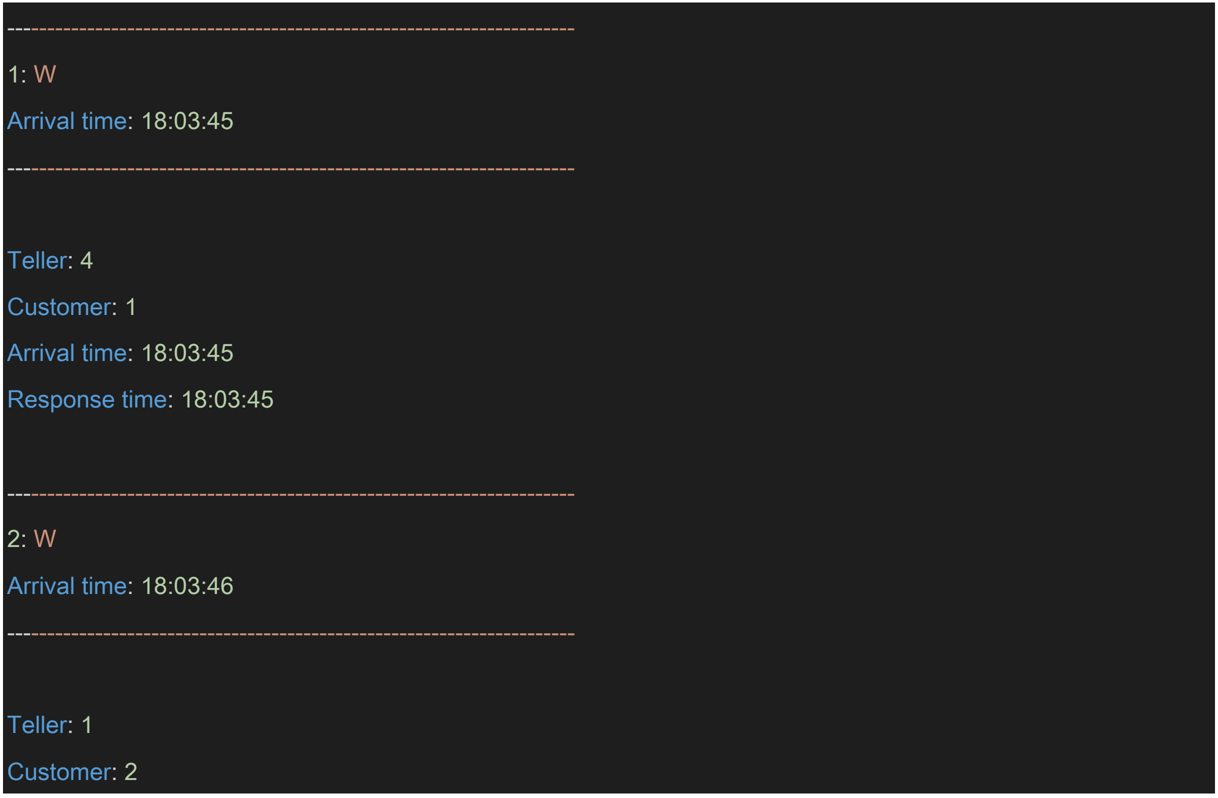
Input: ./main 10 1 2 3 4

```
sohailb% ./main 10 1 2 3 4
```

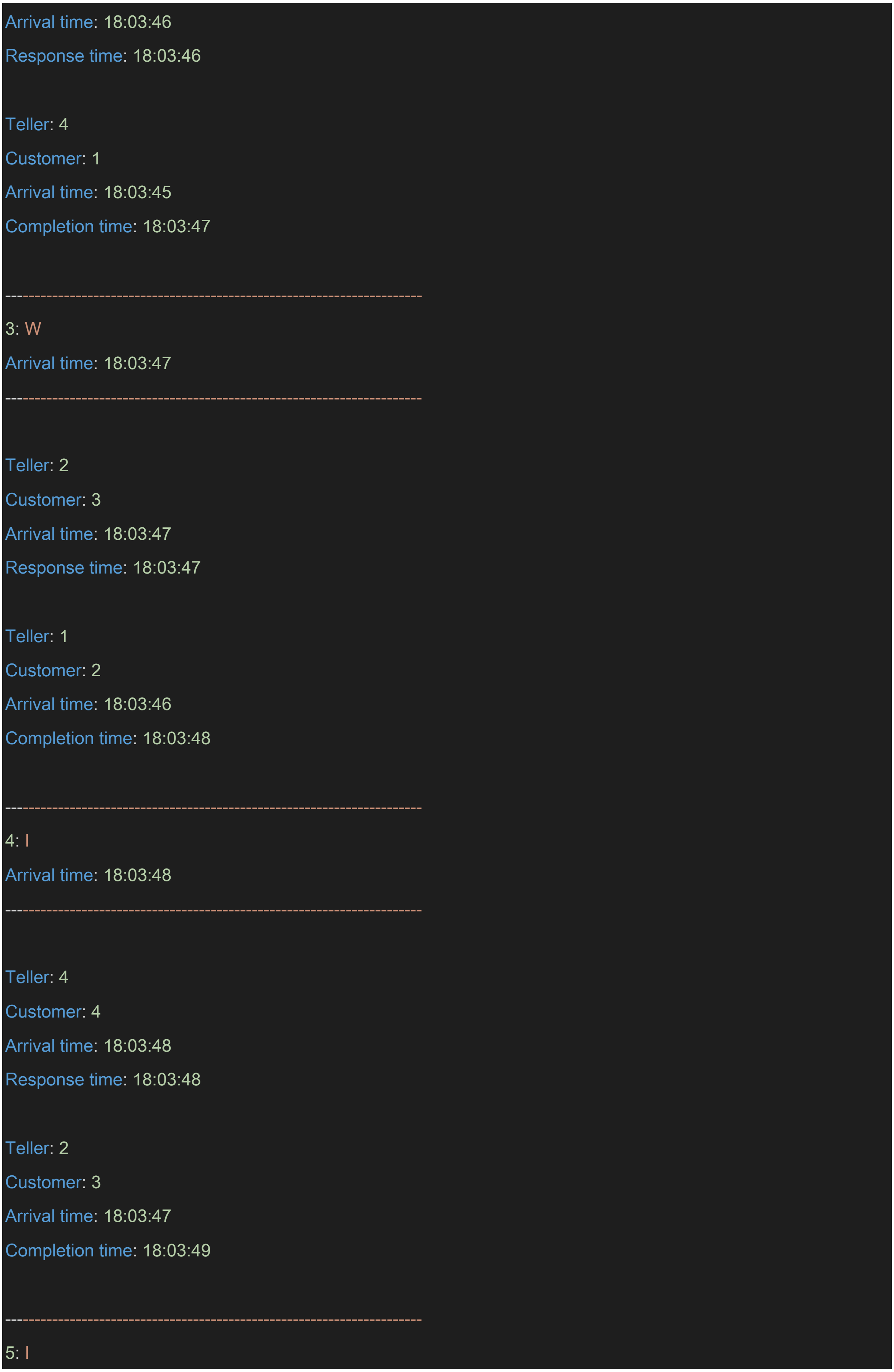
For this example, I'm only going to use 20 customers so I can show the whole output without it becoming 10000 lines.

Output:

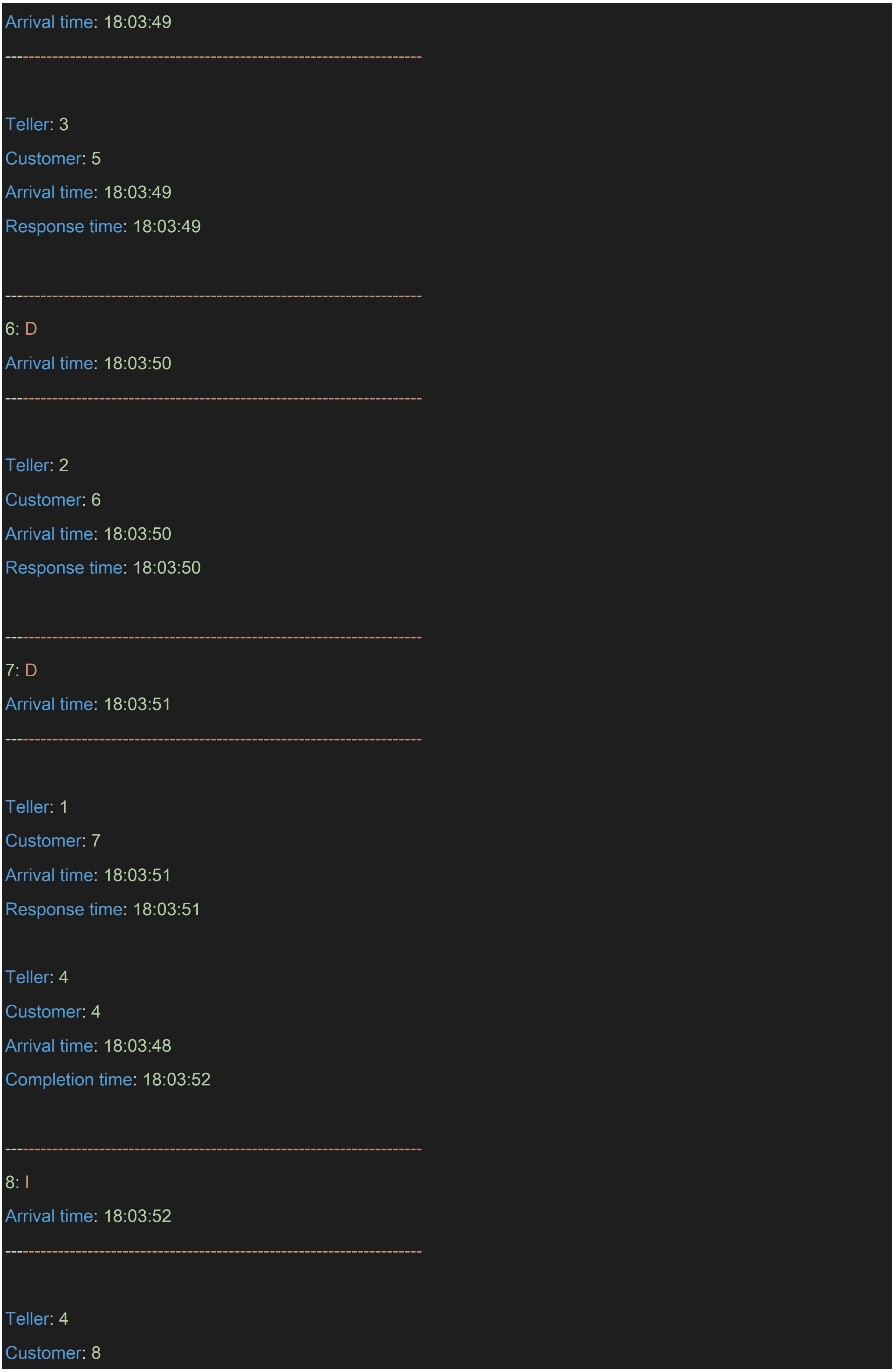
r_log



```
--------------------------------------------------------------------
1: W
Arrival time: 18:03:45
--------------------------------------------------------------------


Teller: 4
Customer: 1
Arrival time: 18:03:45
Response time: 18:03:45


--------------------------------------------------------------------
2: W
Arrival time: 18:03:46
--------------------------------------------------------------------


Teller: 1
Customer: 2
```

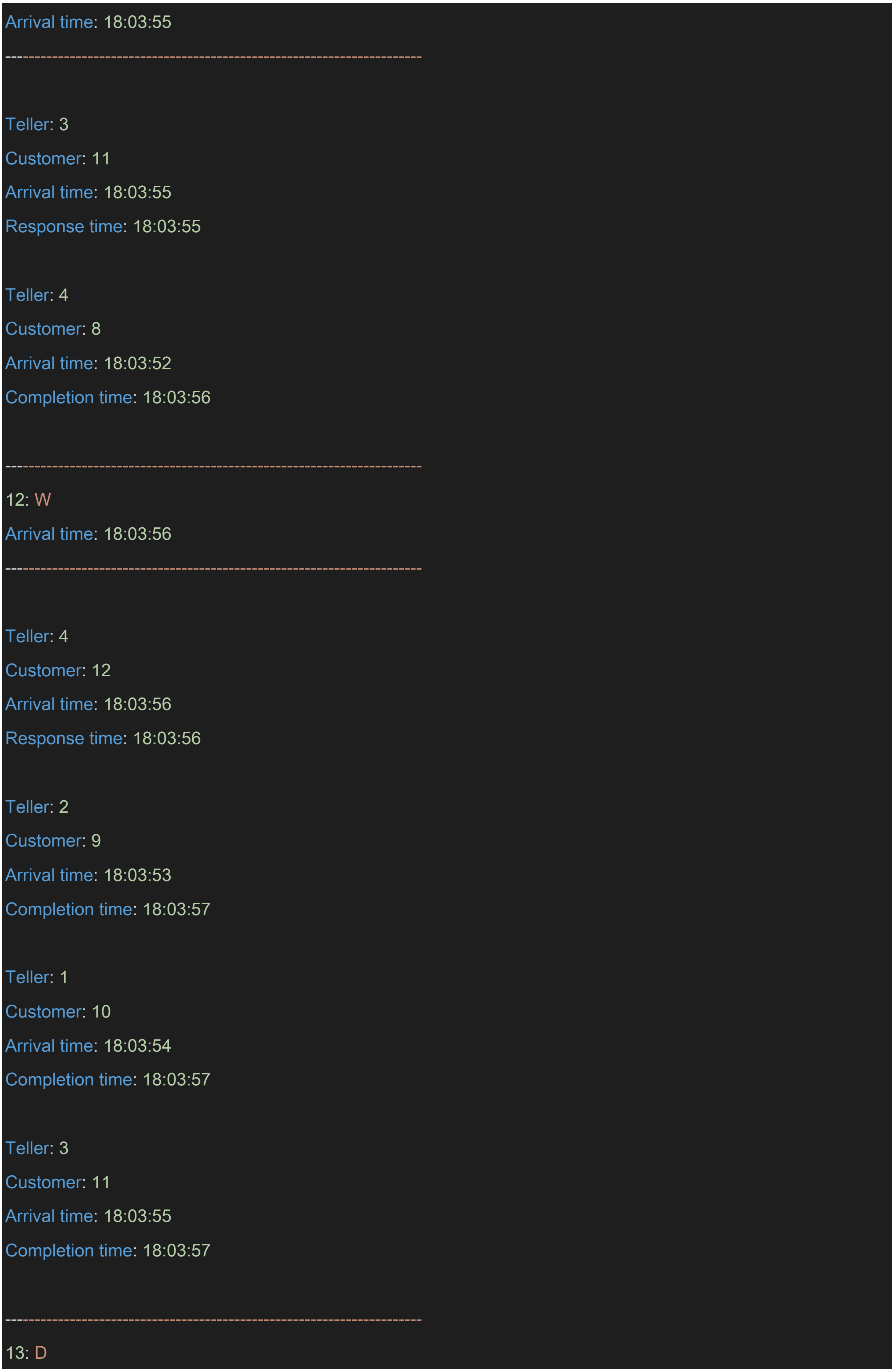


```
Arrival time: 18:03:46
Response time: 18:03:46

Teller: 4
Customer: 1
Arrival time: 18:03:45
Completion time: 18:03:47

------------------------------------------------------------------------
3: W
Arrival time: 18:03:47
------------------------------------------------------------------------

Teller: 2
Customer: 3
Arrival time: 18:03:47
Response time: 18:03:47

Teller: 1
Customer: 2
Arrival time: 18:03:46
Completion time: 18:03:48

------------------------------------------------------------------------
4: I
Arrival time: 18:03:48
------------------------------------------------------------------------

Teller: 4
Customer: 4
Arrival time: 18:03:48
Response time: 18:03:48

Teller: 2
Customer: 3
Arrival time: 18:03:47
Completion time: 18:03:49

------------------------------------------------------------------------
5: I
```

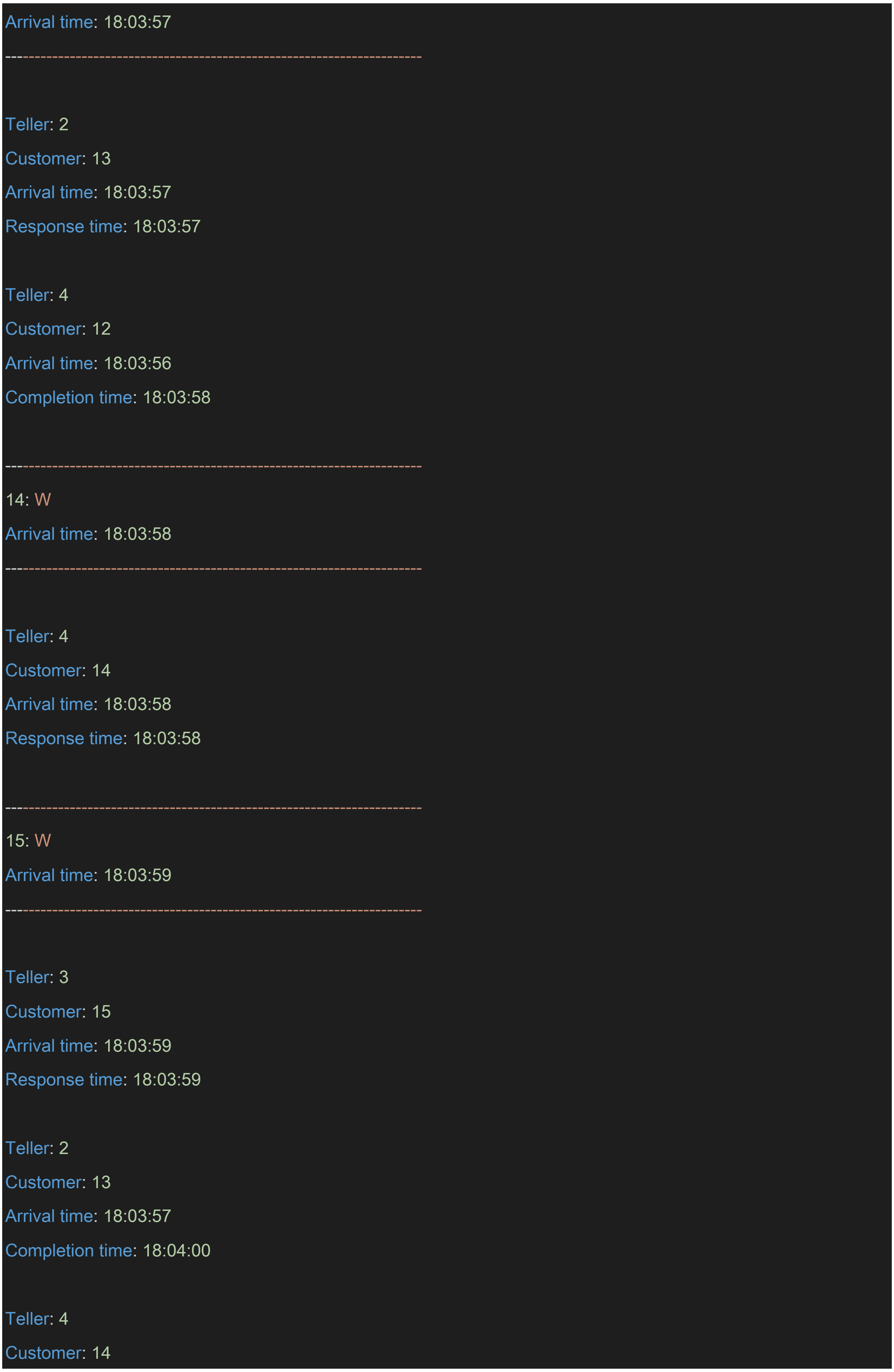


```
Arrival time: 18:03:49
-----------------------------------------------------------------------


Teller: 3
Customer: 5
Arrival time: 18:03:49
Response time: 18:03:49


-----------------------------------------------------------------------
6: D
Arrival time: 18:03:50
-----------------------------------------------------------------------


Teller: 2
Customer: 6
Arrival time: 18:03:50
Response time: 18:03:50


-----------------------------------------------------------------------
7: D
Arrival time: 18:03:51
-----------------------------------------------------------------------


Teller: 1
Customer: 7
Arrival time: 18:03:51
Response time: 18:03:51

Teller: 4
Customer: 4
Arrival time: 18:03:48
Completion time: 18:03:52


-----------------------------------------------------------------------
8: I
Arrival time: 18:03:52
-----------------------------------------------------------------------


Teller: 4
Customer: 8
```

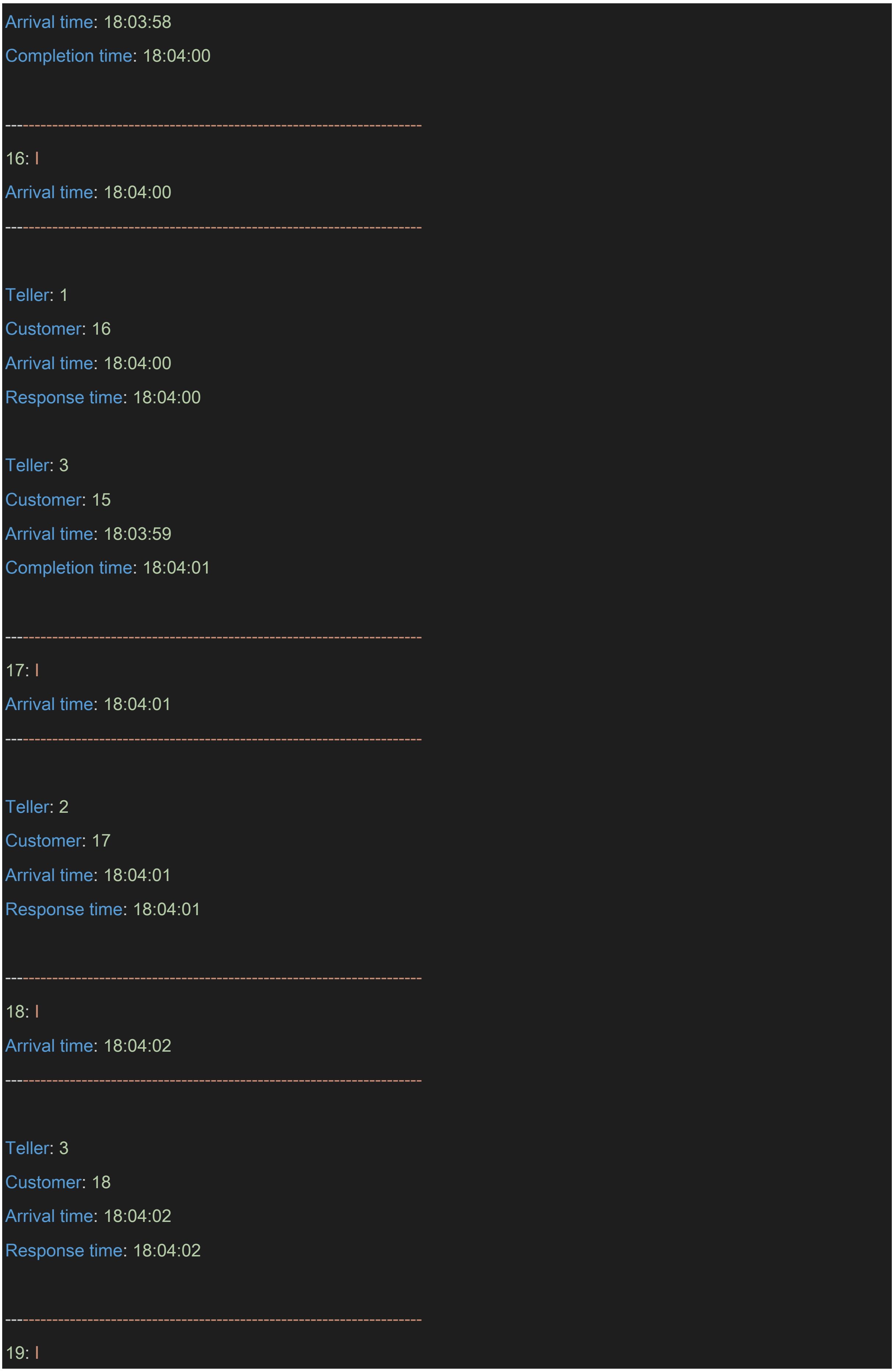
```
Arrival time: 18:03:52
Response time: 18:03:52

Teller: 3
Customer: 5
Arrival time: 18:03:49
Completion time: 18:03:53

Teller: 2
Customer: 6
Arrival time: 18:03:50
Completion time: 18:03:53

------------------------------------------------------------------
9: I
Arrival time: 18:03:53
------------------------------------------------------------------

Teller: 2
Customer: 9
Arrival time: 18:03:53
Response time: 18:03:53

Teller: 1
Customer: 7
Arrival time: 18:03:51
Completion time: 18:03:54

------------------------------------------------------------------
10: D
Arrival time: 18:03:54
------------------------------------------------------------------

Teller: 1
Customer: 10
Arrival time: 18:03:54
Response time: 18:03:54

------------------------------------------------------------------
11: W
```



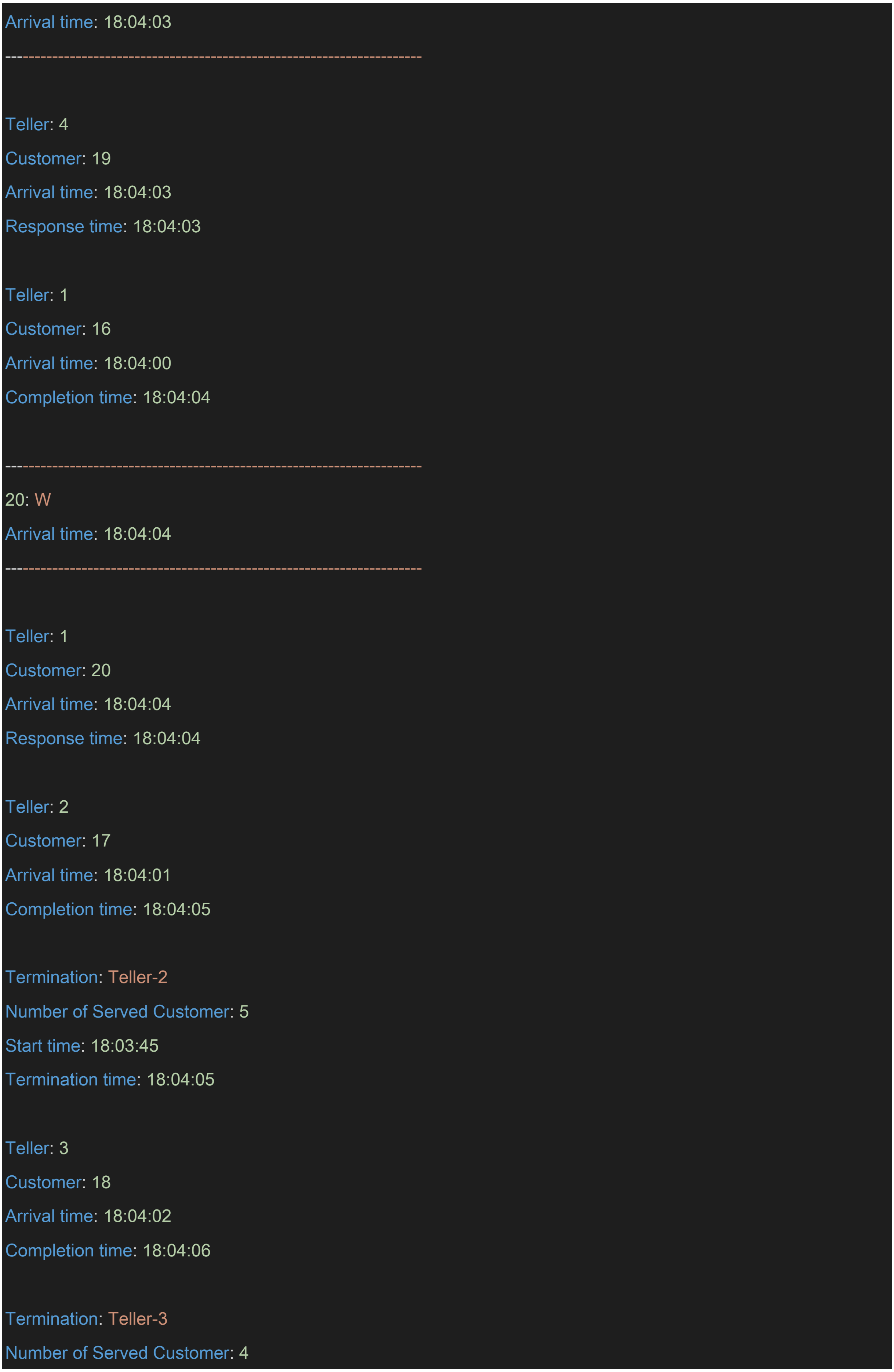
```
Arrival time: 18:03:55
------------------------------------------------------------------------


Teller: 3
Customer: 11
Arrival time: 18:03:55
Response time: 18:03:55

Teller: 4
Customer: 8
Arrival time: 18:03:52
Completion time: 18:03:56


------------------------------------------------------------------------
12: W
Arrival time: 18:03:56
------------------------------------------------------------------------


Teller: 4
Customer: 12
Arrival time: 18:03:56
Response time: 18:03:56

Teller: 2
Customer: 9
Arrival time: 18:03:53
Completion time: 18:03:57

Teller: 1
Customer: 10
Arrival time: 18:03:54
Completion time: 18:03:57

Teller: 3
Customer: 11
Arrival time: 18:03:55
Completion time: 18:03:57


------------------------------------------------------------------------
13: D
```



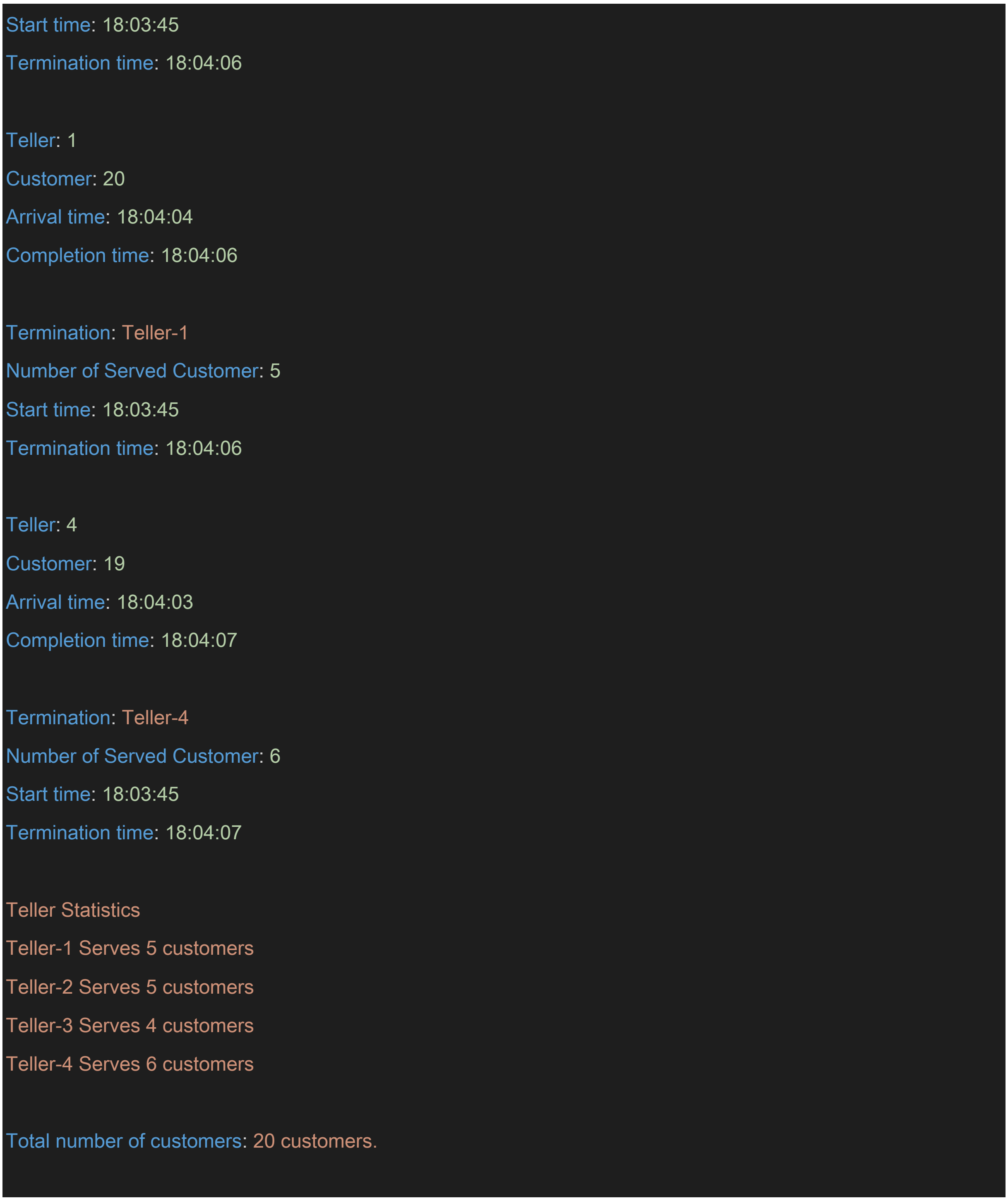
```
Arrival time: 18:03:57
------------------------------------------------------------------------


Teller: 2
Customer: 13
Arrival time: 18:03:57
Response time: 18:03:57

Teller: 4
Customer: 12
Arrival time: 18:03:56
Completion time: 18:03:58


------------------------------------------------------------------------
14: W
Arrival time: 18:03:58
------------------------------------------------------------------------


Teller: 4
Customer: 14
Arrival time: 18:03:58
Response time: 18:03:58


------------------------------------------------------------------------
15: W
Arrival time: 18:03:59
------------------------------------------------------------------------


Teller: 3
Customer: 15
Arrival time: 18:03:59
Response time: 18:03:59

Teller: 2
Customer: 13
Arrival time: 18:03:57
Completion time: 18:04:00

Teller: 4
Customer: 14
```



```
Arrival time: 18:03:58
Completion time: 18:04:00


--------------------------------------------------------------------
16: I
Arrival time: 18:04:00
--------------------------------------------------------------------


Teller: 1
Customer: 16
Arrival time: 18:04:00
Response time: 18:04:00

Teller: 3
Customer: 15
Arrival time: 18:03:59
Completion time: 18:04:01


--------------------------------------------------------------------
17: I
Arrival time: 18:04:01
--------------------------------------------------------------------


Teller: 2
Customer: 17
Arrival time: 18:04:01
Response time: 18:04:01


--------------------------------------------------------------------
18: I
Arrival time: 18:04:02
--------------------------------------------------------------------


Teller: 3
Customer: 18
Arrival time: 18:04:02
Response time: 18:04:02


--------------------------------------------------------------------
19: I
```



```
Arrival time: 18:04:03
-----------------------------------------------------------------------


Teller: 4
Customer: 19
Arrival time: 18:04:03
Response time: 18:04:03

Teller: 1
Customer: 16
Arrival time: 18:04:00
Completion time: 18:04:04


-----------------------------------------------------------------------
20: W
Arrival time: 18:04:04
-----------------------------------------------------------------------


Teller: 1
Customer: 20
Arrival time: 18:04:04
Response time: 18:04:04

Teller: 2
Customer: 17
Arrival time: 18:04:01
Completion time: 18:04:05

Termination: Teller-2
Number of Served Customer: 5
Start time: 18:03:45
Termination time: 18:04:05

Teller: 3
Customer: 18
Arrival time: 18:04:02
Completion time: 18:04:06

Termination: Teller-3
Number of Served Customer: 4
```

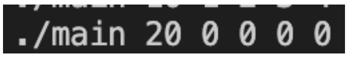


```
Start time: 18:03:45
Termination time: 18:04:06

Teller: 1
Customer: 20
Arrival time: 18:04:04
Completion time: 18:04:06

Termination: Teller-1
Number of Served Customer: 5
Start time: 18:03:45
Termination time: 18:04:06

Teller: 4
Customer: 19
Arrival time: 18:04:03
Completion time: 18:04:07

Termination: Teller-4
Number of Served Customer: 6
Start time: 18:03:45
Termination time: 18:04:07

Teller Statistics
Teller-1 Serves 5 customers
Teller-2 Serves 5 customers
Teller-3 Serves 4 customers
Teller-4 Serves 6 customers

Total number of customers: 20 customers.
```

The output for this seems 100% correct because it satisfies all requirements of the assignment and synchronisation is also achieved.

Example 2:

Input : ./main 20 0 0 0 0

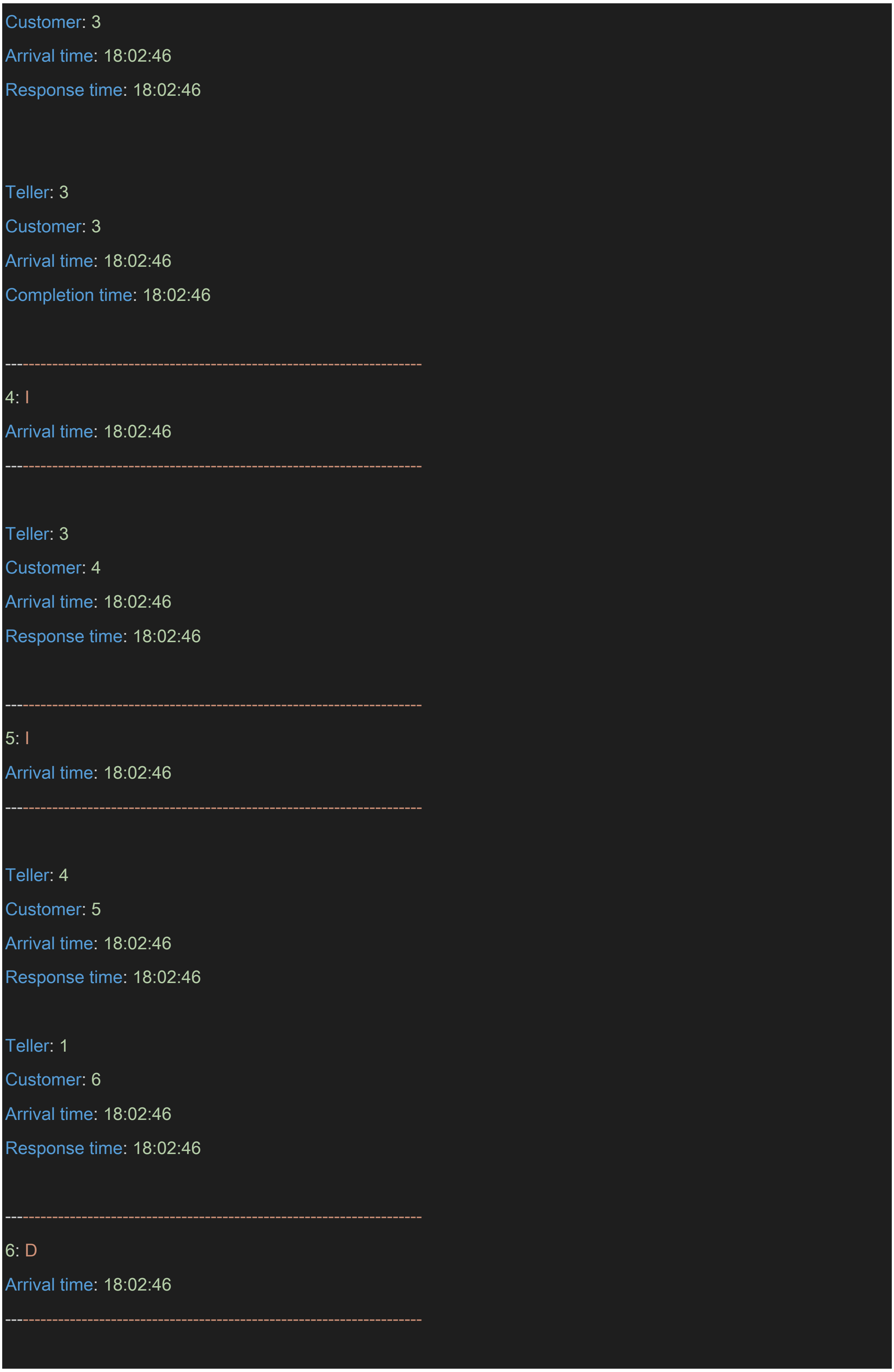


```
./main 20 0 0 0 0
```

Output:

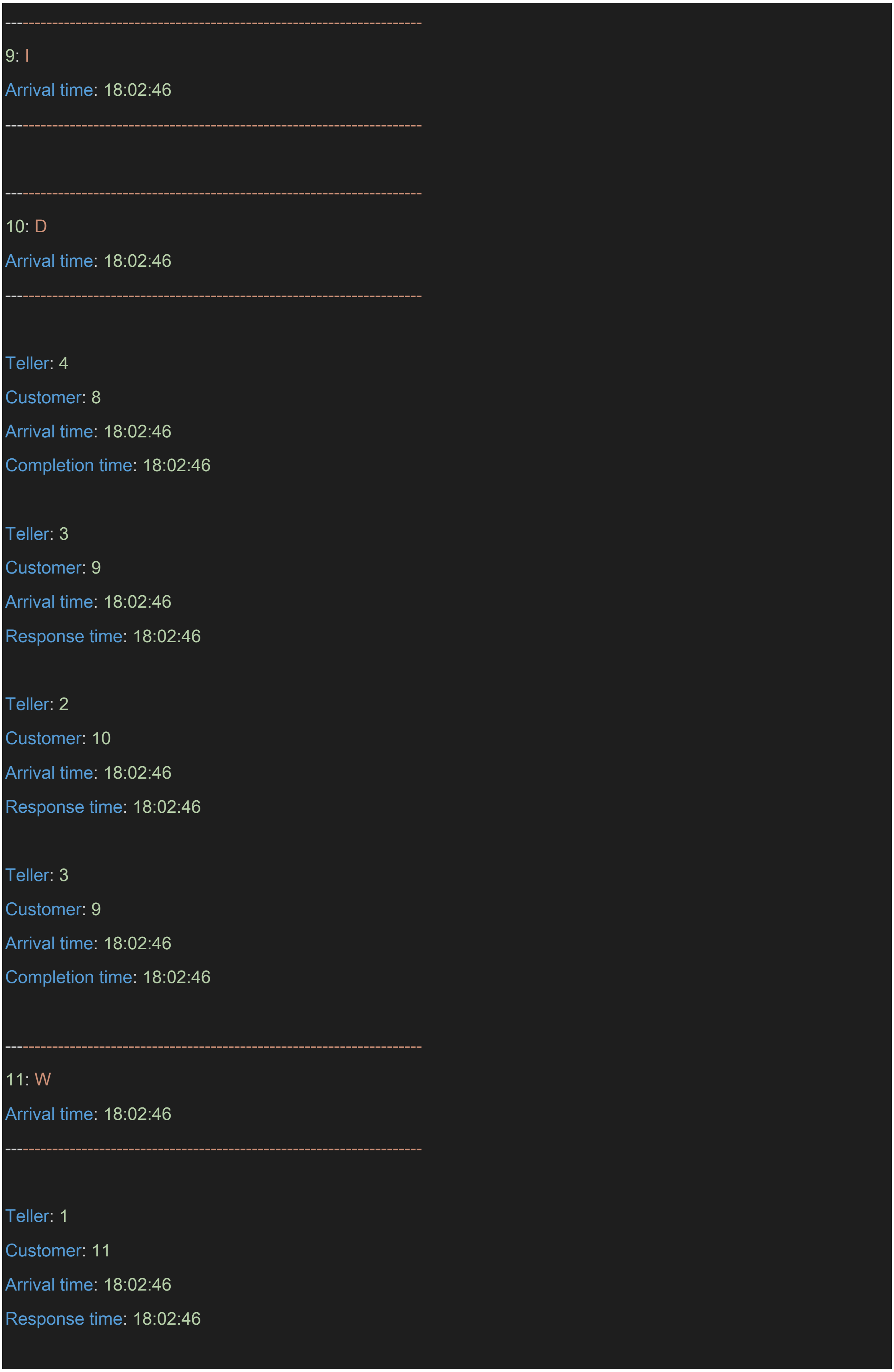
r_log

```
------------------------------------------------------------------------
1: W
Arrival time: 18:02:46
------------------------------------------------------------------------


Teller: 2
Customer: 2
Arrival time: 18:02:46
Response time: 18:02:46


------------------------------------------------------------------------
2: W
Arrival time: 18:02:46
------------------------------------------------------------------------


Teller: 1
Customer: 1
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 2
Customer: 2
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 1
Customer: 1
Arrival time: 18:02:46
Completion time: 18:02:46


------------------------------------------------------------------------
3: W
Arrival time: 18:02:46
------------------------------------------------------------------------


Teller: 3
```

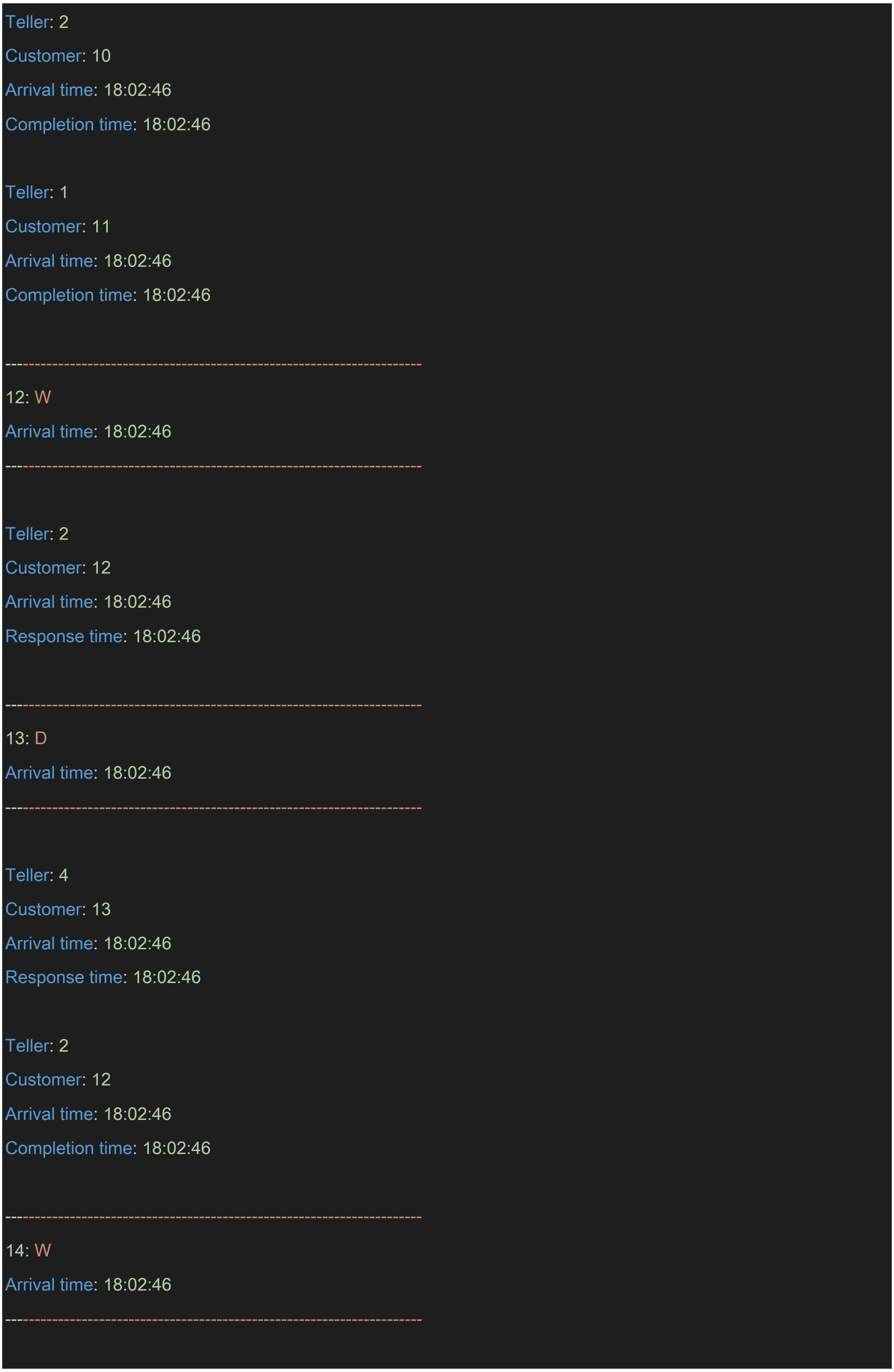


```
Customer: 3
Arrival time: 18:02:46
Response time: 18:02:46


Teller: 3
Customer: 3
Arrival time: 18:02:46
Completion time: 18:02:46

----------------------------------------------------------------------
4: I
Arrival time: 18:02:46
----------------------------------------------------------------------


Teller: 3
Customer: 4
Arrival time: 18:02:46
Response time: 18:02:46

----------------------------------------------------------------------
5: I
Arrival time: 18:02:46
----------------------------------------------------------------------


Teller: 4
Customer: 5
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 1
Customer: 6
Arrival time: 18:02:46
Response time: 18:02:46

----------------------------------------------------------------------
6: D
Arrival time: 18:02:46
----------------------------------------------------------------------
```

```
Teller: 3
Customer: 4
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 4
Customer: 5
Arrival time: 18:02:46
Completion time: 18:02:46

----------------------------------------------------------------------
7: D
Arrival time: 18:02:46
----------------------------------------------------------------------

Teller: 4
Customer: 7
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 1
Customer: 6
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 4
Customer: 7
Arrival time: 18:02:46
Completion time: 18:02:46

----------------------------------------------------------------------
8: I
Arrival time: 18:02:46
----------------------------------------------------------------------

Teller: 4
Customer: 8
Arrival time: 18:02:46
Response time: 18:02:46
```

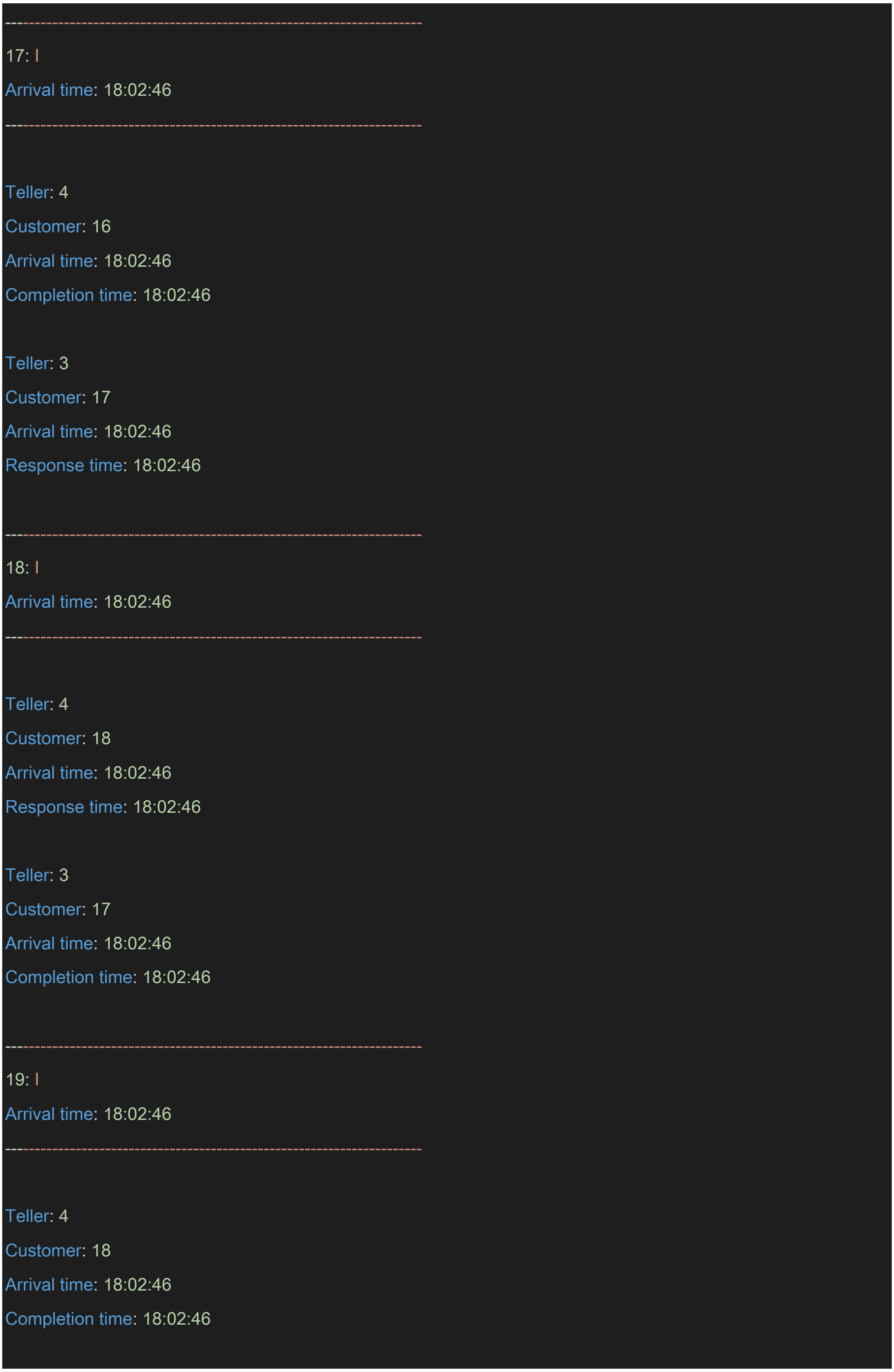


```
------------------------------------------------------------------------
9: I
Arrival time: 18:02:46
------------------------------------------------------------------------


------------------------------------------------------------------------
10: D
Arrival time: 18:02:46
------------------------------------------------------------------------


Teller: 4
Customer: 8
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 3
Customer: 9
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 2
Customer: 10
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 3
Customer: 9
Arrival time: 18:02:46
Completion time: 18:02:46


------------------------------------------------------------------------
11: W
Arrival time: 18:02:46
------------------------------------------------------------------------


Teller: 1
Customer: 11
Arrival time: 18:02:46
Response time: 18:02:46
```

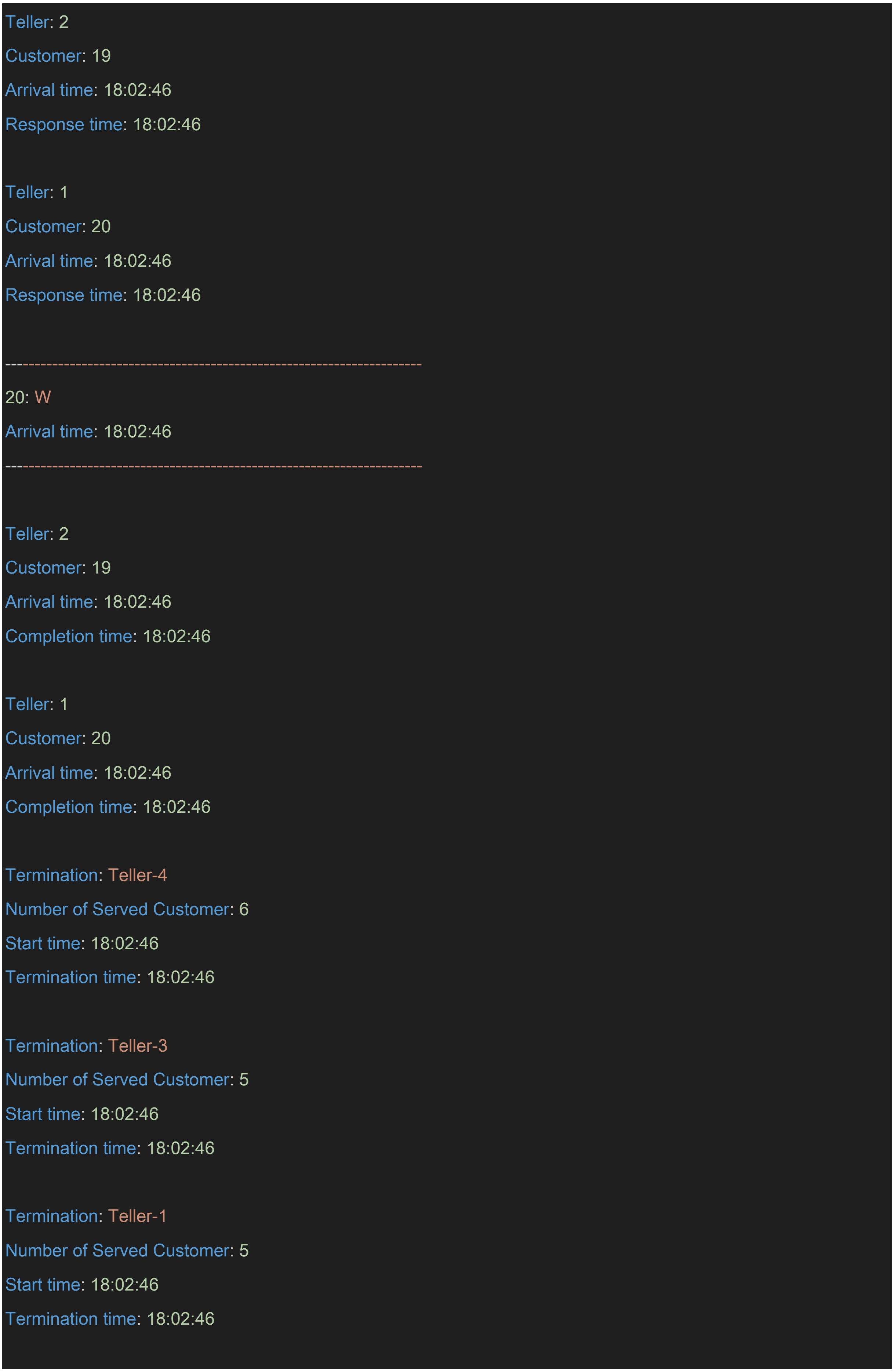


```
Teller: 2
Customer: 10
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 1
Customer: 11
Arrival time: 18:02:46
Completion time: 18:02:46

--------------------------------------------------------------------
12: W
Arrival time: 18:02:46
--------------------------------------------------------------------

Teller: 2
Customer: 12
Arrival time: 18:02:46
Response time: 18:02:46

--------------------------------------------------------------------
13: D
Arrival time: 18:02:46
--------------------------------------------------------------------

Teller: 4
Customer: 13
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 2
Customer: 12
Arrival time: 18:02:46
Completion time: 18:02:46

--------------------------------------------------------------------
14: W
Arrival time: 18:02:46
--------------------------------------------------------------------
```

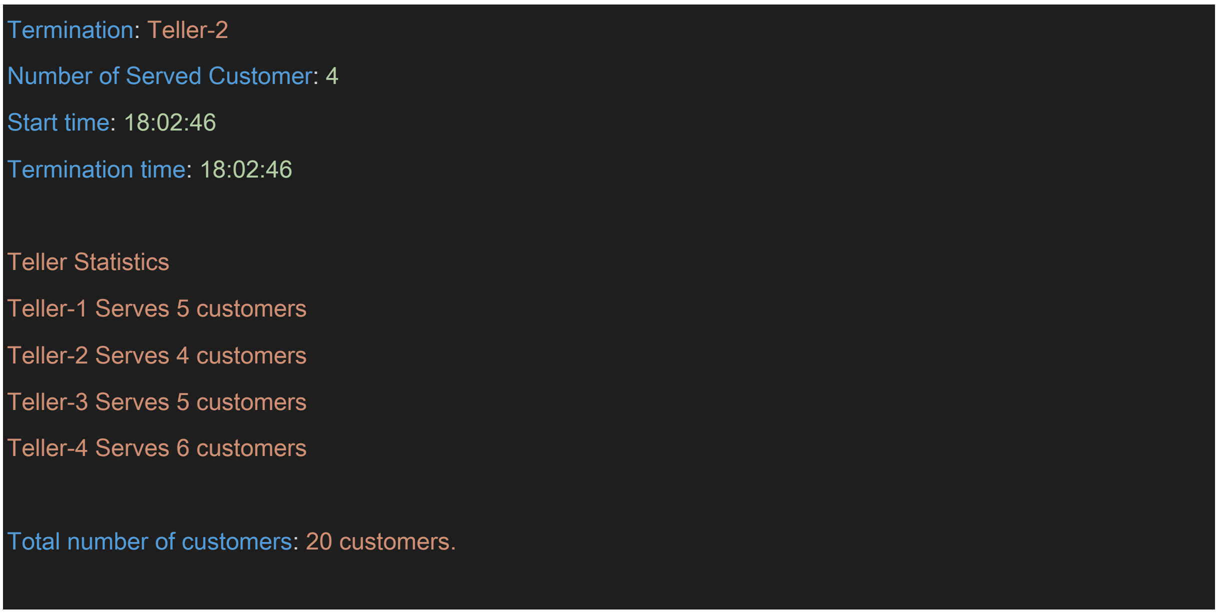
```
Teller: 1
Customer: 14
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 4
Customer: 13
Arrival time: 18:02:46
Completion time: 18:02:46

------------------------------------------------------------------------
15: W
Arrival time: 18:02:46
------------------------------------------------------------------------

Teller: 3
Customer: 15
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 1
Customer: 14
Arrival time: 18:02:46
Completion time: 18:02:46

------------------------------------------------------------------------
16: I
Arrival time: 18:02:46
------------------------------------------------------------------------

Teller: 3
Customer: 15
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 4
Customer: 16
Arrival time: 18:02:46
Response time: 18:02:46
```



```
---------------------------------------------------------------------------------------
17: l
Arrival time: 18:02:46
---------------------------------------------------------------------------------------


Teller: 4
Customer: 16
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 3
Customer: 17
Arrival time: 18:02:46
Response time: 18:02:46


---------------------------------------------------------------------------------------
18: l
Arrival time: 18:02:46
---------------------------------------------------------------------------------------


Teller: 4
Customer: 18
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 3
Customer: 17
Arrival time: 18:02:46
Completion time: 18:02:46


---------------------------------------------------------------------------------------
19: l
Arrival time: 18:02:46
---------------------------------------------------------------------------------------


Teller: 4
Customer: 18
Arrival time: 18:02:46
Completion time: 18:02:46
```



```
Teller: 2
Customer: 19
Arrival time: 18:02:46
Response time: 18:02:46

Teller: 1
Customer: 20
Arrival time: 18:02:46
Response time: 18:02:46

----------------------------------------------------------------------
20: W
Arrival time: 18:02:46
----------------------------------------------------------------------

Teller: 2
Customer: 19
Arrival time: 18:02:46
Completion time: 18:02:46

Teller: 1
Customer: 20
Arrival time: 18:02:46
Completion time: 18:02:46

Termination: Teller-4
Number of Served Customer: 6
Start time: 18:02:46
Termination time: 18:02:46

Termination: Teller-3
Number of Served Customer: 5
Start time: 18:02:46
Termination time: 18:02:46

Termination: Teller-1
Number of Served Customer: 5
Start time: 18:02:46
Termination time: 18:02:46
```



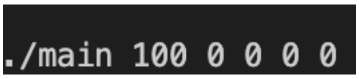
```
Termination: Teller-2
Number of Served Customer: 4
Start time: 18:02:46
Termination time: 18:02:46

Teller Statistics
Teller-1 Serves 5 customers
Teller-2 Serves 4 customers
Teller-3 Serves 5 customers
Teller-4 Serves 6 customers

Total number of customers: 20 customers.
```

For example 2, even when i set all the sleep times to 0 you can see that there are no race conditions or any problem in general as it works as in intended with synchronisation being satisfied.

Example 3:

For this example, I will test on 150 customers however will just show the last 5 customers to prevent having 10000 lines of output.

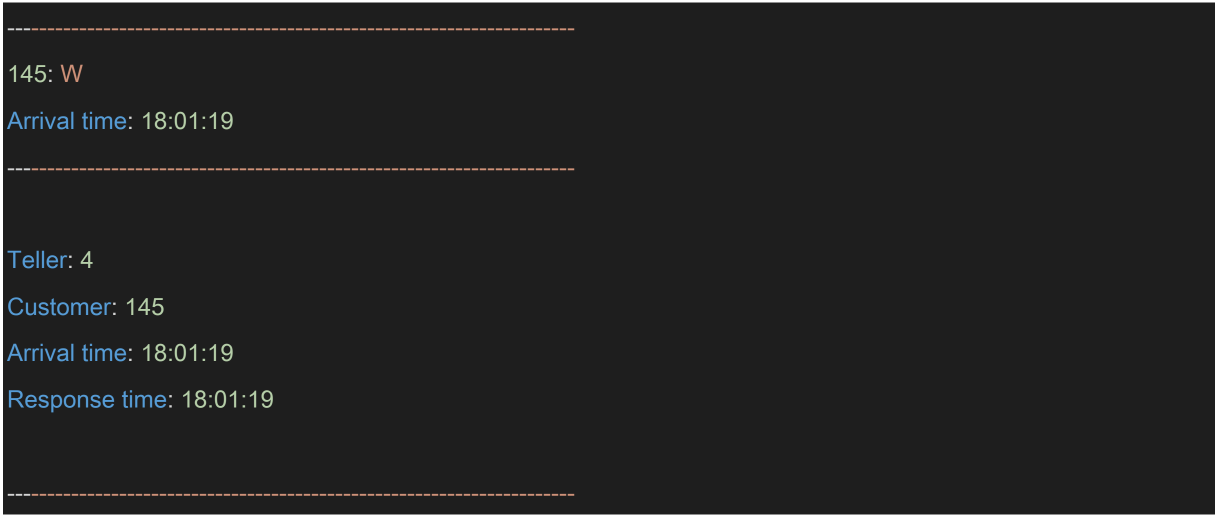
Input: ./main 100 0 0 0 0



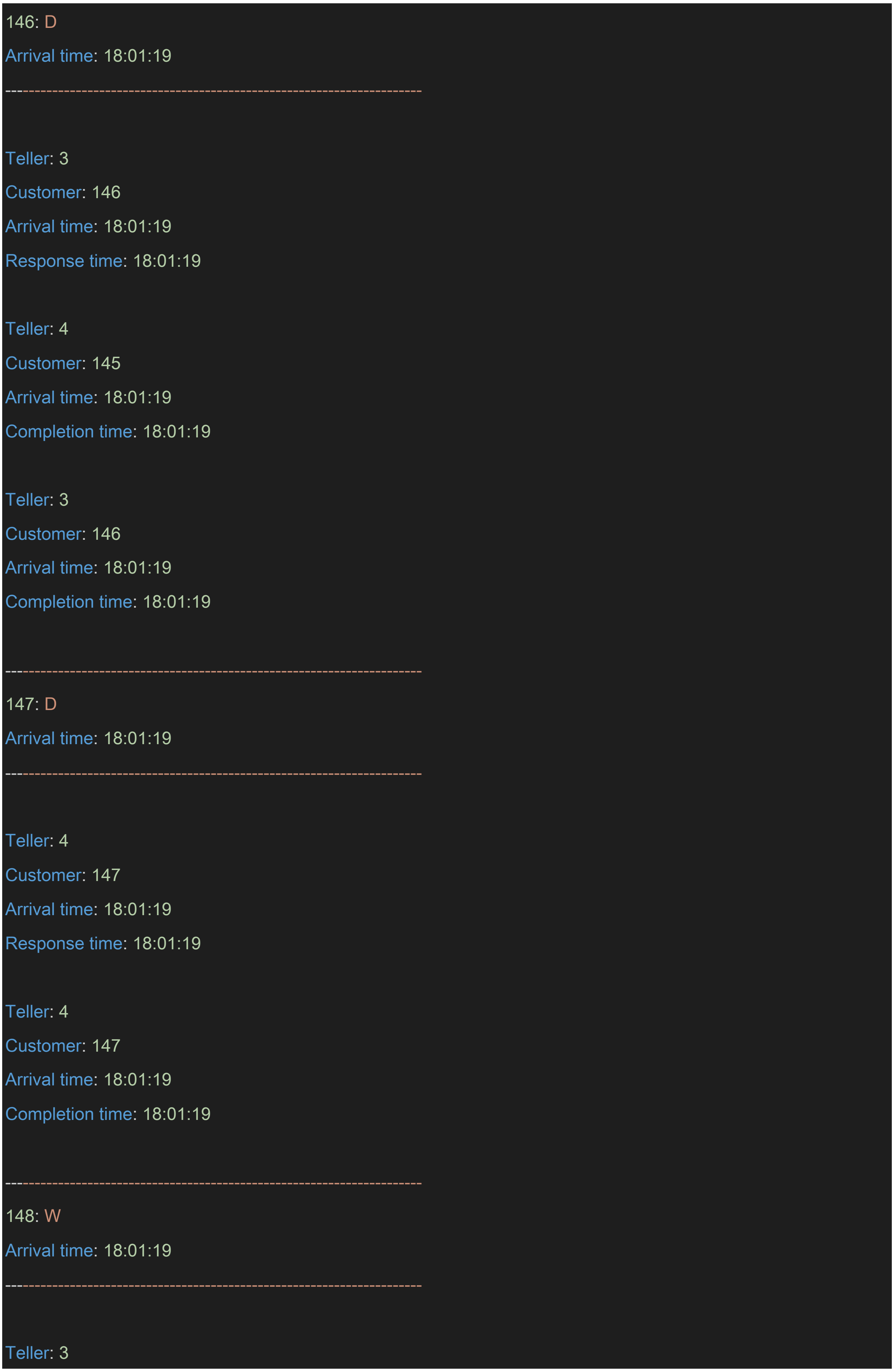
```
./main 100 0 0 0 0
```

Output:

r_log

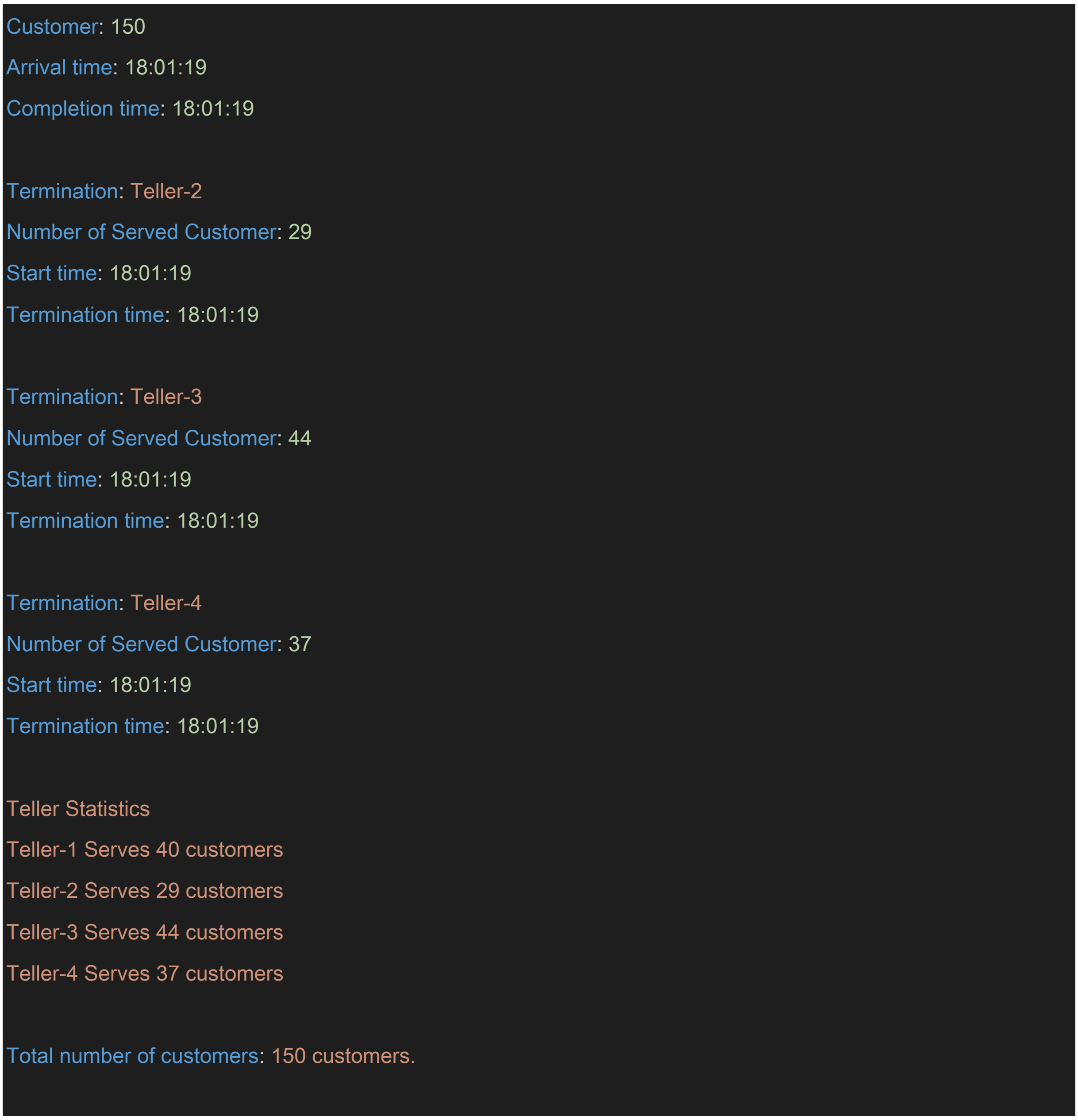


```
-----------------------------------------------------------------------
145: W
Arrival time: 18:01:19
-----------------------------------------------------------------------

Teller: 4
Customer: 145
Arrival time: 18:01:19
Response time: 18:01:19

-----------------------------------------------------------------------
```



```
146: D
Arrival time: 18:01:19
----------------------------------------------------------------------


Teller: 3
Customer: 146
Arrival time: 18:01:19
Response time: 18:01:19

Teller: 4
Customer: 145
Arrival time: 18:01:19
Completion time: 18:01:19

Teller: 3
Customer: 146
Arrival time: 18:01:19
Completion time: 18:01:19


----------------------------------------------------------------------
147: D
Arrival time: 18:01:19
----------------------------------------------------------------------


Teller: 4
Customer: 147
Arrival time: 18:01:19
Response time: 18:01:19

Teller: 4
Customer: 147
Arrival time: 18:01:19
Completion time: 18:01:19


----------------------------------------------------------------------
148: W
Arrival time: 18:01:19
----------------------------------------------------------------------


Teller: 3
```

```
Customer: 148
Arrival time: 18:01:19
Response time: 18:01:19

Teller: 3
Customer: 148
Arrival time: 18:01:19
Completion time: 18:01:19


-----------------------------------------------------------------------
149: D
Arrival time: 18:01:19
-----------------------------------------------------------------------


Teller: 1
Customer: 149
Arrival time: 18:01:19
Response time: 18:01:19

Teller: 1
Customer: 149
Arrival time: 18:01:19
Completion time: 18:01:19


-----------------------------------------------------------------------
150: W
Arrival time: 18:01:19
-----------------------------------------------------------------------


Teller: 4
Customer: 150
Arrival time: 18:01:19
Response time: 18:01:19

Termination: Teller-1
Number of Served Customer: 40
Start time: 18:01:19
Termination time: 18:01:19

Teller: 4
```



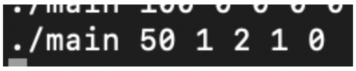
```
Customer: 150
Arrival time: 18:01:19
Completion time: 18:01:19

Termination: Teller-2
Number of Served Customer: 29
Start time: 18:01:19
Termination time: 18:01:19

Termination: Teller-3
Number of Served Customer: 44
Start time: 18:01:19
Termination time: 18:01:19

Termination: Teller-4
Number of Served Customer: 37
Start time: 18:01:19
Termination time: 18:01:19

Teller Statistics
Teller-1 Serves 40 customers
Teller-2 Serves 29 customers
Teller-3 Serves 44 customers
Teller-4 Serves 37 customers

Total number of customers: 150 customers.
```

For this example, you can see that the output is also correct even with 150 customers.

Example 4:

For this example, I will test on 150 customers however will just show the last 5 customers to prevent having 10000 lines of output.

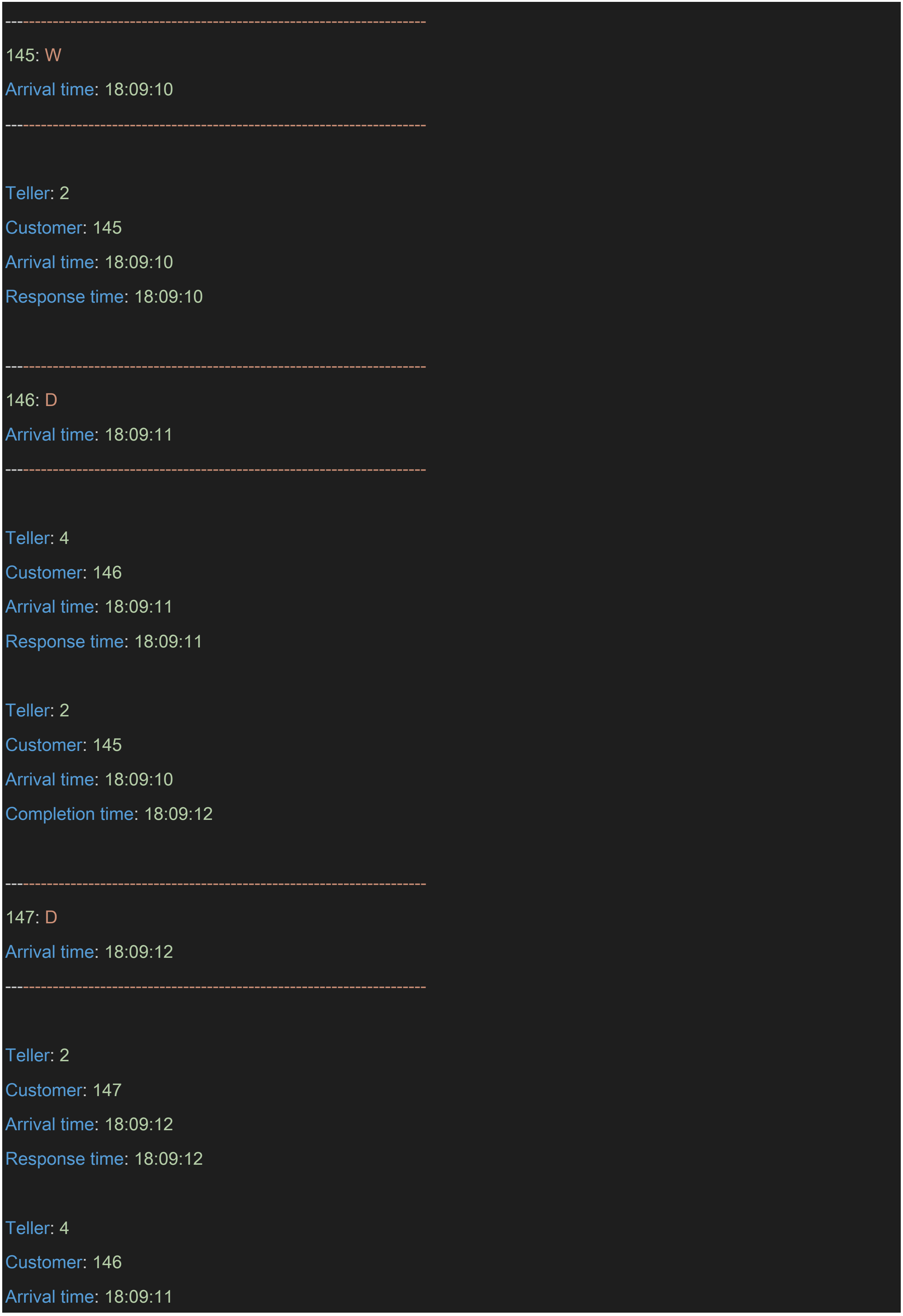
Input: ./main 50 1 2 1 0



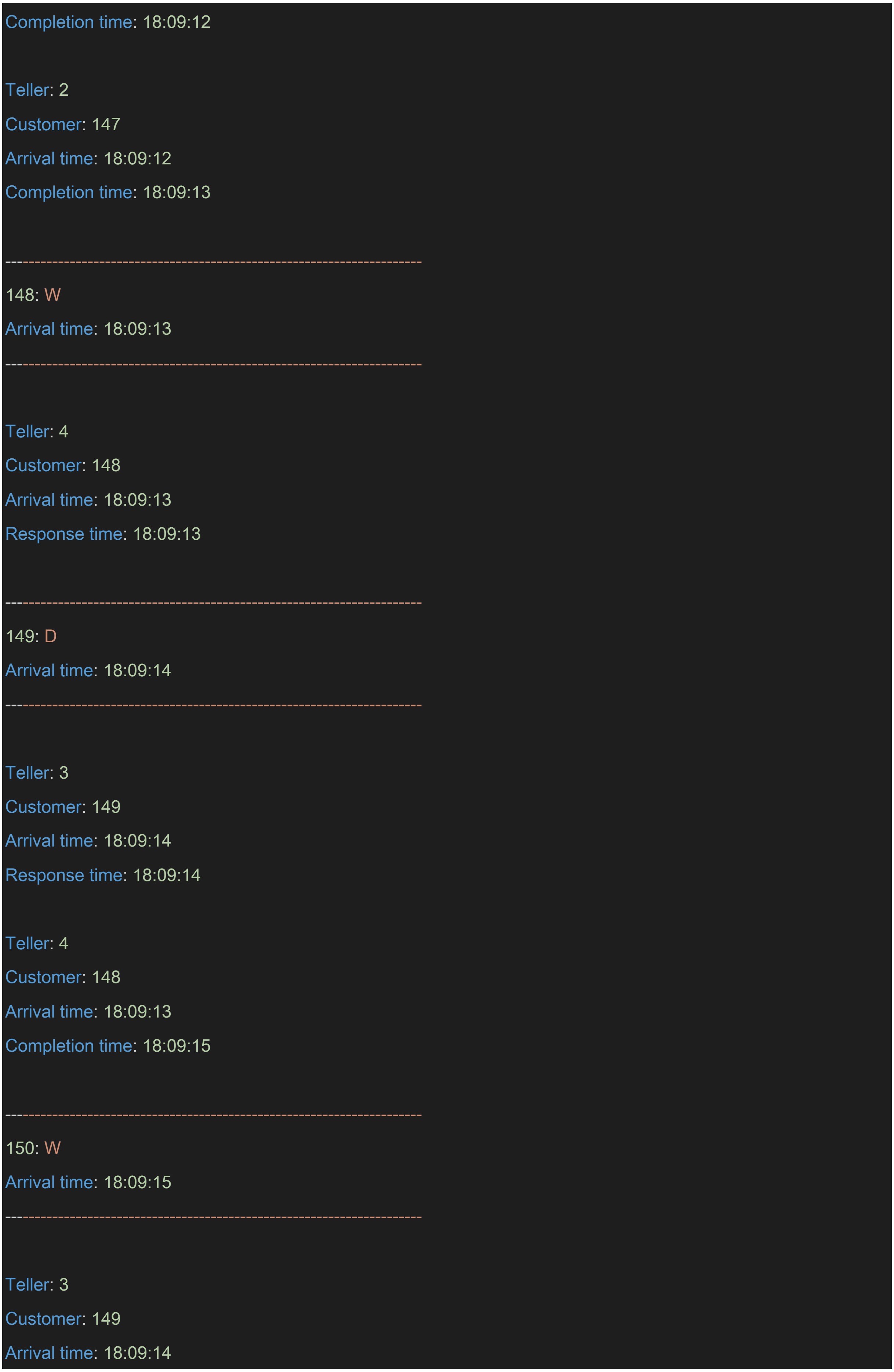
```
./main 100 0 0 0 0
./main 50 1 2 1 0
```

Output:

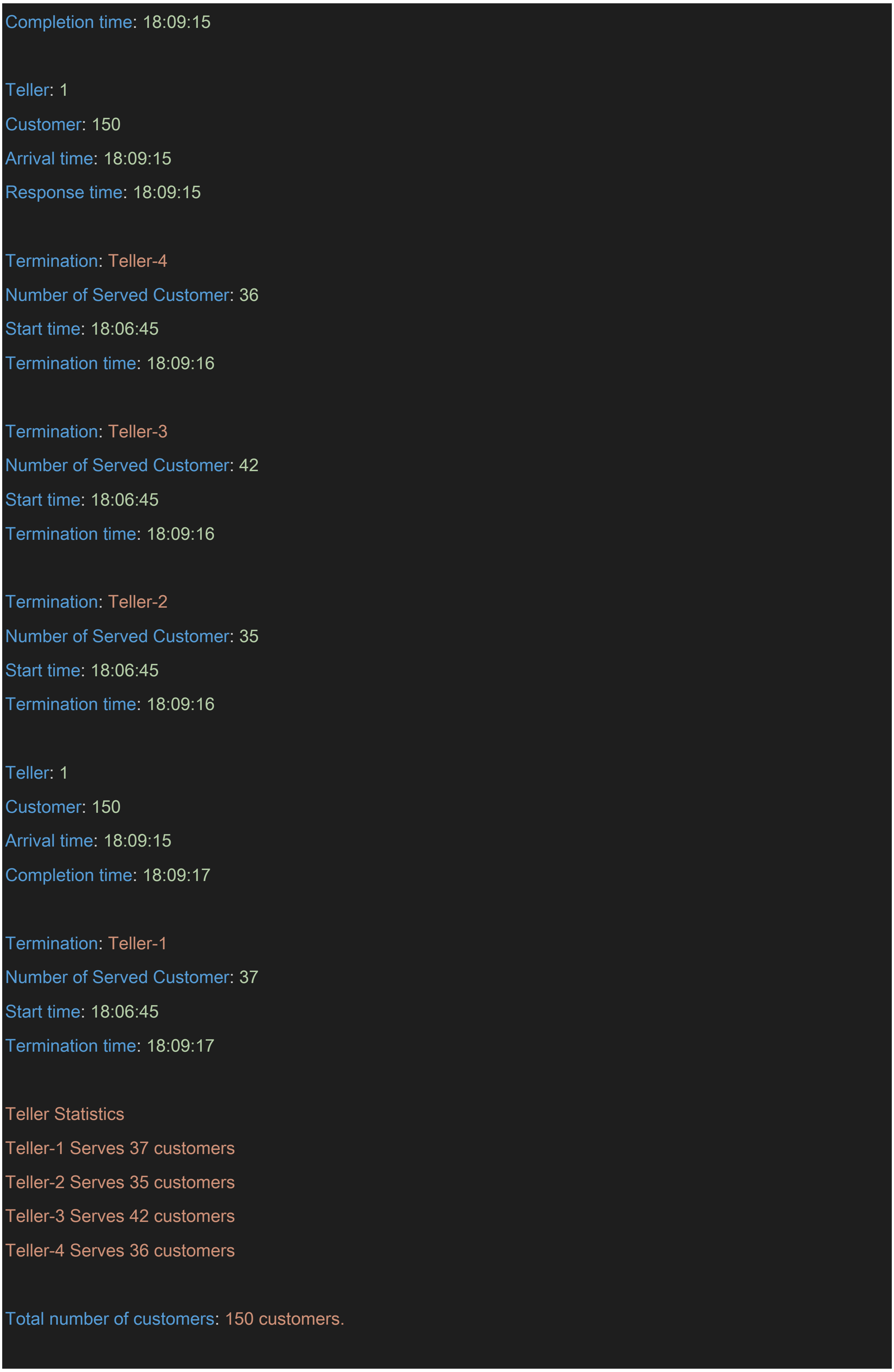
r_log



```
-----------------------------------------------------------------------
145: W
Arrival time: 18:09:10
-----------------------------------------------------------------------

Teller: 2
Customer: 145
Arrival time: 18:09:10
Response time: 18:09:10

-----------------------------------------------------------------------
146: D
Arrival time: 18:09:11
-----------------------------------------------------------------------

Teller: 4
Customer: 146
Arrival time: 18:09:11
Response time: 18:09:11

Teller: 2
Customer: 145
Arrival time: 18:09:10
Completion time: 18:09:12

-----------------------------------------------------------------------
147: D
Arrival time: 18:09:12
-----------------------------------------------------------------------

Teller: 2
Customer: 147
Arrival time: 18:09:12
Response time: 18:09:12

Teller: 4
Customer: 146
Arrival time: 18:09:11
```



```
Completion time: 18:09:12

Teller: 2
Customer: 147
Arrival time: 18:09:12
Completion time: 18:09:13

------------------------------------------------------------------------
148: W
Arrival time: 18:09:13
------------------------------------------------------------------------

Teller: 4
Customer: 148
Arrival time: 18:09:13
Response time: 18:09:13

------------------------------------------------------------------------
149: D
Arrival time: 18:09:14
------------------------------------------------------------------------

Teller: 3
Customer: 149
Arrival time: 18:09:14
Response time: 18:09:14

Teller: 4
Customer: 148
Arrival time: 18:09:13
Completion time: 18:09:15

------------------------------------------------------------------------
150: W
Arrival time: 18:09:15
------------------------------------------------------------------------

Teller: 3
Customer: 149
Arrival time: 18:09:14
```



```
Completion time: 18:09:15

Teller: 1
Customer: 150
Arrival time: 18:09:15
Response time: 18:09:15

Termination: Teller-4
Number of Served Customer: 36
Start time: 18:06:45
Termination time: 18:09:16

Termination: Teller-3
Number of Served Customer: 42
Start time: 18:06:45
Termination time: 18:09:16

Termination: Teller-2
Number of Served Customer: 35
Start time: 18:06:45
Termination time: 18:09:16

Teller: 1
Customer: 150
Arrival time: 18:09:15
Completion time: 18:09:17

Termination: Teller-1
Number of Served Customer: 37
Start time: 18:06:45
Termination time: 18:09:17

Teller Statistics
Teller-1 Serves 37 customers
Teller-2 Serves 35 customers
Teller-3 Serves 42 customers
Teller-4 Serves 36 customers

Total number of customers: 150 customers.
```

For example 4, you can see that it works as intended and synchronisation is satisfied. It also displays that all the assignment requirements are met.

Example 5:

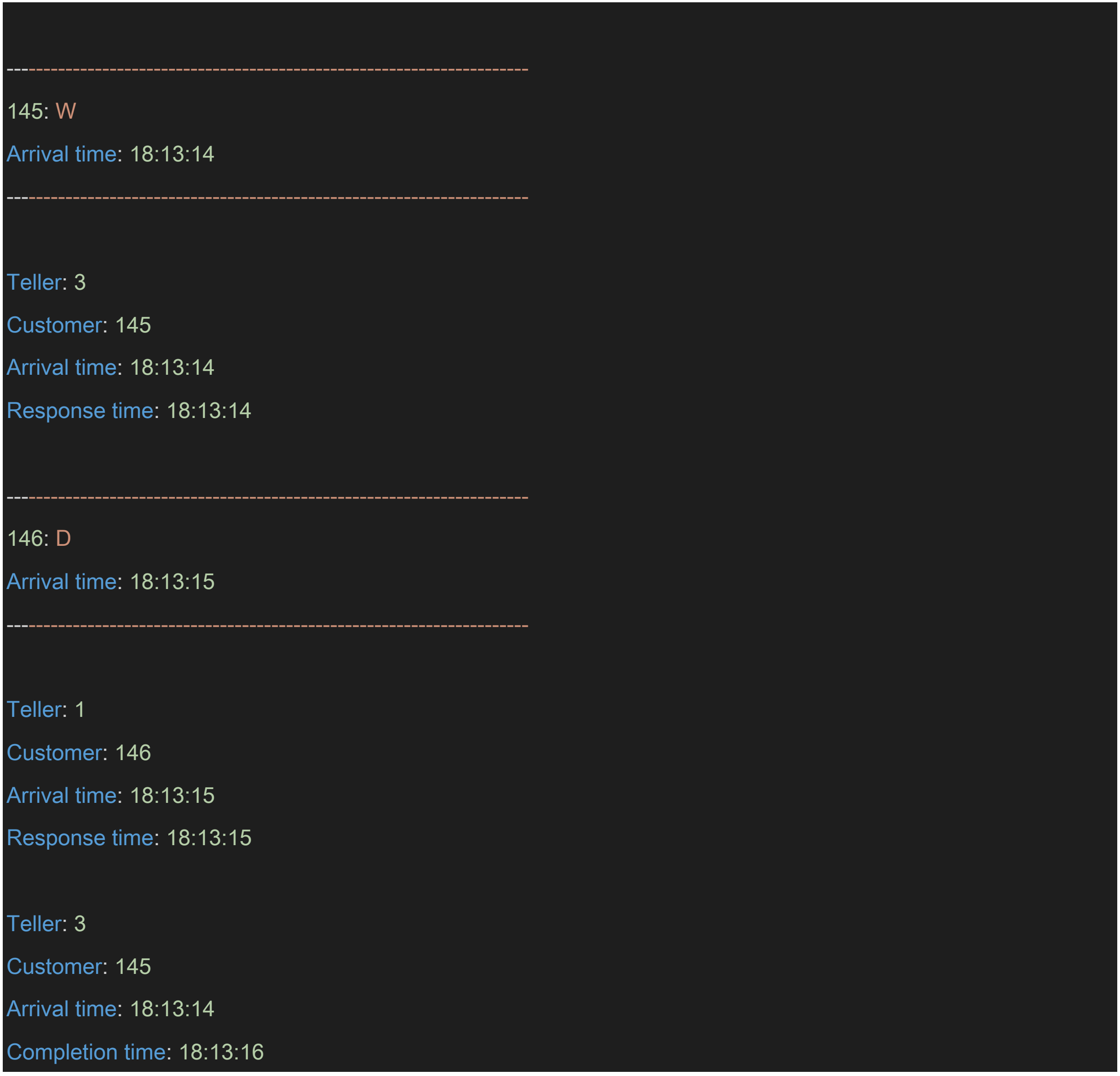
For this example, I will test on 150 customers however will just show the last 5 customers to prevent having 10000 lines of output.

Input: ./main 3 1 2 1 1

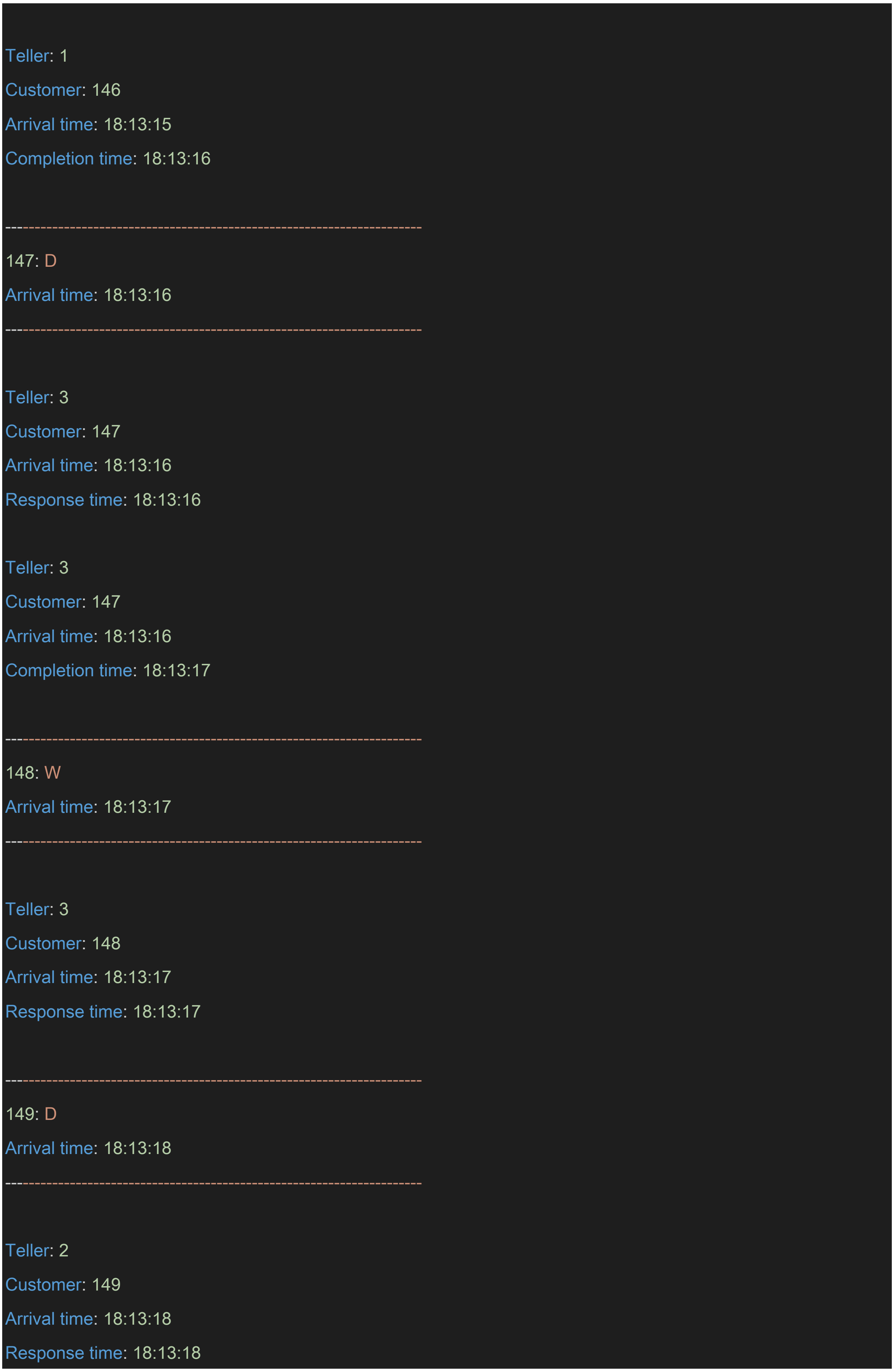
```
./main 3 1 2 1 1
```

Output:

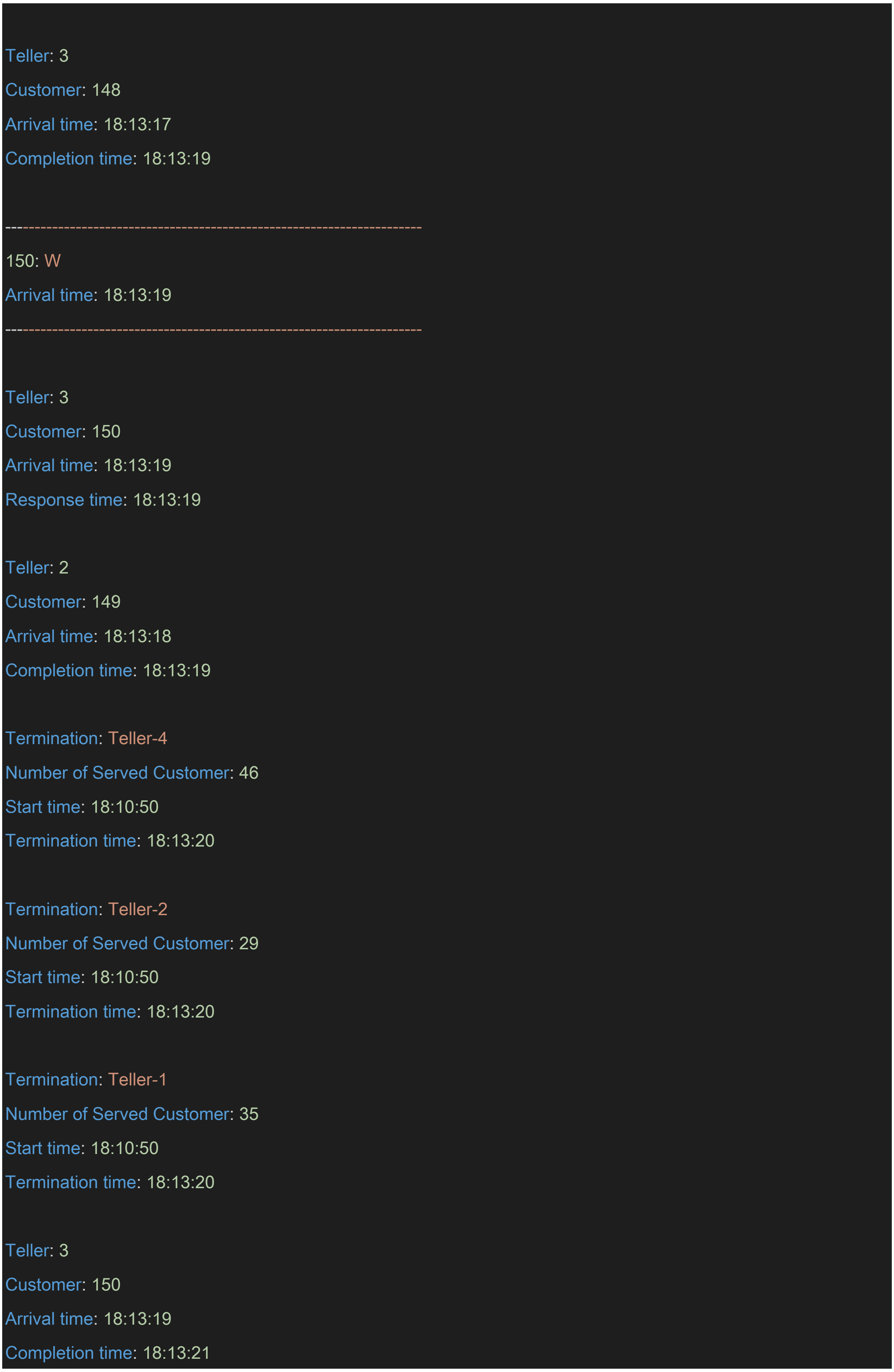
r_log



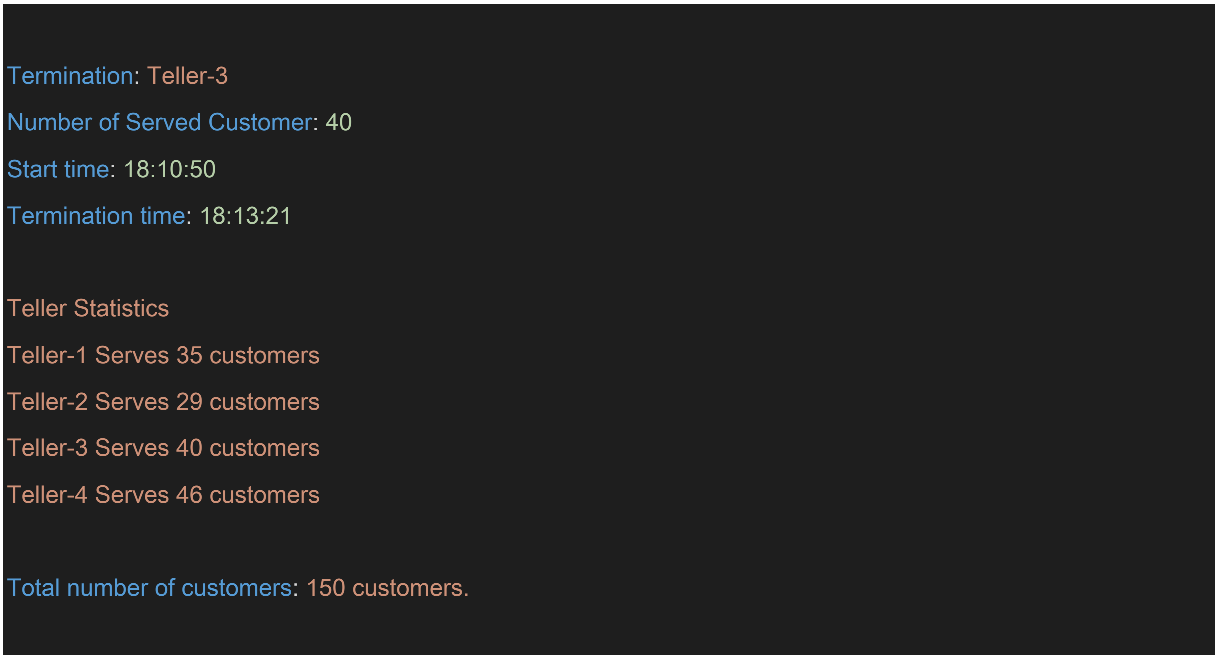
```
-----------------------------------------------------------------------
145: W
Arrival time: 18:13:14
-----------------------------------------------------------------------


Teller: 3
Customer: 145
Arrival time: 18:13:14
Response time: 18:13:14


-----------------------------------------------------------------------
146: D
Arrival time: 18:13:15
-----------------------------------------------------------------------


Teller: 1
Customer: 146
Arrival time: 18:13:15
Response time: 18:13:15


Teller: 3
Customer: 145
Arrival time: 18:13:14
Completion time: 18:13:16
```



```
Teller: 1
Customer: 146
Arrival time: 18:13:15
Completion time: 18:13:16


--------------------------------------------------------------------------
147: D
Arrival time: 18:13:16
--------------------------------------------------------------------------


Teller: 3
Customer: 147
Arrival time: 18:13:16
Response time: 18:13:16

Teller: 3
Customer: 147
Arrival time: 18:13:16
Completion time: 18:13:17


--------------------------------------------------------------------------
148: W
Arrival time: 18:13:17
--------------------------------------------------------------------------


Teller: 3
Customer: 148
Arrival time: 18:13:17
Response time: 18:13:17


--------------------------------------------------------------------------
149: D
Arrival time: 18:13:18
--------------------------------------------------------------------------


Teller: 2
Customer: 149
Arrival time: 18:13:18
Response time: 18:13:18
```



```
Teller: 3
Customer: 148
Arrival time: 18:13:17
Completion time: 18:13:19

--------------------------------------------------------------------
150: W
Arrival time: 18:13:19
--------------------------------------------------------------------

Teller: 3
Customer: 150
Arrival time: 18:13:19
Response time: 18:13:19

Teller: 2
Customer: 149
Arrival time: 18:13:18
Completion time: 18:13:19

Termination: Teller-4
Number of Served Customer: 46
Start time: 18:10:50
Termination time: 18:13:20

Termination: Teller-2
Number of Served Customer: 29
Start time: 18:10:50
Termination time: 18:13:20

Termination: Teller-1
Number of Served Customer: 35
Start time: 18:10:50
Termination time: 18:13:20

Teller: 3
Customer: 150
Arrival time: 18:13:19
Completion time: 18:13:21
```



```
Termination: Teller-3
Number of Served Customer: 40
Start time: 18:10:50
Termination time: 18:13:21

Teller Statistics
Teller-1 Serves 35 customers
Teller-2 Serves 29 customers
Teller-3 Serves 40 customers
Teller-4 Serves 46 customers

Total number of customers: 150 customers.
```

As you can see example 5 also works without any issues because there are no synchronisation issues. It also displays that all the assignment requirements are met.