Assignment 1. Device Drivers for Shared Message Queues (100 points)

Assignment Objectives

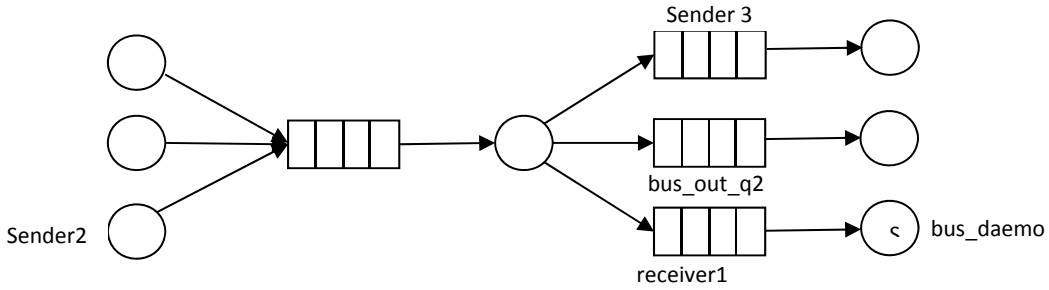
1. To learn the basic programming technique for module and device driver in Linux kernel.
2. To learn thread programming in Linux.
3. To use x86's TSC (Time stamp counter) to measure elapse time.

Project Assignment

Message bus is a useful middleware to support the communication among user applications. The core component of a message bus is its bus daemon which manages connections and forwards messages through message queues. An example of message buses is D bus, developed as part of the freedesktop.org project for desktop applications.

In this assignment, you are required to develop two programs on Galileo board, including

1. A device driver for shared queues which perform the basis enqueue and dequeue operations and record the accumulated queueing time (a queueing time is the elapse time between enqueue and dequeue operations).
2. A test program that initiates multiple threads to access the shared queues as illustrated in the following test case:



In the test case, there are 3 sender threads, one bus daemon thread, and 3 receiver threads, running in a user application. The shared queues are implemented in kernel space and managed by device driver "squeue". To avoid the dynamic creation of shared queues, four queue devices, named "bus_in_q", "bus_out_q1", "bus_out_q2", and "bus_out_q3" are created and added to Linux device file systems when the device driver for the shared queues is installed. The devices act as FIFO (first-in-first-out) queues where messages can be enqueued and dequeued. Each message is represented by a message id, source and destination ids, an accumulated queueing time, and a variable-length character string of maximal length 80. When a message is dequeued, its queueing time is computed by the driver based on the Time Stamp Counter (TSC) values of the enqueue and dequeue instants. The accumulated queueing time of a message is the total queueing time since its first enqueue operation. So, when a message is dequeued, its queueing time should be added to the total. Each queue can be shared by multiple tasks (processes or threads) concurrently and should be implemented as a ring of pointers which point to a dynamically-allocation memory region for storing message body. The default queue length is 10, and can be defined in a macro. The device driver should implement the following file operations:

- *open:* to open a device ("bus_in_q", "bus_out_q1", "bus_out_q2", and "bus_out_q3").
- *write:* to enqueue a message to the device. If the queue is full, -1 is returned and errno is set to EINVAL.

- *read:* to dequeue a message (with the valid queueing time) from the device and copy it to the user buffer. If the queue is empty, -1 is returned and errno is set to EINVAL.
- *release:* to release the file structure.

In the test program, you will create 7 sender threads and 1 receiver thread which use the shared queues to pass message as shown in the test case diagram. The threads operate as follows:

- Sender threads which create one message at a time and write it to bus_in_q. They then take a nap (*sleep)* for a random interval before sending the next message. If an enqueue operation fails, the sender retries it after another nap. When a message is created, it is assigned with a global sequence number as the message id. The destination is chosen randomly from all receivers. The string of each message can consist of arbitrary characters. The length of each string is uniformly distributed from 10 to 80. Apparently, the initial accumulated queueing time must be 0.
- The bus daemon reads messages from bus_in_q and then forwards (writes) the message to one of the bus_out_q according to message destination id. It will take a nap when -1 is received from read or write calls. The failed calls should be retried after the nap.
- A receiver thread repeats the operations of reading all queued messages from its input queue, printing out the message id and source id, as well as the accumulated queueing time, of each received message, and taking a nap when the queue is empty.

In your test program, threads should be initiated first. Then, all sender threads create and send messages continuously for a 10 second period. After all massages are received, your test program ends. The nap time for all senders, bus daemon, and receiver is an interval distributed randomly between 1ms and 10ms.

[For CSE 598 students]

Other than the driver and test programs, you will need to prepare a profiling report. You can use the Linux *perf* tool to collect CPU cycles and numbers of instructions executed in user and kernel space, and memory usage. The profiling should be collected from the execution of

1. Your testing program including the drivers (dynamically allocated message buffers and printing)
2. Your testing program including the drivers (statically allocated message buffers and no printing). Note that each token in the queues needs at most 100 bytes. You can statically allocate 1000 bytes for each queue to avoid kmalloc and free calls for enqueue and dequeue operations.

Due Date

This is assignment is due at 11:59pm on Sep. 22.

What to Turn in for Grading

- Create a working directory to include your source files, makefiles, and performance report (for CSE 598). Your source files should include the main program *main_1.c,* the driver *Squeue.c ,* and any other .c or .h files for the assignment.
- Comment your source files properly and rewrite the readme file to describe how to use your software.
- Compress the directory into a zip archive file named cse438(598)-lastname-assgn01.zip. Note that any object code or temporary build files should not be included in the submission.
- Submit the zip archive to Blackboard by the due date and time.
- Failure to follow these instructions may cause an annoyed and cranky TA or instructor to deduct points while grading your assignment.