L41: Lab 2

Alexandru-Andrei Iacob<aai30>

March 16, 2022

1 Introduction

The process model serves as both a means of isolating program execution [Saltzer and Schroeder, 1975] and as the primary computational abstraction of the system [Silberschatz et al., 2018, pg.105]. The goal of this work is to understand the overhead of Inter-Process Communication (IPC) in the case of the FreeBSD implementation of the UNIX pipe IPC. Given that transferring data through this method requires two copies [McKusick et al., 2014, pg.62], the most relevant implementation aspects of this procedure is whether the data is passed by value or if one copy is elided by pining a page, marking it as copy-on-write (COW) and directly sharing it with the receiver. This report intends to explore the performance impact of buffer sizes on pipe IPC as well as the effectiveness of VM optimisations with regards to the memory threshold from which they are applied.

The experimental design is organised on three levels all based on an IPC benchmark transferring data between two processes with a given buffer size. First, the bandwidth of buffer sizes between 32B and 64MiB is measured with the default VM threshold, as well as with the VM optimisation always turned off and always turned on. Second, DTrace [Cantrill et al., 2004] is used to profile the memory returns of write() and read(), the time spent in either as well their scheduling behaviour. Third, source code inspection and hardware performance counters are used to construct a working model of the overall pipe IPC implementation and its interactions with the microarchitecture of the ARM A72 processor, especially its memory hierarchy. Overall the results show both that large benchmark buffersizes VM optimisations are beneficial in reducing pipe IPC overhead, although severe limitation arise both from the pipe IPC implementation limiting data transfer and from the L2 cache becoming overwhelmed as benchmark buffers grow too large. Furthermore, properly setting the threshold for using VM optimisations has sever impact on performance as memory transfers result in too many context switches. Although promising, these results often draw upon Hardware Performance Counters (HWPMC) for justification, which incur a statistically significant and very large effect on performance. This fear is well-founded, however analysis shows that the broad behavioural trends are largely unaffected by the probe effect and the conclusions can be tentatively trusted.

2 Experimental setup and methodology

Given the security and abstraction benefits of process isolation, operating systems tend to disallow by-default direct memory accesses between the virtual address spaces of different processes. The overhead incurred by having to transfer data using pipe IPC is significant as it must pass through the kernel which itself must create an extra-copy for security reasons. The primary goals of this report is to understand what can be done to reduce this overhead either explicitly by the user-space program (larger buffers) or transparently by the kernel (VM optimisations).

2.1 Hypotheses

1. *Larger pipe buffer sizes improve IPC performance.*
2. *Page-borrowing virtual-memory optimisations:*
   (a) *Are well tuned to current microarchitectures*
   (b) *Always achieve a performance improvement, when enabled at or above the default 8KiB threshold, over the unoptimised baseline.*
3. *The probe effect associated with using HWPMC to explore this workload is negligible. (L41 only)*

2.2 Platform

All experiments are run on a Raspberry Pi 4 - Broadcom 2711. It uses a with a Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz under-clocked at 600 MHz running FreeBSD. Only one of the cores is used for the purposes of this report. Main memory is 8GiB LPDDR4-2400 SDRAM while external memory is maintained in a 64GiB SDCard. Importantly for this report, the processor has a 32KiB data cache, 48KiB instruction L1 cache per core, and 2MiB shared L2 cache. It supports page sizes of 4KiB, 2MiB, and 1GiB.

2.3 Benchmark

The IPC benchmark sets up two processes with access to a pair of IPC endpoints referencing a shared pipe. It tests the performance of the pipe IPC by carrying out write() and read() system calls to send and receive a given amount of bytes. The benchmark pegs itself to one CPU in order to avoid interference from concurrent execution. The amount of data sent at each step—until the total number of bytes has been moved—is based on the benchmark buffersize. It is important to note that both write() and read() may return partial results. The provided benchmark, despite this, throws an exception if the sender's write() operation fails to write the expected amount, however, the read() operation of the receiving process is allowed to be re-invoked if it fails to read the requested amount.

The benchmark can return the average bandwidth across the IPC object as well as time, system-time (stime), user-time (utime), number of messages sent (msgsnd) or received (msgrcv) and the number of voluntary (nvcsw) and involuntary (nivcsw) context switches. It can also gather a number of HWPMCs split across four modes *arch*|*dcache*|*instr*|*tlbmem*. The specific counters used for the analysis will be introduced in Sec.2.4.4.

2.4 Experiments

2.4.1 Statistics and error mitigation

All non-DTrace experiments bellow use 10 benchmark iterations while throwing away the first iteration to avoid startup effects. They were all ran one-after another on a freshly rebooted system with a single ipykernel and ssh connection running. The median is used as the primary summarisation tool given its resistance to skewed distributions alongside the semi-inter-quartile range (SIQR) as the measure of error [Jain, 1991, sec.12.3 and sec.12.8]. The secondary summary statistic is the mean with its standard deviation as the error, it is used when effect sizes are reported. When the exact same measure with different lines is shown side-by-side the y-axis is scaled to the maximum of the two as to allow comparison. The x axis is always the buffersizes and shared. DTrace experiments are only run for 1 iteration, however, their results are used merely as high-level pointers for research directions—and relegated to the appendix.

Results relating to the effects of VM optimisation or to the HWPMC probe effect are statically verified through either a statistical significance test or an effect size calculation for *each buffersize*. This is done because many of the observable behaviours are buffersize-dependent and could easily become indiscernible if all buffersize data was combined before the test. The test used is the two-tailed Student's t-test [Student, 1908] with a significance threshold set to $\alpha = 0.05$. All plotted p-values are capped at 0.1 to allow for easy viewing—their magnitude is not to be considered relevant outside of passing or not passing α. The ttest is paired with the Hedge's g [Enzmann, 2015] effect size that measures the gap between the mean of the two groups divided by the pooled standard deviation—tuned for small sample sizes. The author is not aware of any strong general guidelines for interpreting such effect sizes in computer science, as such the interpretation is based on the generalisable argument of Wilcox [2019, Concerns about Cohen's d, pg.4] which uses the standard deviation gap to determine how exceptional a result is in terms of its quantile—e.g by Chebyshev's inequality (A.4).

2.4.2 Performance

This set of experiments measures the benchmark performance when transferring $16\mathrm{MiB} = 2^{24}\mathrm{B}$ of data using power-of-two user-space buffers ranging from $32\mathrm{B}$ to $16\mathrm{MiB}$ and the default VM optimisation threshold in order to directly evaluate H.1. For analysing the overall benefits of VM optimisation, the benchmark will be ran with thresholds $th \in \{32\mathrm{B}, 8\mathrm{KiB}, 32\mathrm{MiB}\}$ signifying the VM optimisation being always-on (Always-Opt), starting at the default threshold (Default), or always off (No-Opt). A ttest is performed between Default and No-Opt to test H.2b at each buffersize for the standard FreeBSD configuration and between Always-Opt and No-Opt to test the effects of VM optimisation overall.

2.4.3 Tracing and profiling

The performance analysis in Sec.2.4.2 is complemented by tracing and profiling the behaviour of the write() and read() system calls. The first level of this investigation includes counting the number of write() and read() calls and aggregate statistics of the memory (buffersize) they are called with as well as the actual memory they report as having written/read. Alongside these aggregations, the exact memory values are saved. The second level inspects the scheduling behaviour of the CPU, specifically when are the threads running each of the system calls on/off cpu as well as when they go to sleep while waiting for data to be written into/read from the kernel pipe IPC buffer.

2.4.4 Hardware behaviour investigation

These experiments only use thresholds $th \in \{32\mathrm{B}, 32\mathrm{MiB}\}$ as the behaviour of the Default condition is a mixture of No-Opt and Always-Opt. This allows for an easier focus on the interaction between VM optimisation and the hardware while testing H.2a.

The expected cause of copy elision's difference in performance is a reduction in executed instructions. However, most executed instructions are going to be unrelated to the behaviour impacted by VM optimisation as a significant portion of them may not be loads/stores. Additionally, many loads and stores are not themselves related to transferring data between buffers and may in fact be the result of other system behaviour such as context switches or unrelated procedure calls. As such we would expect only the portion of load/stores related to buffer data transfer to be reduced. To quantify, this the ratio of loads+store between Always-Opt and No-Opt is compared against the bandwidth ratio and total instructions retired ratio. This leads to further dives into the cycles per instruction (CPI) of each threshold as well as their memory access patterns in terms of L1D and L2D cache hits rates. Finally, main memory usage is quantified by proxy through memory bus accesses.

2.5 Measuring the probe effect

Given the three relevant HWMPC modes as well as the two VM optimisation conditions are explored in Sec.3.4, showing graphs for the un-probed and probed-version behaviour for Always-Opt and No-Opt for all HWMPC modes is unfeasible. Consequently, the bandwidths obtained when running in one of the three HWPMC modes for a given buffersize and number of iterations are combined into one data series. Since the Hedge's effect size relies on the mean and not the median, this series is summarised via its mean and standard deviation for easy correlation with the chosen effect size. This has the added advantage of pooling data from all HWPMC bandwidths regardless of where they stand in the sorted order of the series. With the number of modes effectively reduced to one, the mean bandwidth of HWPMC modes is compared against the non-probed condition for both Always-Opt and No-Opt. The statistical and practical significance is then assesd via a ttest, effect size calculation and by comparing inflection points in the two graphs. The overall goal of this investigation is not to test H.3 in terms of the general absolute HWPMC performance impact, but for each buffersize in particular as to allow for a direct comparison of performance trends and inflection points against the non-probed conditions.

3 Results and discussion

The initial results of the performance analysis (Sec.2.4.2) shown in fig.1, partially confirm H.1 as they show a clear improvement in band-with as buffersize increases. However, the bandwidth peaks at a buffer size of $2^{16}\mathrm{B}$ and then declines. The decline is independent of VM optimisation as it is present for all conditions. The No-Opt threshold outperforms Default for buffer sizes $b = 2^{13}\mathrm{B}$ and $b = 2^{14}\mathrm{B}$, while nearly matching performance for $b = 2^{15}\mathrm{B}$. The difference between them is statistically significant as $p < \alpha = 0.05$ for $b \geq 2^{13}$. As such, the strongest version of H.2b stating the VM optimisation above the default threshold always outperforms the non-optimised version can be rejected. However, it does indeed appear that VM optimisation tends to either improve or not harm performance with the default threshold, fact confirmed by the statistically significant bandwidth improvements of Default and Always-Opt over No-Opt at $2^{15}\mathrm{B} \leq b \leq 2^{19}\mathrm{B}$.

It must be noted that the p-value graph does reveal experimental limitation, its large oscillations when comparing Default and No-Opt for $b \leq 8\mathrm{KiB}$ are unexpected as the two should have near-identical performances with only random differences. The limitation may be caused by the sample-size being too low or by running the No-Opt experiments right after the Default ones, thus having long-term system effects—e.g heat, memory segmentation e.t.c—impact results.

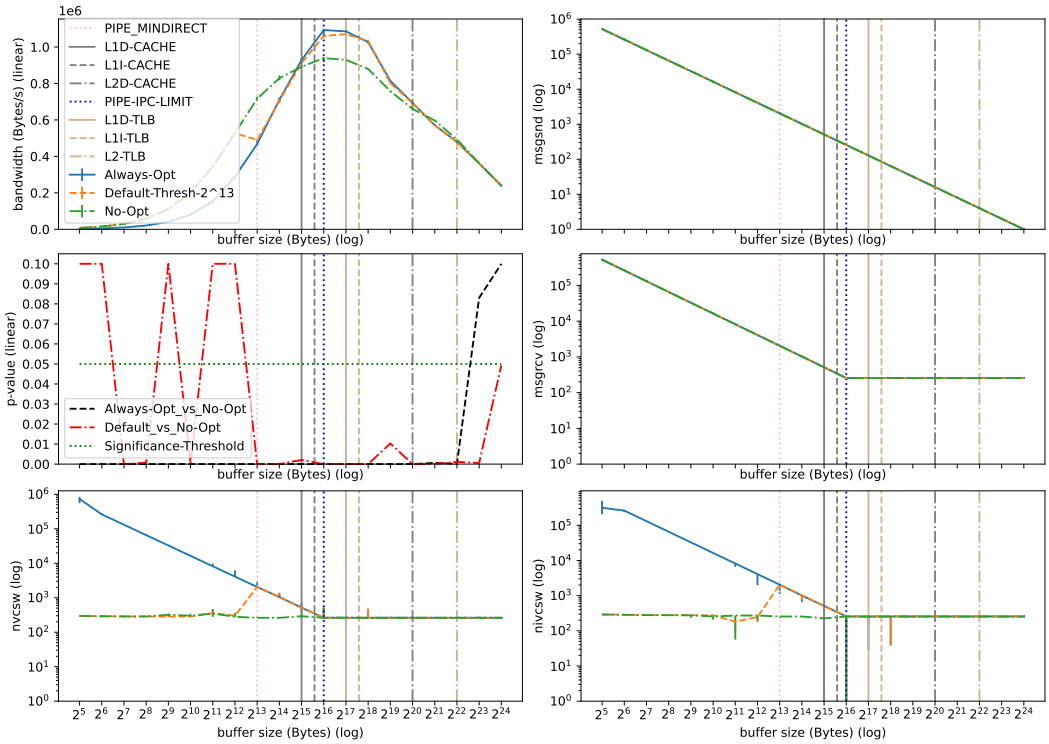


Figure 1: Benchmark measures sans msgsnd show the same extremum at $b = 2^{16}$B. P-values are capped (Sec.2.4.1).

VM optimisation overhead Given the previous test of H2b, from this point onward most experiments will refer to Always-Opt and No-Opt given that Default behaviour is a mixture of the two. The bandwidth differences between Always-Opt and No-Opt are always statistically significant for $b \leq 2^{23}$B. For small buffer sizes, the unremarkable performance of Always-Opt correlates with the number of voluntary and involuntary context switches up until $b = 2^{16}$B where they plateau and bandwidth peaks.

Pipe IPC and VM optimisation limitations Since the primary benefit of VM optimisation for this benchmark is eliding the copy when write() is called, the decline of both Always-Opt and No-Opt towards the same bandwidth serves as an indicator that increasing overhead in data transfer to the receiving process may hide any improvements brought by VM optimisation at large buffer sizes. This is further supported by the plateau in IPC messages received at the same $b = 2^{16}$B, meaning that increases in user-space buffer-size do not lead to decreases in the number of messages received—which should correspond to read() calls.

3.1 Memory usage at a syscall level

All results in this section serve purely as starting points for investigation, the DTrace output is show in A.1. When inspecting the system-call behaviour—by investigating only the parameters and return values—of write() and read() using DTrace, it becomes clear that the number of write() and read() system calls exactly follow the number of messages sent and received from fig.1. Furthermore, the aggregate memory behaviour of write() is entirely predictable as it never fails to send the exact amount of memory it is called with.

The aggregate memory behaviour of read() is identical up until a buffersize of $b = 2^{16}$B is reached, at which point the amount of memory that read() can return plateaus at 2^{16}B. For any buffersize b past these values, read() will have to be repeatedly called exactly $b/2^{16}$ times, with a total of $2^{24}/2^{16}$ calls for the transferring all the data. This is further confirmed by the exact memory values returned by read() equaling the maximal return amount.

3.2 Pipe IPC implementation and benchmark execution

Investigating the implementation of pipe IPC in the FreeBSD kernel reveals that the kernel buffers used for data transfer are capped. The system maintains a maximal total amount of memory for all pipes. If total pipe memory

usage exceeds 50% of max, the kernel caps pipe size to 2^{12}B. For usage less than 50% pipes are allowed to grow by doubling from the default 2^{14}B up until a hard-coded limit of 2^{16}B is reached. While the VM optimisation could avoid this limitation, the implementation chooses to enforce only allowing data transfer up to the kernel pipe IPC buffer size to keep the optimisation transparent. In the case of the synthetic benchmark used for this evaluation, the steps taken to eliminate all non-necessary work in the system make reaching the 50% limit highly unlikely.

At a system-call level the sender calls write() repeatedly until the entire data amount has been sent. If the underlying "pipe_write" implementation does not use VM optimisation then the user-space buffer contents get copied into the pipe IPC buffer, calls to write() execute until the buffer has been filled at which point write must call the receiving process to empty the buffer and receive the data. If VM optimisation is turned on write(), immediately tells the receiving process to copy data directly from the pinned pages. After all the data has been transferred, read() is explicitly called in the receiving process to put data into its user-space buffer. Even if that data is already available, read() still only returns chunks upwards of the max pipe IPC data transfer.

3.3 The context switch overhead of VM optimisation

The gap in voluntary and involuntary context switches observed between the No-Opt and Always-Opt conditions in fig.1 potentially explains part of the initial performance disadvantage incurred by Always-Opt. In the case of not using VM optimisation, data is copied from the user-space buffer to the pipe buffer by each consecutive write() call. Once the buffer is full and must be emptied, write() puts itself to sleep with msleep() and causes a voluntary context switch to the receiving process. The receiving process then empties the buffer and wakes up the write thread which is then ran by the scheduler after the receiver returns—causing an involuntary context switch. This behaviour means that the number of context switches for No-Opt is proportional to the total number of bytes divided by the max kernel pipe IPC buffer size $^{2^{24}}/_{2^{16}}$. This max kernel pipe IPC buffer size is always reached as no read() syscall is executed until after all write() calls are complete, meaning that the buffer can repeatedly grow and copy-over its previous content.

For a user-space buffer size which triggers the usage of VM optimisation, each write() call cause an immediate context switch. The receiving process must copy the page data directly. Before the maximum pipe IPC data transfer is reached, the number of context switches required for the benchmark is proportional to the total amount of data divided by the user-space buffer size $^{2^{24}}/_{b}$. After the user-space buffer size reaches the maximum pipe IPC data transfer, write() calls always cause a context switch even without VM optimisation since the kernel pipe IPC buffer is instantly filled. Consequently, the number of context switches is equalised. This behaviour is not sufficient to explain the gap between using and not using VM optimisation for smaller buffersizes, given that in fig.1 the crossover point is actually at a buffersize of 2^{15}B rather than the maximum pipe IPC transfer rate of 2^{16}B—indicating that copy elision benefits overtake context switch overhead. However, context switching is stated as the main reason for introducing a VM optimisation threshold in the implementation.

To confirm that the implementation of pipe IPC as well as the benchmark itself behave as described above, DTrace can be used to compare the system-call interface against the actual work carried out by the sender and receiver threads. The first step is to inspect how the distribution of system call execution time changes as buffer size increases—purely for observing the trend given that specific DTrace timestamps will be affected by the probe effect. All results for this are shown in A.2 due to space constraints. When applying quantize() to the system call time for a given buffer size below the pipe IPC max transfer, the No-Opt condition has a number of long write() calls approximately equal to the number of context switches—to empty the buffer—as well as approximately $^{2^{24}}/_{b} - nvcsw$ calls which can be several order of magnitude shorter. On the other hand, for Always-Opt transferring amounts of data below the pipe IPC limit no extreme skew is present. After the maximal IPC data transfer size is reached any differences in quantize() distribution largely disappear. Similar results, shown in A.3, are obtained when using the DTrace sched provider to track how often the sender or receiver thread goes to sleep/wake or on/off-cpu. Both align, as expected, with the context switching behaviour presented in Sec.3.3

3.4 Hardware behaviour

Fig.2 presents the ratio of speculative loads+stores between Always-Opt and No-Opt, it starts off extremely high because of context switching, becomes optimal ($\approx 25\%$ lower at $b = 2^{16}$B) and then decline until a $< 5\%$ difference remains. This behaviour mirrors that of bandwidth in fig.1 and should be considered the primary source of benefits from VM optimisation. The benefits of this reduction come despite total instructions retired being near-equal for $b \geq 2^{16}$B. Nonetheless, the bandwidth performance improvement is less than would be expected based on the overall decrease in loads+stores as it peaks at a mere $\approx 11\%$.

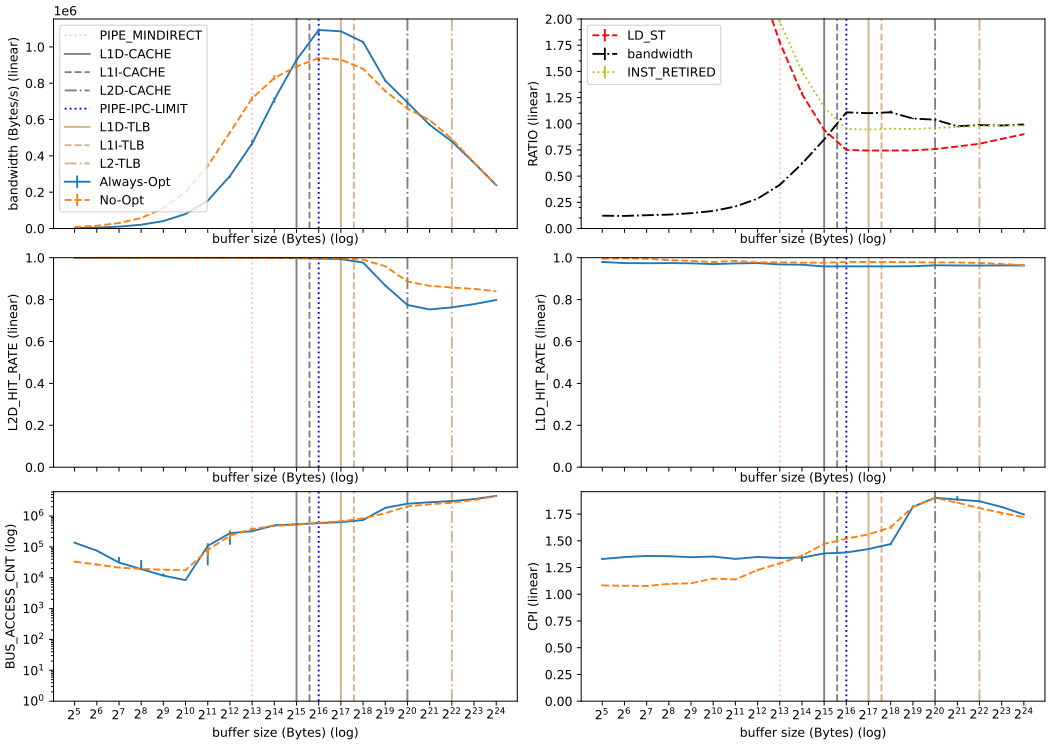


Figure 2: HWPMC results. The y-axis for ratios is capped as the initial values may be exceedingly large at ≈ 5.

The number of loads and stores cannot indicate anything about how efficiently each one of the instructions is executed. The first indication that VM optimisation may also impacts the efficiency of each load/store comes from the benchmark CPI. Always-Opt maintain a fairly constant CPI for each buffersize before peaking for buffersizes $2^{18}\text{B} \leq b \leq 2^{20}\text{B}$ and then declining. No-Opt CPI increases throughout until $b = 2^{20}\text{B}$ followed by a decline. Given that the primary difference between Always-Opt and No-Opt is in their usage of loads/stores, one potential explanation is that these CPI changes are caused by changes between the memory access efficiency of the two conditions. This can be inspected by looking at levels of the memory hierarchy.

Overall, some of the cost of having to create two copies without VM optimisation could be reduced via caching. Loads and stores are the triggers for cache accesses, as such the number of L1D and L2D cache accesses follow the number of load/stores and likely do not carry additional information. However, the L1D cache overall hit rate, fig.2 , shows a distinct advantage for No-Opt over the entire benchmark. Given that the task is to repetitively and sequentially read data into a kernel buffer, cache misses should also happen one-after-another in a predictable pattern that triggers hardware prefetching. Hardware prefetching can pull data regardless of the specific bounds on the benchmark buffer [Drepper, 2007, sec.6.3], meaning that it can potentially help improve performance of future write() calls . Furthermore, FreeBSD data copying functions such as the arm-optimised assembly memcpy make use of explicit instructions to cause prefetching when reading from a buffer. These mechanisms make memory access far less costly and thus may be responsible for the smaller bandwidth boost compared to the reduction in load-stores at the peak throughput point. Importantly, the L1D hit-rate for No-Opt remain lower than for Always-Opt even at the max kernel pipe size despite the fact that context switches and total instructions have equalised by this point. This shows that parts of the data may still be in cache when the sender makes the receiver empty the kernel buffer as the ARM-specific cpu_switch implementation does not flush the cache during switching.

While the L2D cache size is $2\text{MiB} = 2^{20}\text{B}$ and could thus hold a significant portion of the benchmark, the steep decline in L2D hit-rate after a buffer-size of 2^{18}B indicates that larger benchmark buffersizes tend to overwhelm the L2D cache before $b = 2^{20}\text{B}$. This corresponds to both the stepest part of the bandwidth decline, CPI increase, and the (later-discussed) increase in BUS usage shown in fig.2. Overall, L2D cache exhaustion can be considered one of the primary causes of the decline in performance at large buffer sizes discussed in Sec.3.5 and Sec.3.

The final piece of the memory puzzle comes in the form of memory accesses at levels higher up than the cache in the memory hierarchy. Overall, as can be seen in fig.2, both Always-Opt and No-Opt require similar access to the memory bus throughout. Importantly, when the L2D cache performance begins to collapse at $b = 2^{18}\text{B}$, bus

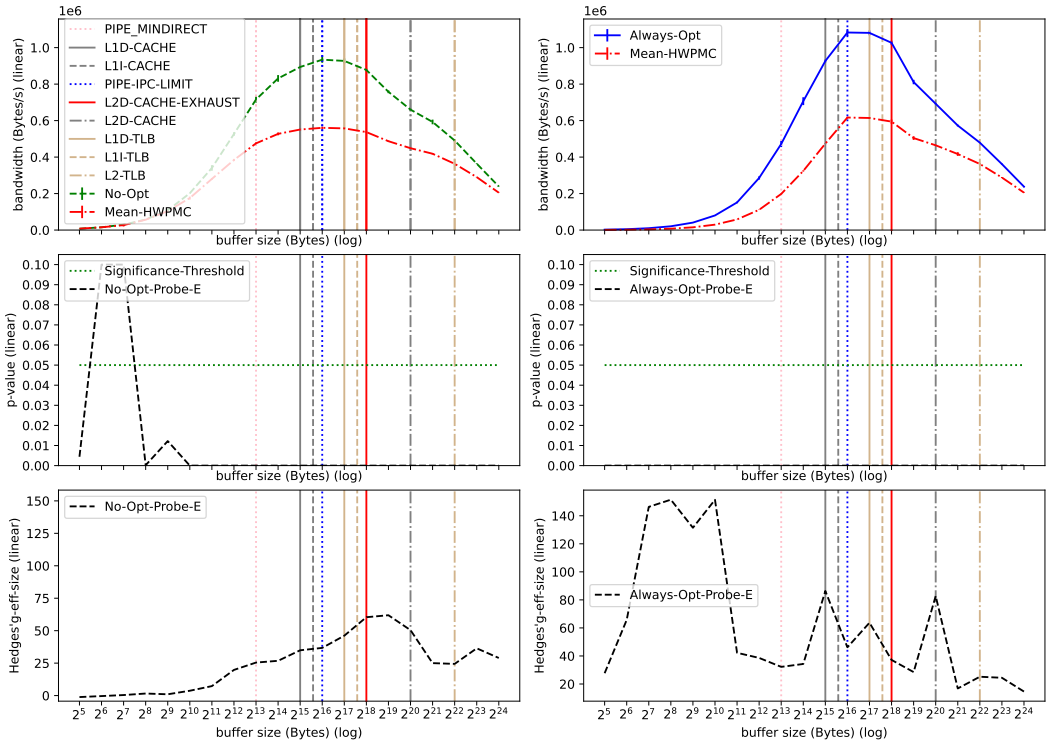


Figure 3: The non-probed Always-Opt and No-Opt behaviour vs the mean HWPMC probe effect.

access see an immediate increase together with CPI and bandwidth. All performance counters which were excluded from this analysis followed very similar trends and were deemed unlikely to offer additional information for the current report.

These hardware effects complement the previously realised report on software behavioral changes for memory copying techniques as buffer sizes increase. For small buffers, the anterior analysis revealed a dominance of trap servicing time as the data transfers were dwarfed by procedure calls and spinlocks. Addditionally, it revealed that while buffersize increases are generally correlated with improved data transfer performance up to a peak, a decline comes as the system spends more time servicing page faults and page zeroing. The system must also transition from using lightweight copying routines like copycommon to heavy memcpy. These effects are also present here and likely pay a large role in the homogenisation of Always-Opt and No-Opt performance at large buffers as overhead hides the benefits of a medium-sized copy elision per write() call.

On the basis of these results, H.2a can be accepted given that VM optimisations do help reduce the number of loads and stores sufficiently as to produce a performance improvement despite all of the advantages that caching and the organisation of the memory hierarchy in general provides for repeated copies/access of data. Although the default VM optimisation threshold proved imperfect for this benchmark, it did land close to the crossover point of $b = 2^{15}$B in fig.1 on a modern system with a benchmark that heavily favours copying compared to real-world use-cases—as it always fills the kernel pipe IPC buffer with writes to completion rather than interleaving write() and read() calls and thus causing additional context switches for No-Opt.

3.5 Measuring the HWPMC probe effect

To draw final conclusions, it must be established whether the probe effect has affected any of the findings from the previous section. For the No-Opt condition (fig.3) there isn't a strong trend for statistical significance at buffer seizes smaller than 2^8, potentially because system-call and context switch overhead dominate the run-time in a manner which is not strongly reflected by the HWPMC or because the previously mentioned small sample size is insufficient to detect it. Consequently, the null hypothesis that there is no probe effect for HWPMC counters when not using VM optimisations cannot be rejected for small buffers. For larger buffer sizes than 2^8 No-Opt consistently show a statistically significant probe effect. As such, the effect size can also be meaningfully discussed for larger buffers. The non-probed No-Opt mean being 50 standard deviations higher than the mean of the probed condition

would correspond to it being at least in the top 0.0004 of values for the probability distribution of bandwidths at that given buffer size—by Chebyshev's inequality. This is far beyond what would be considered a "large effect" in any field, as is the case for the effect size seen at every buffer size on the graph.

For the Always-Opt condition (fig.3), the probe effect always passes the statistical significance threshold. Consequently, the effect size is always meaningful to discuss and it confirms the visual intuition that the HWPMC probe effect is much more impactful for Always-Opt than No-Opt, at least until a buffer size larger than $2^{20}\mathrm{B}$ is reached. Additionally, its graph has a more meaningful shape as it shows how the probe effect is remarkably high for small buffers in particular, potentially because of heightened system activity caused by the repeated system calls, context switches and pipe IPC data transfer causing more architectural and micro-architectural events where the probe effect can significantly impact results(supported by the very large initial instruction retired ratio between Always-Opt and No-Opt discussed in sec.3.4). The decay in effect size at large buffer sizes for both conditions follows from main-memory transfer becoming the main bottleneck as it is beyond the scope of CPU HWPMC—besides simply counting how many bus accesses are done which should have an insignificant cost.

This analysis makes it clear that the probe effect is present and potentially very large in magnitude. As such, the most direct interpretation of being "negligible" would lead to the outright rejection of H.3. However, while the bandwidth difference are indeed large, the probed Always-Opt and No-Opt performance in fig.3 follow the same overall behaviours and trends as the non-probed versions. To be precise, the same initial performance gap between No-Opt and Always-Opt is present and the crossover point is still in the neighbourhood of the maximum pipe IPC size. Furthermore, the same clear decline happens once the L2D cache becomes overwhelmed. All of this being taken into account, H.3 is quantitatively rejected but holds true in the sense that the probe effect is unlikely to have impacted any conclusions drawn on the basis of the HWPMC.

4 Conclusion

This investigation started with the goal of understanding, in the form of a case study on the pipe IPC implementation in FreeBSD, the IPC overhead caused by process isolation and how the OS and system may work to lower them. To achieve this, three hypotheses were constructed and tested with the following outcomes.

1. *Larger pipe buffer sizes improve IPC performance.*
   This hypothesis was rejected in its strongest form since the performance benefits of VM-optimisation peak at a buffersize equal to the maximum pipe IPC data transfer of $64\mathrm{KiB} = 2^{16}\mathrm{B}$ and then plateau and finally decline. A lighter form of the hypothesis stating that larger buffer sizes tend to be associated with better IPC performance until the fast levels of the memory hierarchy become unable to service them can be accepted.

2. *Page-borrowing virtual-memory optimisations:*

   (a) *Are well tuned to current microarchitectures.*
   This sub-hypothesis can be accepted as VM optimisations do succeed in lowering loads and stores to cause a noticeable and statistically significant improvement in benchmark performance for buffersizes larger than $2^{15}\mathrm{B}$. This is despite copying being advantaged by both benchmark structure and caching.

   (b) *Always achieve a performance improvement, when enabled at or above the default 8KiB threshold, over the unoptimised baseline.*
   As written, this hypothesis must be rejected since a performance improvement is not seen for all buffersizes equal to or larger than the default threshold. However, the imperfect timing of the default $8\mathrm{KiB}$ threshold is not cause for concern as the benchmark is unrepresentative of realistic workloads.

3. *The probe effect associated with using HWPMC to explore this workload is negligible.*
   This hypothesis is quantitatively rejected as experiments show that the probe effect of HWPMC is very large. However, the behaviour of the benchmark follows the same overall trends as without HWPMC enabled. Thus the probe effect is unlikely to affect the conclusions of this analysis.

Several shortcomings of current work and future research avenues remain. First, more samples are needed for this kind of investigation in order to avoid noise interfering with the statistical significance of results (Sec.3.5). Second, attempting an intervention to modify the kernel max pipe IPC buffer and data transfer size would reveal if the current limit is the primary cause of the performance peak or if the cache exhaustion would have caused a decline too soon for it to matter—it could also show if VM optimisations are particularly held back by the limit. Third, tracing the HWPMC behaviour to specific parts of the kernel would likely be quite revealing.

References

Bryan Cantrill, Michael W Shapiro, Adam H Leventhal, et al. Dynamic instrumentation of production systems. In *USENIX Annual Technical Conference, General Track*, pages 15–28, 2004.

Ulrich Drepper. What every programmer should know about memory. 01 2007.

Dirk Enzmann. Notes on effect size measures for the difference of means from two independent groups: The case of cohen's d and hedges' g, 01 2015.

R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley New York, 1991.

Marshall Kirk McKusick, George V Neville-Neil, and Robert NM Watson. *The design and implementation of the FreeBSD operating system*. Pearson Education, 2014.

J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts, 10th Edition*. Wiley, 2018. ISBN 978-1-118-06333-0. URL `http://os-book.com/OS10/index.html`.

Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

Rand Wilcox. A robust nonparametric measure of effect size based on an analog of cohen's d, plus inferences about the median of the typical difference. *Journal of Modern Applied Statistical Methods*, 17(2):1, 2019.

A Appendix

A.1 Syscall memory

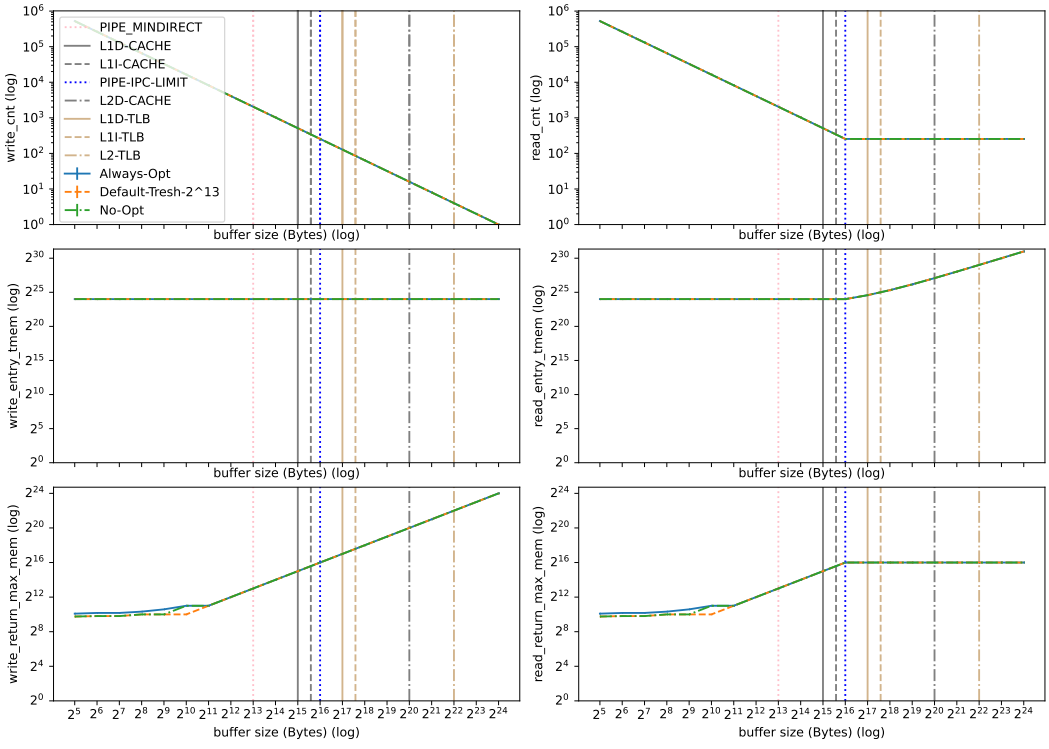


Figure 4: Aggregate metrics for write and read calls and memory writes/requests as well as their return values.

From fig.4 we can indeed see that write() always returns as much memory as it requests—max returned memory and total requested memory are identical—while read() plateaus at 2^{16}B, the maximum pipe IPC data transfer. Read attempts to return the entire requested size each time and must make $2^{24}/2^{16}$ calls after the plateau—thus

accumulating more total requested memory than the benchmark actually contains. The DTrace script tracing the exact read return values is not shown due to appendix space constraint as read() behaviour can already be inferred from this graph.

A.2 Syscall time quantisation

Time quantisation is more difficult to show due to space constraints. As such, only quantisations for the smallest buffer size and for the pipe IPC max data transfer are shown to emphasises the different behaviour of No-Opt and Always-Opt as caused by No-Opt only emptying its kernel buffer when its full while Always-Opt causes a data transfer for each write. Fig.5 and fig.6 compare No-Opt to Always-Opt for the smallest buffersize $b = 2^5\mathrm{B}$ while fig.7 and fig.8 do the same for $b = 2^{16}\mathrm{B}$. They clearly show initial clustering for No-Opt time into a group of very fast write() calls with slow write() calls happening when the kernel pipe IPC buffer is full. For always-opt little to no clustering takes place. The behaviour homogenises once the max pipe IPC data transfer is reached.

```
write_qtime
           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  517636
            4096 |                                         4683
            8192 |                                         1458
           16384 |                                         231
           32768 |                                         1
           65536 |                                         1
          131072 |                                         1
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         0
         2097152 |                                         0
         4194304 |                                         0
         8388608 |                                         0
        16777216 |                                         277
        33554432 |                                         0

read_qtime
           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  517482
            4096 |                                         4990
            8192 |                                         1553
           16384 |                                         249
           32768 |                                         0
           65536 |                                         0
          131072 |                                         0
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         0
         2097152 |                                         0
         4194304 |                                         0
         8388608 |                                         0
        16777216 |                                         13
        33554432 |                                         0
        67108864 |                                         0
       134217728 |                                         0
       268435456 |                                         0
       536870912 |                                         0
      1073741824 |                                         0
      2147483648 |                                         0
      4294967296 |                                         0
      8589934592 |                                         0
     17179869184 |                                         0
     34359738368 |                                         0
     68719476736 |                                         0
    137438953472 |                                         1
    274877906944 |                                         0
```

Figure 5: No-Opt read and write pow2-quantisation for a buffer size of $2^5\mathrm{B}$

```
write_qtime
           value  ------------- Distribution ------------- count
            8192 |                                         0
           16384 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  511018
           32768 |@                                        13257
           65536 |                                         3
          131072 |                                         0
          262144 |                                         9
          524288 |                                         0

read_qtime
           value  ------------- Distribution ------------- count
            2048 |                                         0
            4096 |@@@@@@@@@@@@@@@                          182866
            8192 |                                         137
           16384 |@@@@@@@@@@@@@@@@@@@@@@@@@@@              332314
           32768 |@                                        8961
           65536 |                                         2
          131072 |                                         0
          262144 |                                         7
          524288 |                                         0
         1048576 |                                         0
         2097152 |                                         0
         4194304 |                                         0
         8388608 |                                         0
        16777216 |                                         0
        33554432 |                                         0
        67108864 |                                         0
       134217728 |                                         0
       268435456 |                                         0
       536870912 |                                         0
      1073741824 |                                         0
      2147483648 |                                         0
      4294967296 |                                         0
      8589934592 |                                         0
     17179869184 |                                         0
     34359738368 |                                         0
     68719476736 |                                         1
    137438953472 |                                         0
```

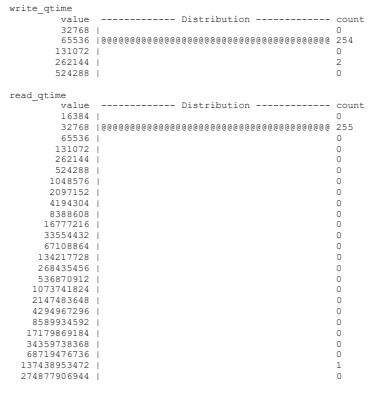
Figure 6: Always-Opt read and write pow2-quantisation for a buffer size of $2^5\mathrm{B}$

A.3 Write and read sleep and wake behaviour

Sleep and wake behaviour are presented similarly to the time quantisations with direct DTrace output. Fig.9 and fig.10 compare No-Opt to Always-Opt for the smallest buffersize $b = 2^5\mathrm{B}$ while fig.11 and fig.12 do the same for $b = 2^{16}\mathrm{B}$. The sender and receiver are identified by the pid and tid, shown after the write and read counts above right before the syscall entry/return cnt. For No-Opt, the sender begins by having to go to sleep only when the kernel pipe IPC buffer is full, when the max pipe IPC transfer is reached it must go to sleep for every transfer. Under Always-Opt the sender always goes to sleep as it does not have a kernel pipe IPC buffer. The results for the sender and receiver going on and off-cpu are similar—and also reflect context switches—and not shown to save space.

A.4 Chebyshev's inequality

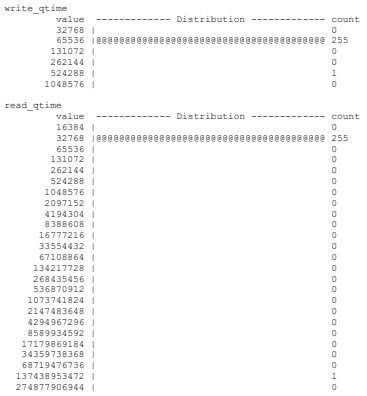
For most probability distributions Chebyshev's inequality states that at most $1/k^2$ of the values will be more than k standard deviations away from the mean.



```
write_qtime
           value  ------------- Distribution ------------- count
            32768 |                                       0
            65536 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 254
           131072 |                                       0
           262144 |                                       2
           524288 |                                       0

read_qtime
           value  ------------- Distribution ------------- count
            16384 |                                       0
            32768 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 255
            65536 |                                       0
           131072 |                                       0
           262144 |                                       0
           524288 |                                       0
          1048576 |                                       0
          2097152 |                                       0
          4194304 |                                       0
          8388608 |                                       0
         16777216 |                                       0
         33554432 |                                       0
         67108864 |                                       0
        134217728 |                                       0
        268435456 |                                       0
        536870912 |                                       0
       1073741824 |                                       0
       2147483648 |                                       0
       4294967296 |                                       0
       8589934592 |                                       0
      17179869184 |                                       0
      34359738368 |                                       0
      68719476736 |                                       0
     137438953472 |                                       1
     274877906944 |                                       0
```

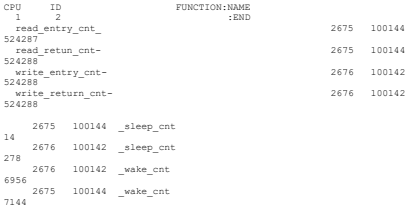
Figure 7: No-Opt read and write pow2-quantisation for a buffer size of 2^{16}B



```
write_qtime
           value  ------------- Distribution ------------- count
            32768 |                                       0
            65536 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 255
           131072 |                                       0
           262144 |                                       0
           524288 |                                       1
          1048576 |                                       0

read_qtime
           value  ------------- Distribution ------------- count
            16384 |                                       0
            32768 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 255
            65536 |                                       0
           131072 |                                       0
           262144 |                                       0
           524288 |                                       0
          1048576 |                                       0
          2097152 |                                       0
          4194304 |                                       0
          8388608 |                                       0
         16777216 |                                       0
         33554432 |                                       0
         67108864 |                                       0
        134217728 |                                       0
        268435456 |                                       0
        536870912 |                                       0
       1073741824 |                                       0
       2147483648 |                                       0
       4294967296 |                                       0
       8589934592 |                                       0
      17179869184 |                                       0
      34359738368 |                                       0
      68719476736 |                                       0
     137438953472 |                                       1
     274877906944 |                                       0
```

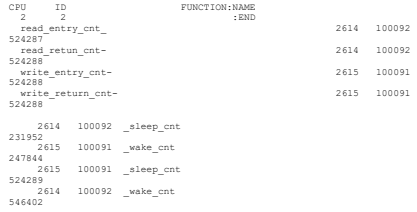
Figure 8: Always-Opt read and write pow2-quantisation for a buffer size of 2^{16}B



```
CPU     ID                    FUNCTION:NAME
  1      2                     :END
  read_entry_cnt_                                  2675   100144
524287
  read_retun_cnt-                                  2675   100144
524288
  write_entry_cnt-                                 2676   100142
524288
  write_return_cnt-                                2676   100142
524288

     2675   100144  _sleep_cnt
14
     2676   100142  _sleep_cnt
278
     2676   100142  _wake_cnt
6956
     2675   100144  _wake_cnt
7144
```

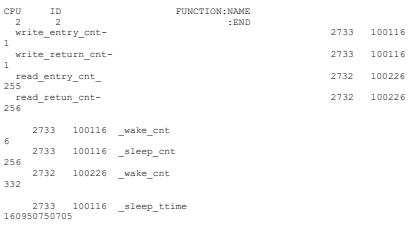
Figure 9: No-Opt sender and receiver thread sleep and wake behavior for a buffersize of 2^5B.



```
CPU     ID                    FUNCTION:NAME
  2      2                     :END
  read_entry_cnt_                                  2614   100092
524287
  read_retun_cnt-                                  2614   100092
524288
  write_entry_cnt-                                 2615   100091
524288
  write_return_cnt-                                2615   100091
524288

     2614   100092  _sleep_cnt
231952
     2615   100091  _wake_cnt
247844
     2615   100091  _sleep_cnt
524289
     2614   100092  _wake_cnt
546402
```

Figure 10: Always-Opt sender and receiver thread sleep and wake behavior for a buffersize of 2^5B.

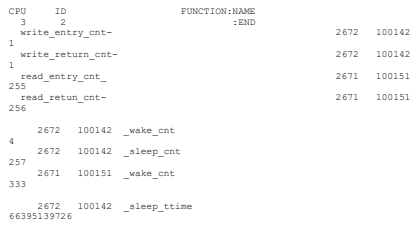


```
CPU     ID                    FUNCTION:NAME
  2      2                     :END
  write_entry_cnt-                                 2733   100116
1
  write_return_cnt-                                2733   100116
1
  read_entry_cnt_                                  2732   100226
255
  read_retun_cnt-                                  2732   100226
256

     2733   100116  _wake_cnt
6
     2733   100116  _sleep_cnt
256
     2732   100226  _wake_cnt
332

     2733   100116  _sleep_ttime
160950750705
```

Figure 11: No-Opt sender and receiver thread sleep and wake behavior for a buffersize of 2^{16}B.



```
CPU     ID                    FUNCTION:NAME
  3      2                     :END
  write_entry_cnt-                                 2672   100142
1
  write_return_cnt-                                2672   100142
1
  read_entry_cnt_                                  2671   100151
255
  read_retun_cnt-                                  2671   100151
256

     2672   100142  _wake_cnt
4
     2672   100142  _sleep_cnt
257
     2671   100151  _wake_cnt
333

     2672   100142  _sleep_ttime
66395139726
```

Figure 12: Always-opt sender and receiver thread sleep and wake behavior for a buffersize of 2^{16}B.