Comp304 Project 2

Kemal Bora Bayraktar (75618) – Eray Sözer (69335)

Installation

Use the provided Makefile. Type **make** in the project directory at the command line. Provided Makefile compiles and runs the program with the default program arguments. If you want to specify the program arguments, compile the program using **make gcc.** Then run the program like this:

./main -p 0.5 -t 100 -s 1 -n 30

Type **make clean** to clean the executable files.

Part 1

In Part 1, we created methods for producers (Landing, Launching and Assembly), and consumers (Pad A, Pad B and Control Tower). Then we called these methods on different threads and when they become completed, we joined these threads and freed the memory.

In producers we created while loop which checks if the simulation time has been reached. If it has been reached it exits the method, if not it first sleeps for 1*t and then using the possibility method (random generator) we checked if we should generate a job and if generated, we insert it into the corresponding queue (landing_queue, launching_queue, assembly_queue).

In the pad methods of the consumers, just like in producers we created a while loop which checks if the simulation time has been reached. If it has been reached it exits the method, if not it checks if there is any jobs in its queue (padA_queue, padB_queue) if no jobs it sleeps for 2 second, if there is it grabs the first job and sleeps for its job time (which simulates doing the job). It accesses the job time using the job struct.

Finally, in the control tower, like the rest of the methods there is a while loop which checks if the simulation time has been reached. If it has been reached it exits the method, if not it first looks landing queue. If there are jobs in it, we check the waiting times on Pad A and Pad B and insert the job to the one with the shortest waiting time. Then we check launch queue. If it is not empty and there are no jobs in Pad A

(since landing is prioritized) we insert the launch job the Pad A. The same thing is done for assembly but for Pad B.

Also, in the main method we created a launch job and put it into launch queue since the first job is required to be launching.

```
At 0 sec landing    :
At 0 sec launch     : 1
At 0 sec assembly   :
At 0 sec emergency  :
At 0 sec padA       : 1(D)
At 0 sec padB       :

At 1 sec landing    :
At 1 sec launch     : 1
At 1 sec assembly   :
At 1 sec emergency  :
At 1 sec padA       : 1(D)
At 1 sec padB       :

At 2 sec landing    : 2
At 2 sec launch     : 1
At 2 sec assembly   :
At 2 sec emergency  :
At 2 sec padA       : 1(D)
At 2 sec padB       : 2(L)

At 3 sec landing    : 2
At 3 sec launch     : 1
At 3 sec assembly   :
At 3 sec emergency  :
At 3 sec padA       : 1(D)
At 3 sec padB       : 2(L)

At 4 sec landing    : 3
At 4 sec launch     : 1
At 4 sec assembly   :
At 4 sec emergency  :
At 4 sec padA       : 1(D)
At 4 sec padB       : 3(L)

At 5 sec landing    : 3
At 5 sec launch     : 1
At 5 sec assembly   :
At 5 sec emergency  :
At 5 sec padA       : 1(D)
At 5 sec padB       : 3(L)

At 6 sec landing    : 4 3
At 6 sec launch     :
At 6 sec assembly   :
At 6 sec emergency  :
At 6 sec padA       : 4(L)
At 6 sec padB       : 3(L)
```

Part 2

In this part we are asked to solve the starvation that may be caused to the ground jobs (assembly and launch) in Part 1. For the solution we are asked to check the following conditions when enqueueing jobs to the pads: You should favor ground jobs when there are no landing jobs, and you should favor ground jobs when there are 3 or more launches or 3 or more assemblies waiting. In our implementation of Part 1 we were already checking the first condition, so we just added the second condition into our if statements. Our if statement of launch now checks if there are no jobs waiting in Pad A and there are jobs waiting in launch queue or if there are more than 2 launch jobs waiting on the queue. If the conditions are met the job is enqueued into Pad A queue. The same is done for assembly but for Pad B. Also, when checking whether to enqueue landing jobs to the pads we now also check for if there are 3 or more launches or 3 or more assemblies waiting. If there is, we don't enqueue landing jobs.

```
At 65 sec landing    : 38
At 65 sec launch     : 26 29 37
At 65 sec assembly   : 31
At 65 sec emergency  :
At 65 sec padA       : 26(D) 38(L)
At 65 sec padB       : 31(A)

At 66 sec landing    : 38
At 66 sec launch     : 26 29 37 39
At 66 sec assembly   : 31
At 66 sec emergency  :
At 66 sec padA       : 26(D) 38(L) 29(D)
At 66 sec padB       : 31(A)
```

Here is an example output showing how the implementation works. At t = 65 seconds we have 29 and 37 in the launch queue (26 is not in the launch queue because it was enqueued into the Pad A queue). At t = 66 seconds a launch job is generated and the number of launch jobs in the launch queue becomes 3 so the first job in the launch queue is dequeued from launch queue and popped into the Pad A queue working as intended.

In our implementation whenever we enqueue a job into the pads we dequeue that job from its initial job queue (landing_queue, launching_queue, assembly_queue). Therefore, it avoids starvation since after we enqueue the ground jobs to the pad queues when the above 3 limit condition is met, we directly dequeue it from its original queue and the number drops back down to 2 and landing jobs can again be enqueued before another ground job can be generated and won't pile up and experience starvation.

Part 3

In Part 3, we created a producer method and a thread for emergency jobs just like we did for landing, launching and assembly jobs in Part 1. This thread checks if it has been 40*t since it created an emergency job. If it hasn't been, it sleeps for 1*t. If it has been 40*t, it creates 2 new emergency jobs and pops them in the emergency queue. This emergency queue is checked in the control tower thread. If it is empty nothing is done, if not following conditions are checked:

- If there are no jobs being currently worked (the execution of the job has started, but it is not yet finished) at in the Pad A or Pad B queues the emergency job is put in the first place of Pad A or Pad B queue.
- If there are jobs being currently worked on in Pad A and Pad B queues it checks the remaining times for these jobs to be done and puts the emergency job in the second place of the queue with shortest remaining time (if there is an emergency job in the second place of the queue it also takes it in the account when calculating remaining time).
- Finally, in the pads thread when time comes to executing emergency job it sleeps for 1*t then dequeues it from the pad queue.

```
At 79 sec landing    : 42
At 79 sec launch     :
At 79 sec assembly   : 41
At 79 sec emergency  :
At 79 sec padA       : 42(L)
At 79 sec padB       : 41(A)

At 80 sec landing    : 42 45
At 80 sec launch     :
At 80 sec assembly   : 41
At 80 sec emergency  : 44 43
At 80 sec padA       : 42(L) 44(E) 43(E) 45(L)
At 80 sec padB       : 41(A)

At 81 sec landing    : 45
At 81 sec launch     :
At 81 sec assembly   : 41
At 81 sec emergency  : 44 43
At 81 sec padA       : 44(E) 43(E) 45(L)
At 81 sec padB       : 41(A)

At 82 sec landing    : 45 46
At 82 sec launch     :
At 82 sec assembly   : 41
At 82 sec emergency  : 44 43
At 82 sec padA       : 44(E) 43(E) 45(L) 46(L)
At 82 sec padB       : 41(A)

At 83 sec landing    : 45 46
At 83 sec launch     :
At 83 sec assembly   : 41
At 83 sec emergency  : 43
At 83 sec padA       : 43(E) 45(L) 46(L)
At 83 sec padB       : 41(A)

At 84 sec landing    : 45 46 47
At 84 sec launch     :
At 84 sec assembly   : 41
At 84 sec emergency  : 43
At 84 sec padA       : 43(E) 45(L) 46(L) 47(L)
At 84 sec padB       : 41(A)

At 85 sec landing    : 45 46 47
At 85 sec launch     :
At 85 sec assembly   : 41
At 85 sec emergency  :
At 85 sec padA       : 45(L) 46(L) 47(L)
At 85 sec padB       : 41(A)
```

As expected at 40*t (80 sec) it created 2 emergency jobs and placed both in Pad A. At t = 80 seconds there are jobs being currently worked on in both pads (Landing in A, Assembly in B) and since there is less remaining time in Pad A it ended up putting both jobs in Pad A.

Keeping Logs

For keeping logs there are 2 different methods: one is for printing to the command line, and one is for printing to the log file. KeepLog takes a job variable as input and prints it to the job.log file. The id, status, request time, end time turnaround time and pad of the jobs are printed. The KeepLog method is called whenever a job is completed (in the pad methods after the sleep). We used a file mutex to make sure that only one job is writing to the file.

The second method is named Print_Jobs_Terminal, it has an initial while loop which checks if n seconds have passed after the beginning of the program. After n seconds it starts the main job. It first has while loop like the other methods which checks whether the simulation time has been reached. Then it stores the IDs and the types of the jobs in the Pad A queue and Pad B queue. It also checks and stores the IDs of the jobs in land, launch, assembly, and emergency queues. Then it prints the stored IDs into the terminal and sleep for 1 second. In the output some IDs are repeated because that job has been enqueued to the pad queues and waiting for its turn on the pad. The launch, assembly, land, and emergency queue outputs are not just outputs of their own queue but their own queue plus the jobs in the pad with the same job type.

EventID	Status	Request_Time	End_Time	Turnaround_Time	Pad
2	L	2	4	2	B
1	D	0	6	6	A
3	L	6	8	2	B
5	L	8	10	2	A
7	D	12	16	4	A
8	L	12	18	6	A
4	A	6	20	14	B
10	L	18	20	2	A
12	L	20	22	2	A
13	L	24	26	2	A
15	L	26	28	2	A
16	L	28	30	2	A
6	A	8	32	24	B
17	L	30	32	2	A
18	L	32	34	2	A
14	D	24	38	14	A
20	L	38	40	2	A
9	A	12	44	32	B
21	L	42	44	2	A
22	L	44	46	2	A
23	L	46	48	2	A
25	L	48	50	2	A