Operating Systems (70033) – Assessed Coursework
Producer-Consumer

Due date: 8^{th} **December 2021**
(Submission on CATE by 19:00)

The purpose of this coursework is to use the *producer-consumer* scenario and gain experience into using system calls for semaphores. A typical producer-consumer scenario consists of a group of producers 'producing' jobs with a group of consumers 'consuming' these jobs. In order for the producers and consumers to be able to communicate with each other, a shared data structure is traditionally used to store the jobs and with which the producers and consumers interact. Due to the shared nature of this data structure, it needs to be protected by synchronisation mechanisms to ensure consistent access to the data.

In this coursework, we implement the producer-consumer scenario using POSIX threads, where each producer and consumer will be running in a different thread. The shared data structure used will be a *circular queue*, where a producer adds to the end of the queue and a consumer consumes from the front of the queue. The queue will contain details of the job, which has an id (the location that it occupies in the queue $\Rightarrow$ job ids will start from 0) and a duration (the time taken to 'process' the job – for this coursework, processing a job will mean that the consumer thread will sleep for that duration).

We will implement a **main** program that will contain two functions – one for the producer and one for the consumer, each of which will be run in a separate thread. Depending on the number of producers and consumers required, we will be creating those many threads.

The sequence of steps for the main program (and the two functions) are as follows:

1. Main program
   (a) Read in four command line arguments - size of the queue, number of jobs to generate for each producer (each producer will generate the same number of jobs), number of producers, and number of consumers.
   (b) Set-up and initialise the required data structures and variables, as necessary.
   (c) Set-up and initialise semaphores, as necessary.
   (d) Create the required producers and consumers.
   (e) Quit, but ensure that there is process clean-up.
2. Producer
   (a) Initialise parameters, as required.
   (b) Add the required number of jobs to the circular queue, with each job being added once every 1 – 5 seconds. If a job is taken (and deleted) by the consumer, then another job can be produced which has the same id. Duration for each job should be between 1 – 10 seconds. If the circular queue is full, block while waiting for an empty slot and if a slot doesn't become available after *20 seconds*, quit, even though you have not produced all the jobs.
   (c) Print the status (example format given in *example_output.txt*).
   (d) Quit when there are no more jobs left to produce.
3. Consumer
   (a) Initialise parameters, as required.
   (b) Take a job from the circular queue and 'sleep' for the duration specified. If the circular queue is empty, block while waiting for jobs and quit if no jobs arrive within *20 seconds*.

(c) Print the status (example format given in *example_output.txt*).

(d) If there are no jobs left to consume, wait for *20 seconds* to check if any new jobs are added, and if not, quit.

Your solution must handle multiple producers and consumers concurrently and use semaphores to restrict access to the shared resource (in this case, the circular queue). The solution should also deal with input errors, system call errors, and any other errors, appropriately. You are welcome to come up with challenging/complicated/interesting solutions ☺. *Hint*: in order to make the Producer/Consumer wait for 20 seconds, the elegant (and preferred) solution is to implement this delay as part of the semaphore. Useful command-line tools for debugging by checking the status of semaphores are *ipcs* and *ipcrm*.

Note: Download the outline source code from CATE. You are not obliged to use this; however, they should simplify the process of achieving working solutions. You can choose to change these files as necessary. Details on the support files are provided below:

- These support files have been tested on Linux in the DoC labs. You can however choose to use your own programming environment.
- The file *main.cc* contains the overall structure of the programme and has a simple example of creating and using a thread.
- The file *helper.h* includes the required header files and some pre-defined variables. Please **ensure** that you **change** the value of the semaphore key in order for it to be unique amongst students.
- The file *helper.cc* contains a few helper functions, including functions for creating, using and destroying semaphores.
- The file *example_output.txt* contains an example output using the following parameters – Queue size = 5, Number of Items produced per producer = 6, Number of producers = 2, and Number of consumers = 3. Please note this is just a *suggested* output – you **do not** have to make your output look *exactly* similar ☺.
- The *Makefile* allows users to use the *make* command on Linux to compile the various parts of the coursework. It can also be used to help configure your preferred development environment with the correct commands, flags and parameters.

You should hand in the following to CATE:

- An archive (**code.tar.gz**) containing **only** the C++ files (*main.cc, helper.cc, helper.h*), the *Makefile*, and a file *output.pdf*, containing the screenshot(s) of your programme compilation, execution and output. **Do not** put these files in a separate directory and remove any extraneous directories (such as .git, __MACOSX, etc.). A small penalty will be applied for submissions not meeting the criteria.

Tutorials

Be aware that the basic functionalities of these may slightly differ from the lectures.

Semaphores – `https://beej.us/guide/bgipc/html/multi/semaphores.html`
POSIX Threads Programming – `https://computing.llnl.gov/tutorials/pthreads/`

Useful Links

Workstations in DoC – `https://www.doc.ic.ac.uk/csg/facilities/lab/workstations`
Remote Login in DoC – `https://www.doc.ic.ac.uk/csg/services/linux`
Creating and unpacking .tar.gz files – `http://arxiv.org/help/tar`
ipcs - provide information on ipc facilities – `http://linux.die.net/man/1/ipcs`
ipcrm - remove a message queue, semaphore set or shared memory id – `http://linux.die.net/man/1/ipcrm`