Damian Franco

Meiling Traeger

CS 481-003

Programming Assignment #6

For PA06, we are asked to design the waiting area of a Universal Studios ride that involves dinosaurs chasing the riders and putting them in danger. The implementation of the simulation here was to be done using C and multiple threads. We are given many rules and statistics behind the simulation that we had to follow.

First, we programmed taking in user input. The two main inputs that we are taking from the user are the *N* and *M* variables where the *N* relates to the number of Ford Explorer cars and *M* is the number of people that can fit in each car. Next, the creation of the threads is handled. The number of threads that are created is determined by the number of Ford Explorer cars there are, which is also the input *N* from the user. A thread must also be created for the handling of the waiting line logic. For us, a timer thread was also needed to have the simulation constantly running at a fixed rate. Time and timing was the most difficult to simulate. We remembered Dr. Sun talked about timers and system timing in lecture, so we revisited those lectures but most of those did not translate into what we were trying to do on this assignment. The solution that we came up with came from a combination of multiple code examples on the various websites that help out students like us. So, the amount of threads in our simulation is *N*+2, one thread for time management, and the other for the waiting line management.

The next step was to synchronize the threads without causing any deadlocks and keeping all threads within the critical section. A mutex lock and a pthread condition variable is exactly what we needed to achieve synchronization. The mutex lock will allow each thread to access the global variables without causing data discrepancies. A pthread condition variable will allow all the car and line threads to wait for a signal from the timer thread that will call the threads into action. Pseudocode for the thread coordination within our program is shown on the next page.

Thread Coordination Pseudocode:

function timerThread()

Do timer handling and check for every tick/one second

if (1 second has gone by)

Lock the mutex

Condition variable signals all the threads to wake up and run

Unlock the mutex

end if

end function

Function carThread()

Wait for the condition variable to signal

if (condition variable is signaled)

Lock the mutex

Do car thread handling here

Unlock the mutex

end if

Sleep thread to allow other threads to enter the shared resources safely

Call function recursively to wait for the next time to run

end function

Function lineThread()

Wait for the condition variable to signal

if (condition variable is signaled)

Lock the mutex

Do line thread handling

Unlock the mutex

end if

Sleep thread to allow other threads to enter the shared resources safely

Call function recursively to wait for the next time to run

end function

With the process shown above, the threads are synchronized and running, but this did have its difficulties. Many times when we were testing, sometimes data would get really buggy and would print out many numbers that were completely out of our range (negative values). This was partly due to the threads all being called at the same time and many of the threads trying to reach and grab resources at the same time. We were lost, but after calling the sleep function to sleep the thread and let the other threads reach into the global variables helped fix this problem mostly. Our program still has a problem every once and a while where there will be a vastly wrong answer, specifically in the wait-line. Thread synchronization was a difficult task that we achieved by utilizing many pthread function calls and some intuition.

After achieving thread synchronization, we handled the functionality of the line and car threads. This was a fairly straightforward task, which involved changing and checking many global variables. The only thing that we could not implement into the program itself was the logic behind the tracking of the average waiting time. Each time we tried to implement it, we had output that was something that was completely out of range. We ended up just getting the average waiting time with some functions within Excel that allowed us to get that after the fact. The other problem we had was tracking the time representation HH:MM:SS in the string, which prints out some bad outputs. All the other functionality of the program is implemented and, for the most part, is working.

Overall, the implementation of the project led to us having some long nights and really stressful times. Thread synchronization is difficult and took us a very long time to implement, and our implementation is not the best or most efficient. Sometimes our program works, and sometimes it does not, and that is because of our implementation of the thread synchronization. We had many runs of the program that would work and make it all the way through the 10 minutes of simulation, but sometimes it would not and would just freeze all the threads. This was and still is a problem that we tried to solve, but we did not have enough time to fully revitalize. The state of the program is something that we are proud of because of the time and effort we put in it, but there are still some problems which did not stop us from fully implementing what was asked.

Below, you can find various screenshots of the programs output:

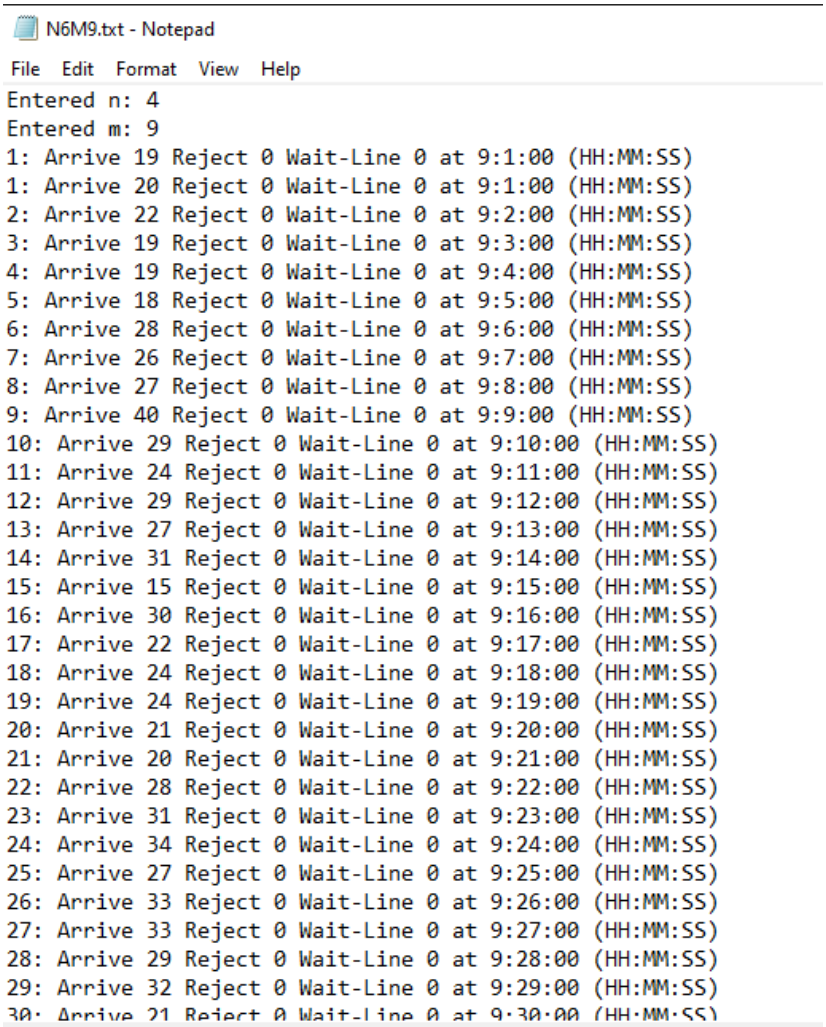
Screenshot of the thread synchronization working with 2 car threads and 1 line thread:

```
dfranco@b146-12:/nfs/student/student/d/dfranco/CS/CS481/Assignment6$ gcc -pthread PA06.c -lm
dfranco@b146-12:/nfs/student/student/d/dfranco/CS/CS481/Assignment6$ ./a.out -N 2 -M 7
Entered n: 2
Entered m: 7
Clock Thread 20362 is running
Line Thread 20363 is running
Car Thread 20365 is running
Car Thread 20364 is running
Line Thread 20363 is running
Car Thread 20365 is running
Car Thread 20364 is running
```

The output of the program when it is ran:

```
dfranco@b146-12:/nfs/student/student/d/dfranco/CS/CS481/Assignment6$ gcc -pthread PA06.c -lm
dfranco@b146-12:/nfs/student/student/d/dfranco/CS/CS481/Assignment6$ ./a.out -N 2 -M 7
Entered n: 2
Entered m: 7
Clock Thread 19873 printed
1: Arrive 21 Reject 0 Wait-Line 21 at 9:1:00 (HH:MM:SS)
2: Arrive 19 Reject 0 Wait-Line 26 at 9:2:00 (HH:MM:SS)
3: Arrive 21 Reject 0 Wait-Line 47 at 9:3:00 (HH:MM:SS)
4: Arrive 27 Reject 0 Wait-Line 74 at 9:4:00 (HH:MM:SS)
5: Arrive 40 Reject 0 Wait-Line 100 at 9:5:00 (HH:MM:SS)
6: Arrive 21 Reject 0 Wait-Line 121 at 9:6:00 (HH:MM:SS)
7: Arrive 27 Reject 0 Wait-Line 148 at 9:7:00 (HH:MM:SS)
8: Arrive 27 Reject 0 Wait-Line 161 at 9:8:00 (HH:MM:SS)
9: Arrive 29 Reject 0 Wait-Line 190 at 9:9:00 (HH:MM:SS)
10: Arrive 19 Reject 0 Wait-Line 209 at 9:10:00 (HH:MM:SS)
11: Arrive 20 Reject 0 Wait-Line 215 at 9:11:00 (HH:MM:SS)
12: Arrive 24 Reject 0 Wait-Line 239 at 9:12:00 (HH:MM:SS)
13: Arrive 24 Reject 0 Wait-Line 263 at 9:13:00 (HH:MM:SS)
14: Arrive 34 Reject 0 Wait-Line 283 at 9:14:00 (HH:MM:SS)
15: Arrive 38 Reject 0 Wait-Line 321 at 9:15:00 (HH:MM:SS)
16: Arrive 27 Reject 0 Wait-Line 341 at 9:16:00 (HH:MM:SS)
17: Arrive 28 Reject 0 Wait-Line 362 at 9:17:00 (HH:MM:SS)
18: Arrive 25 Reject 0 Wait-Line 380 at 9:18:00 (HH:MM:SS)
19: Arrive 31 Reject 0 Wait-Line 411 at 9:19:00 (HH:MM:SS)
20: Arrive 31 Reject 0 Wait-Line 435 at 9:20:00 (HH:MM:SS)
21: Arrive 26 Reject 0 Wait-Line 447 at 9:21:00 (HH:MM:SS)
22: Arrive 22 Reject 0 Wait-Line 469 at 9:22:00 (HH:MM:SS)
23: Arrive 26 Reject 0 Wait-Line 488 at 9:23:00 (HH:MM:SS)
24: Arrive 27 Reject 0 Wait-Line 508 at 9:24:00 (HH:MM:SS)
```

The text file after the program has completed:



N6M9.txt - Notepad

File Edit Format View Help

```
Entered n: 4
Entered m: 9
1: Arrive 19 Reject 0 Wait-Line 0 at 9:1:00 (HH:MM:SS)
1: Arrive 20 Reject 0 Wait-Line 0 at 9:1:00 (HH:MM:SS)
2: Arrive 22 Reject 0 Wait-Line 0 at 9:2:00 (HH:MM:SS)
3: Arrive 19 Reject 0 Wait-Line 0 at 9:3:00 (HH:MM:SS)
4: Arrive 19 Reject 0 Wait-Line 0 at 9:4:00 (HH:MM:SS)
5: Arrive 18 Reject 0 Wait-Line 0 at 9:5:00 (HH:MM:SS)
6: Arrive 28 Reject 0 Wait-Line 0 at 9:6:00 (HH:MM:SS)
7: Arrive 26 Reject 0 Wait-Line 0 at 9:7:00 (HH:MM:SS)
8: Arrive 27 Reject 0 Wait-Line 0 at 9:8:00 (HH:MM:SS)
9: Arrive 40 Reject 0 Wait-Line 0 at 9:9:00 (HH:MM:SS)
10: Arrive 29 Reject 0 Wait-Line 0 at 9:10:00 (HH:MM:SS)
11: Arrive 24 Reject 0 Wait-Line 0 at 9:11:00 (HH:MM:SS)
12: Arrive 29 Reject 0 Wait-Line 0 at 9:12:00 (HH:MM:SS)
13: Arrive 27 Reject 0 Wait-Line 0 at 9:13:00 (HH:MM:SS)
14: Arrive 31 Reject 0 Wait-Line 0 at 9:14:00 (HH:MM:SS)
15: Arrive 15 Reject 0 Wait-Line 0 at 9:15:00 (HH:MM:SS)
16: Arrive 30 Reject 0 Wait-Line 0 at 9:16:00 (HH:MM:SS)
17: Arrive 22 Reject 0 Wait-Line 0 at 9:17:00 (HH:MM:SS)
18: Arrive 24 Reject 0 Wait-Line 0 at 9:18:00 (HH:MM:SS)
19: Arrive 24 Reject 0 Wait-Line 0 at 9:19:00 (HH:MM:SS)
20: Arrive 21 Reject 0 Wait-Line 0 at 9:20:00 (HH:MM:SS)
21: Arrive 20 Reject 0 Wait-Line 0 at 9:21:00 (HH:MM:SS)
22: Arrive 28 Reject 0 Wait-Line 0 at 9:22:00 (HH:MM:SS)
23: Arrive 31 Reject 0 Wait-Line 0 at 9:23:00 (HH:MM:SS)
24: Arrive 34 Reject 0 Wait-Line 0 at 9:24:00 (HH:MM:SS)
25: Arrive 27 Reject 0 Wait-Line 0 at 9:25:00 (HH:MM:SS)
26: Arrive 33 Reject 0 Wait-Line 0 at 9:26:00 (HH:MM:SS)
27: Arrive 33 Reject 0 Wait-Line 0 at 9:27:00 (HH:MM:SS)
28: Arrive 29 Reject 0 Wait-Line 0 at 9:28:00 (HH:MM:SS)
29: Arrive 32 Reject 0 Wait-Line 0 at 9:29:00 (HH:MM:SS)
30: Arrive 21 Reject 0 Wait-Line 0 at 9:30:00 (HH:MM:SS)
```

CSV file of the output in Excel:

	People Arrival (N=2, M=7)		Wait-List (N = 2, M =7)	Wait-List (N = 4, M =7)	Wait-List (N = 6, M =7)		Rejected (N=2, M=7)	Rejected (N=4, M=7)	Rejected (N=6, M=7)
1	23	1	9	11	0	1	0	0	0
2	29	2	24	7	0	2	0	0	0
3	21	3	45	0	0	3	0	0	0
4	34	4	65	17	0	4	0	0	0
5	32	5	83	6	0	5	0	0	0
6	20	6	89	5	0	6	0	0	0
7	22	7	97	8	0	7	0	0	0
8	22	8	105	9	0	8	0	0	0
9	28	9	119	7	0	9	0	0	0
10	19	10	124	21	0	10	0	0	0
11	23	11	133	28	2	11	0	0	0
12	25	12	144	12	0	12	0	0	0
13	21	13	151	6	4	13	0	0	0
14	28	14	165	3	3	14	0	0	0
15	29	15	180	17	7	15	0	0	0
16	26	16	185	3	3	16	0	0	0
17	24	17	202	11	5	17	0	0	0
18	22	18	203	29	8	18	0	0	0
19	28	19	224	10	9	19	0	0	0
20	17	20	227	15	1	20	0	0	0
21	30	21	243	32	7	21	0	0	0
22	32	22	261	41	4	22	0	0	0
23	30	23	277	47	2	23	0	0	0
24	23	24	286	34	4	24	0	0	0
25	24	25	296	42	9	25	0	0	0
26	13	26	295	32	8	26	0	0	0
27	19	27	300	53	8	27	0	0	0
28	14	28	300	51	5	28	0	0	0
29	26	29	312	65	5	29	0	0	0
30	27	30	318	50	4	30	0	0	0

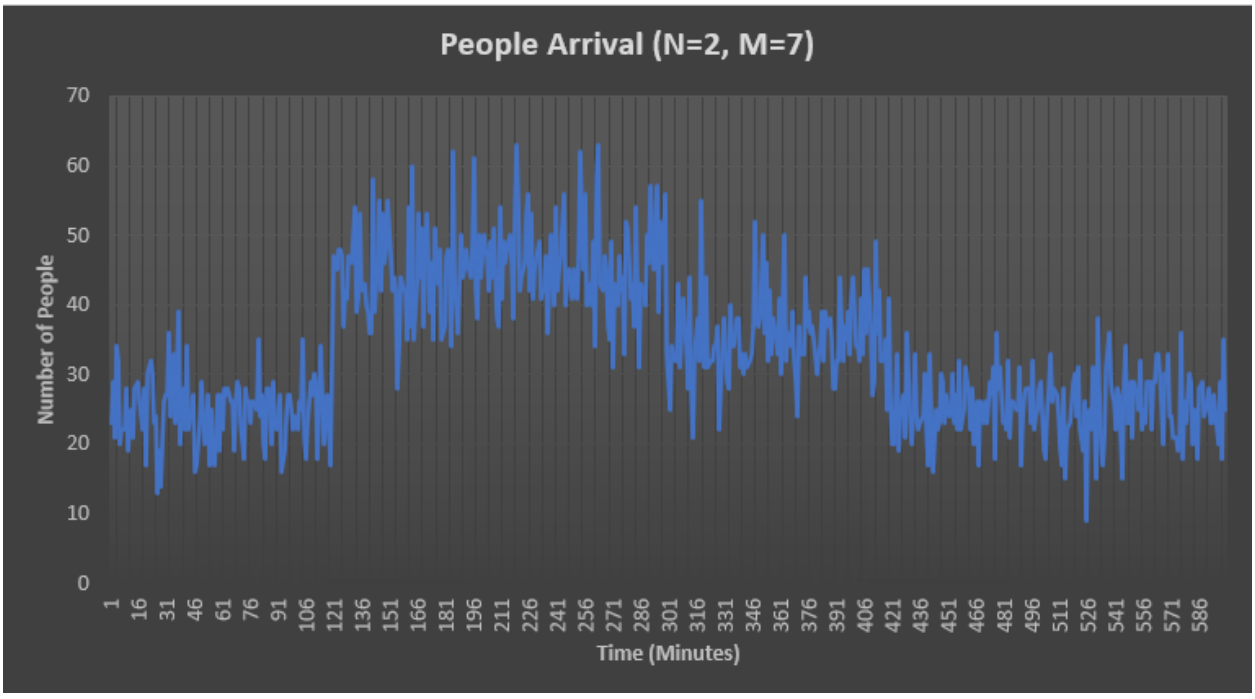
Looking at the data, we can notice that the more cars and seats in the cars we have, the better the turn-out is and the less people we have to turn away. The rejection ratio is very bad whenever we only have two cars being run and improves when we have more cars. We noticed that the number of cars that our ride has is correlated to total wait time and rejection ratio. The time also has a huge impact on the current state of our line management and ride. Around 120-400 seconds of time, most people show up and the wait-lines get longer. Overall, the data is very self-explanatory and we are glad that our program produced some decent data.

Data:

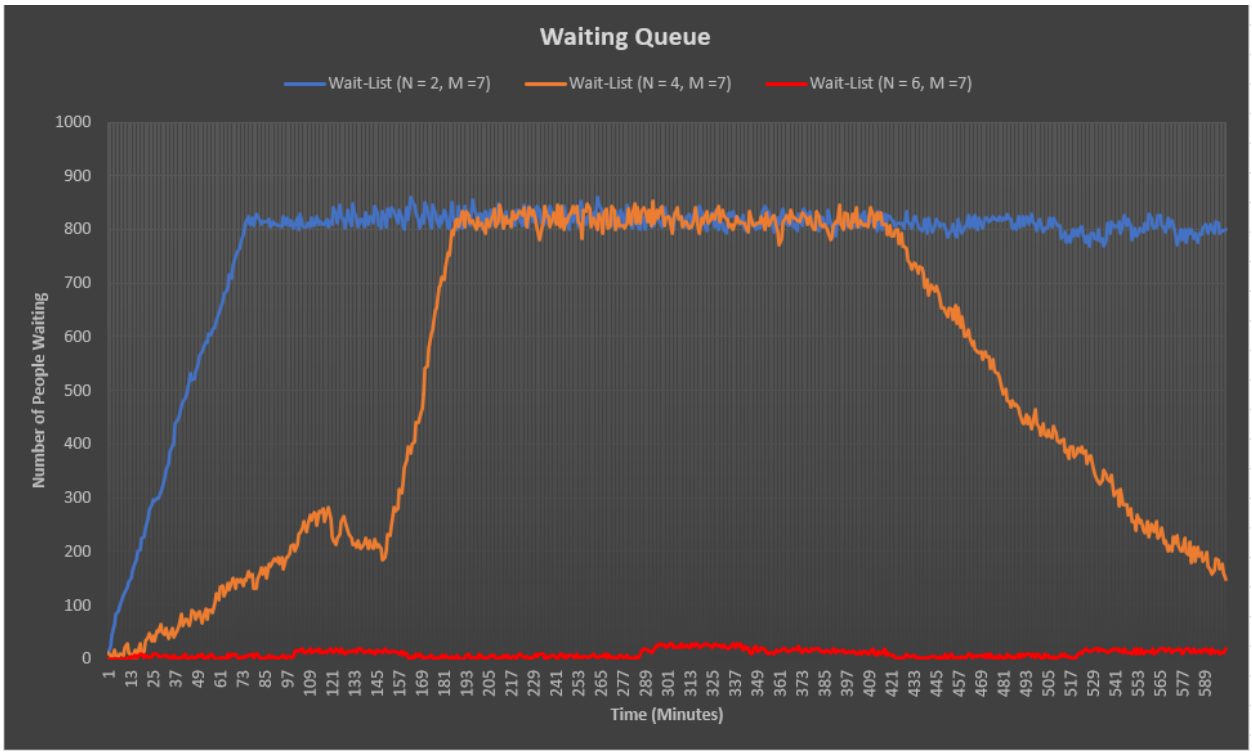
Table for different simulation parameters (Average wait time in the N=4, M=7 column should be 46, not 146):

(N,M)	Total Arrival	Total GoAway	Rejection Ratio	Average Wait Time (mins)	Max Waiting Line
N = 2, M = 7	19904	17632	0.88	176	859 at 164s
N = 2, M = 9	20126	17287	0.86	153	853 at 217s
N = 4, M = 7	19823	9553	0.48	146	849 at 258s
N = 4, M = 9	19977	0	0	0	499 at 379s
N = 6, M = 7	19713	0	0	0	257 at 310s
N = 6, M = 9	19888	0	0	0	29 at 307s

Graph of the people arriving at the given times:



Graph of the waiting queues for different simulations:



Graph of the people being rejected for different simulations: