/proc/meminfo Covert Channel User Guide
Written by: Benjamin Steenkamer, February 2018

Compiling Channel:

Type `make`
Two executable should be generated: `source` and `sink`
A subfolder should also be generated: `output`
This subfolder is used by the sink program if a "-p" argument is given.

Running Channel:

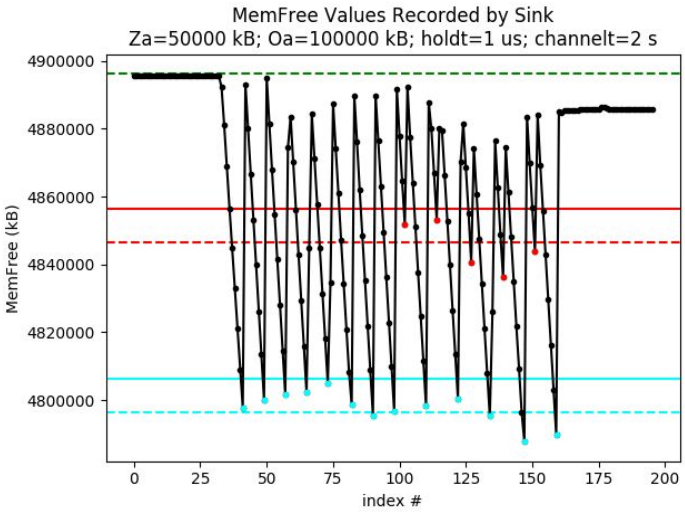
The sink receives data by recording the MemFree values for `CHANNEL_TIME` seconds. After that many seconds, it then goes back and attempts to interpret the data. The sink must be started *at least* slightly before the source. This allows the sink to start recording the MemFree values just before the source starts sending. Both the source and sink will spend `CALIB_TIME` seconds calibrating baseline values. This time in not included in `CHANNEL_TIME`.

Run the source from one container: `./source`
Run the sink from a different container: `./sink`
The containers must be running on the *same physical machine*. Both programs will output messages to state what stage of the transmission they are in.

The sink can take one argument: `./sink -p`
This will cause the sink to dump it's recorded data to the `output` subfolder and generate a plot of the transmission. This plot is very useful when trying to visualize or debug the channel performance. The actual plotting of the data is handled by `plot.py`. You can change the settings of the graph from this file, such as enabling the legend.

Plot Explanation:

An example of a plot is shown below. This transmission shows 8 1's being transmitted in a row (a start sequence), then an alternating pattern of 5 0's and 5 1's (the actual transmission data).



Za: Zero alloc; The amount of memory the source is allocating each time it sends a 0.
Oa: One alloc; The amount of memory the source is allocating each time it sends a 1.
holdt: hold time; The amount of time the source is keeping a chunk of memory allocated before it frees it. The source also waits this amount of time after freeing memory before it allocates a new chunk.
channelt: channel time; The amount of time the sink will record MemFree values after the calibration is done.
Black line with dots: The recorded values of MemFree by the sink. The dots are actual values the sink recorded, and the line is there to make rise and fall of the values easily visible.
Dashed green line: The average value of MemFree as calculated by the sink during the calibration stage. After a piece of memory is freed by the source, the value of MemFree should ideally return to around this line.
Solid red line: The largest value of MemFree that will be recognized as a 0 during analysis by the sink. The offset of this line from the dashed red line is determined by the DETECT_VARIANCE variable. If DETECT_VARIANCE is 0, the two lines will overlap.
Dashed red line: The nominal 0 value. In an ideal environment, each transmission of a 0 should "peak" at this line.
Red dot: The index where the sink has found a local minimum and determined it represents a 0 being transmitted by the source.
Solid cyan line: The largest value of MemFree that will be recognized as a 1 during analysis by the sink. As with the solid red line, DETECT_VARIANCE affects this line's position.
Dashed cyan line: The nominal 1 value. In an ideal environment, each transmission of a 1 should "peak" at this line.
Cyan dot: The index where the sink has found a local minimum and determined it represents a 1 being transmitted by the source.
Index #: This will potentially be replaced in the future with the time the sink recorded the MemFree value.

Configuration Values in `meminfo.hpp`:
`HIGH_BIT_ALLOC`: How much memory in kB will be allocated by the source when sending a 1.
`LOW_BIT_ALLOC`: How much memory in kB will be allocated by the source when sending a 0. `HIGH_BIT_ALLOC` should always be greater than `LOW_BIT_ALLOC`. Making `HIGH_BIT_ALLOC` twice as much as `LOW_BIT_ALLOC` helps prevent misreads if there is moderate noise in the channel. Making these two values large (100000/200000 kB or larger) will increase channel transmission reliability and help prevent noise interference at the cost of making the channel bandwidth slower (allocating and setting large amounts of memory is slow). Decreasing these values (below 100 MB allocs) makes the channel bandwidth increase, but also makes the transmission much more susceptible to noise interference.

`DETECT_VARIANCE`: Used by the sink when detecting values in its recorded data. This represents how lenient the sink is when detecting a 0 or 1. When the channel starts, the sink calculates the nominal values of MemFree that will be used to represent a 0 or 1. `DETECT_VARIANCE` allows the threshold

for a 0 and 1 to be widened. For instance, if `DETECT_VARIANCE` is 0.2 or 20%, values above the nominal 0 value, but are still within 20% percent of it will be detected as a 0. Values above the nominal 1 but are still within 20% percent of it will be detected as a 1. This helps the sink still detect values when there is noise present. Do not make this value too big, or incorrect readings will occur.

`CHANNEL_TIME`: How many seconds the sink will record MemFree values. The source has at most this much time to send all of its data to the sink. The source can finish sending data in less time than `CHANNEL_TIME`, but the sink will always record for this many seconds, regardless of the status of the source. If the source takes longer than `CHANNEL_TIME` seconds to send all of its data, the sink will not be able to see all of the transmission. Increase this value if the source says the channel expired before it could finish the transmission.

`CALIB_TIME`: How many seconds the source and sink spend calculating the average value of MemFree. This value doesn't need to be large. Calibration by both the source and sink occur before they start their main task.

`CALIB_DELAY`: The period in microseconds, of when the source and sink record MemFree during the calibration period. This can be a large value (+0.1 seconds) as they don't need that many data points to calculate a good average value.

`RECORD_DELAY`: The period in microseconds, of when the sink records MemFree values during the channel transmission. This value should be in the 1 to 10 millisecond range so that the sink does not miss a transmission from the source. If the sink is missing many transmission form the source, try decreasing this value so the sink records more often. You don't want to make this value too small, or the sink will be recording an unnecessarily large amount of data. This up more system memory and will introduce noise to the channel.

`HOLD_TIME`: How long, in microseconds, the source will keep a piece of memory allocated for a transmission of a 0 or 1. After this amount of time, the source will deallocate the memory and then wait another `HOLD_TIME` microseconds before allocated memory for the next transmission. This means the period of one transmision will be a minimum of 2 * `HOLD_TIME` microseconds.
This value was meant to give the sink enough time to detect the presence of an allocation. However, this value can be very small now, as the sink's algorithm has changed and can detect very short allocation times, provided `RECORD_DELAY` is sufficiently small. You may even be able to make this value 0.

Note:
Changes made in this header file are sometimes not detected by the Makefile when compiling the `source` executable. If `source` is not compiling with the new values in the header file, type `make clean` and then `make`. I will try to figure out why this occurs.

Configuration Values in `meminfo.cpp`:

`int src_seq[]:` This is a sequence of bits sent by the source before it begins sending the actual data bits. After recording for `CHANNEL_TIME` seconds, the sink will first look for this sequence of bits to see if the source was present and transmitting at the time of recording. If the sink does not find this sequence in its recording, it will not read out any data. If the sink does find this sequence, it will start to read out all values that appear right after this sequence.

The sequence should be a unique sequence of bits such that it is easily distinguishable from channel noise. 8 bits in length is generally a reliable size for the sequence, although it could be made shorter.

Configuration Values in `source.cpp`:

`NUM_BITS_TO_SEND`: This is how many data bits the source will transmit to the sink. Right now, the source sends an alternating series of 0's and 1's. If this value is set to 10, for example, the source will send 5 0's and 5 1's in alternating order. If you want to change the sequence of bits the source sends, you can edit the logic in `send_data()`. This value does *not* include the number of bits in `src_seq`.

Configuration Values in `sink.cpp`:

There are no configuration variables to be set in this file. See `meminfo.hpp`.