Detailed Course Deliverable Information

INFR 2820U & 2830U: Algorithms and Data Structure + Operating Systems

Combined Project

The University of Ontario Institute of Technology is a publically funded degree-granting university in Oshawa, Ontario. Boasting thousands of students and hundreds of employees, UOIT has expressed interest in funding research related to software-based switching technologies and their use in virtualized environments.

Working together as a team, UOIT would like you to leverage your expert knowledge to develop a software-based switch emulator as a proof-of-concept for future research related to software-based switching technologies. Although traditional switching technologies have existed for decades, UOIT has decided to expand their technical research scope to include software-defined networking in the future. Before diving into such a complex topic, they would like to see if your team is capable of emulating fundamental switching concepts in software using knowledge acquired from your Algorithms and Data Structures and Operating System courses.

For this portion of the ITSW, we have combined two course deliverables from Algorithms and Data Structures and Operating Systems into one single project. This project has components assessed by each of your professors based on their own metrics; however, together they should be implemented in a single piece of code.

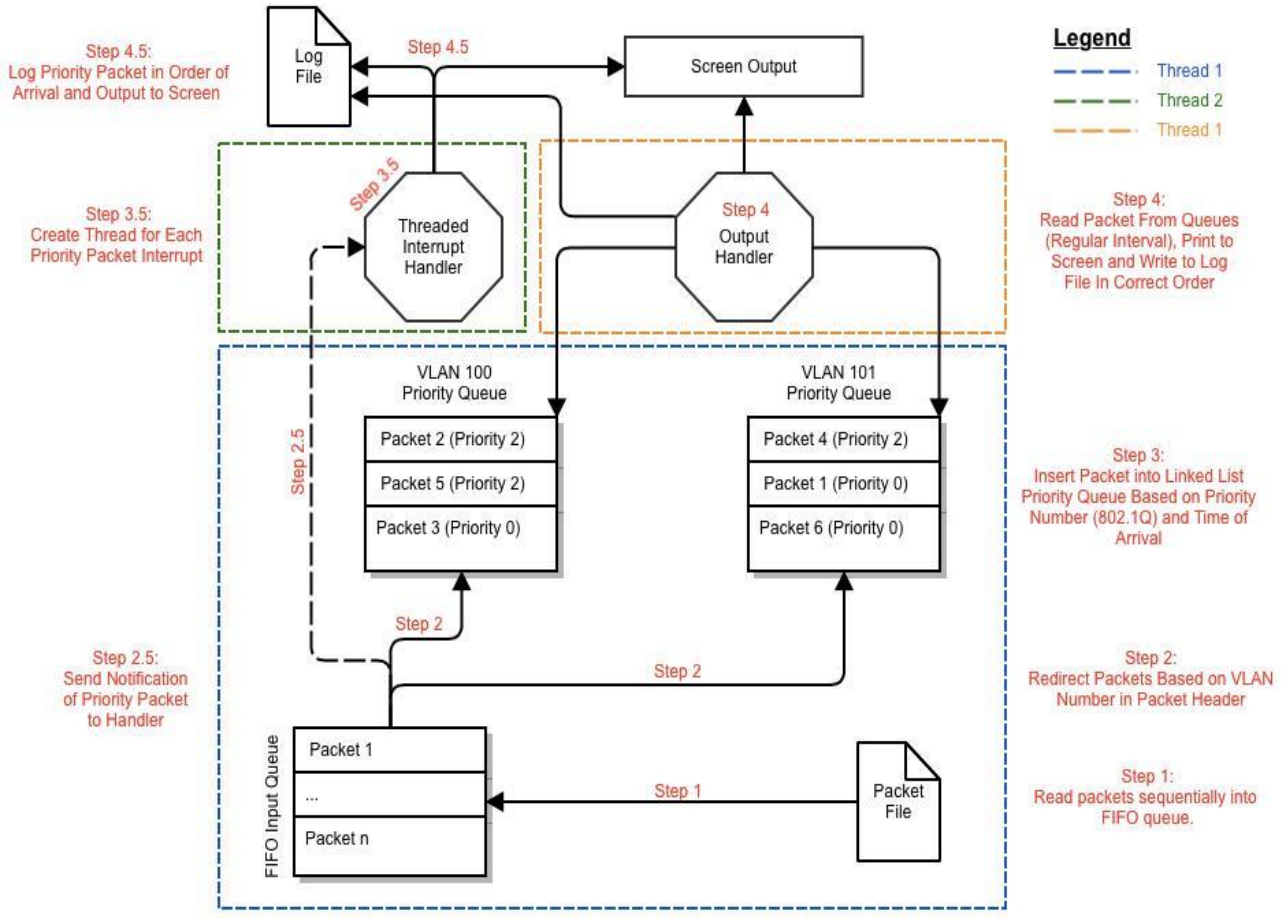

Figure 1. Switching Project Overview

Important Note: *Thread 1 components (blue boxed area) should be completed using knowledge from your ADS course, whereas threads 2 and 3 (green and orange, respectively) will require knowledge from your OS course.*

Instructions:

The above combined project should be structured and developed into a working piece of code that will be submitted to your professors at the end of this semester via their drop boxes on Blackboard. **Code should not be submitted to the ITSW coordinator!**

All project code should be implemented in C++ to leverage your existing knowledge of the language. All developed code should be object oriented and should meet the requirements outlined below.

External libraries may be used to implement components like threading and file management, etc.; however, all queue components should be developed according to the image above (i.e. a FIFO queue should be used to read in the initial packet file data and a priority queue should be used for the VLAN priority queues).

A packet file will be provided to you by your ADS professor in the upcoming weeks via Blackboard. Its contents will include a multitude of standard Ethernet frames including an 802.1Q tag. These packets will contain a basic ASCII payload that is printable to the screen (for example, containing a single English word) and should be processed using the architecture outlined in this section.

Additional details and technical requirements will be laid out by your professors as the semester progresses, **including report requirements for your OS course**. Please get in touch with the appropriate professor if you require additional clarification.

The project's due date is April 1st, 2018 at 11:59pm via Blackboard. You will need to submit your code to the Blackboard drop box for each of your courses (one submission for ADS, another for OS), unless otherwise stated.

Architecture Requirements

1. Set up **thread 1** for processing incoming packets. This thread should perform the following steps:
    a. Sequentially read packets from the provided packet file into a FIFO (first in, first out) queue for processing. **(Step 1)**
    b. Read packets sequentially from the FIFO queue and redirect them into one of two priority queues based on their VLAN tag and priority number (802.1Q). **(Step 2)**
        i. First, read the packet popped off the FIFO queue and determine its VLAN number by inspecting the packet's 802.1Q tag.
        ii. Next, determine which destination interface queue the packet needs to be inserted into (one priority queue should be created for VLAN 100 and another for VLAN 101).
        iii. Once you have determined the destination queue, inspect the packet's 802.1Q priority value and insert it into the correct destination queue. **(Step 3)**
            1. A priority value of '0' indicates 'best effort' and should be inserted into the destination queue based on its time of arrival (i.e. it should be added to the end of the queue)
            2. A priority value of '2' indicates a 'excellent effort' priority and should be inserted in front of existing packets in the queue based on its time of arrival (i.e. it should be inserted ahead of priority '0' packets, but should not be put ahead of already queued priority '2' packets).
        iv. If a priority '2' packet has been detected, a notification and copy of the packet should be sent to the threaded interrupt handler (thread 2). **(Step 2.5)**
    c. Once all packets have been inserted into the correct interface queues, this thread should terminate.
2. Set up **thread 2** for processing incoming priority packet interrupts. This thread should perform the following steps:
    a. Priority packet interrupts should be received as a copy of the original packet.
    b. For each priority packet interrupt, a new child thread should be created to log an alert to both a log file and the screen. This thread should be given an object containing information about the packet (i.e. source MAC and destination MAC) and a timestamp of when the interrupt was received. **(Step 3.5)**
    c. A lock should be created to control which threads are able to write to the log file and the screen. A thread should only be able to write to the log file and screen if it has access to the lock. When a thread is given the lock, it should

write an alert to the log file and screen stating that a priority packet was received and should include information about the packet's destination and source MAC address. **(Step 4.5)**

i. Ultimately the result should be that threads log priority packet alerts to the log file and screen in the order of arrival. If the lock handling mechanism is not implemented properly, a race conditional will occur causing threads to write to the log file and screen in an unpredictable fashion.

3. Set up **thread 3** for printing queued packets to the screen and logging them to the log file. This thread should perform the following steps:
   a. All packets processed by this thread need to be outputted to the same screen used by thread 2 and the same log file used by thread 2.
   b. Every 1 second, a packet should be read from the front of interface #1's priority queue (VLAN 100) and removed from its queue. **(Step 4)**
      i. The thread should then wait to have access to the lock used to control writing to the log file and screen.
      ii. Once the lock is obtained, the thread should write the packet's contents to the log file (source and destination MAC + payload contents) and to the screen.
      iii. After the log and screen operation has been completed, the lock should be released.
   c. Immediately after, a packet should be read from the front of interface #2's priority queue (VLAN 101) and removed from its queue. **(Step 4)**
      i. The thread should then wait to have access to the lock used to control writing to the log file and screen.
      ii. Once the lock is obtained, the thread should write the packet's contents to the log file (source and destination MAC + payload contents) and to the screen.
      iii. After the log and screen operation has been completed, the lock should be released.
   d. This thread should sleep for 1 second and then loop to continue processing the queued packets until both queues are empty. Once both queues have been emptied, the thread should terminate.

ADS Reporting Requirements

A report should be submitted with the following:

- Implement the priority Q's using a heaps
- Evaluate the performance for different sizes of the input files.
- Compare the run-time performance to the theoretical performance

OS Reporting Requirements

A report should be submitted with the following:

- How many processes can be in the running/executing state simultaneous (number of threads).
- How does the operating system use interrupts? (interrupts methods)
- Why are system calls needed? Try to think of the benefits on all levels (dual -mode operation)