EE463 Operating Systems Term Project-B Memo 04/20/22

Hesham T. Banafa
Muhannad Al-Ghamdi

**Current state summary:**
The car-park is operating correctly in terms of generating new arrivals based on the provided RNG from the main loop. Currently, the main loop generates, allocates and initialize incoming cars, then later added to the Car-park queue if preconditions checkout. We have also implemented in-valets code for thread management and synchronization of shared data of the process. The in-valets wait on semaphore initialized to 0 named ‘arrivals’ accordingly with it’s purpose. It is noted that the semaphore implementation used allow the threads to sleep until the semaphore is ‘posted’ or ‘signaled’. In-valets currently take arriving cars from the queue and park them in empty slots.

**Finished:**
Queue implementation is complete and tested to operate correctly.
Code is split into files for isolation (main, monitor, in-valets, out-valets).
Main loop generating incoming cars.
Monitor calculates some statistics, updateStats, print to terminal, and re-draw GUI window.
GUI working with queue.
CLI arguments parsed correctly according to Project document.
Makefile updated to compile correctly.

**Work in progress:**
SIGINT and SIGTERM handling: graceful thread termination and cleanup and print final car-park report to stdout.

Out-valets: initial code as per design.

Testing for deadlocks: theory and testing.

**Problems during work:**
C language tool chain does not allow multiple includes. Having circular includes produces this compile-time error. It was later found that it is common practice to use ‘include guards’. The following illustrates the pre-proccessor directives for include guards.

```
#ifndef HEADFILENAME_H
#define HEADFILENAME_H
{{ Header content, varibale and function decelerations }}
#endif
```

During compile-time, the per-procseesor checks if the compile-time symbol HEADFILENAME_H is defined to decied whether include the whole header again. It serves the purpose of “include once” or

```
#pragma once
```

Falsely implemented CarInit when it is already implemented in CarPark.o. ‘ld’ output mentions the CarInit is defined multiple times.

EE463 Operating Systems Term Project-B Memo 04/20/22

Hesham T. Banafa
Muhannad Al-Ghamdi

**Not Started:**
Out-valet functionality.

Calculations of all statistics.

Proper thread termination is good practice, despite the fact that the OS (Linux) frees all the resources held by the starting process after terminaion. Graceful termination of the program can be done in 2 common methods. The first method is to send pthread_cancel() to each thread. Ideally, the thread should terminate immediately. However, if the thread unsets ‘cancel state’ with pthread_setcanelstate(), a cancel request is then queued until the thread re-enables cancellation.

Moreover, pthread_setcanceltype() alllow to set asynchronous cancel where the thraed is canceled at any point, which may introduce data corruption of shared data is used. Another type is deferred, where the thread is canceled when it reaches a cancel point, by calling pthraed_testcancel() when ever is safe.

The second method, which is more reliable in our testing is to use a shared flag. Since threads by default share data with the parent thread, we can make the thread check the value of a global flag to indicate when it should call pthread_exit().