Troca de Contexto no Kernel Linux - Estudo de Arquiteturas

Lucas Reis
Jonathas Silveira

Tópicos em Sistemas Operacionais

Agenda

- Relembrando...
- Metodologia
- Problemas de Desenvolvimento
- Estudo do Kernel
- Resultados
  - Experimentos
  - Scripts
- Conclusão

Relembrando...

- Estudo de troca de contexto do kernel Linux em três arquiteturas
  - ARM (Raspberry Pi 3)
  - RISC-V (QEMU)
  - x86_64 (i7 - 16Gb)
- Estudo do Kernel Linux para análise
- Geração de benchmarks e comparação de resultados

Metodologia

- Leitura do código do kernel para troca de contexto
- Benchmarks de análise de desempenho de troca de contexto
- Validação dos resultados pela comparação do estudo

Metodologia - Hardware

- X86
  - Intel i7 4770
  - Ubuntu 18 LTE
  - Kernel 4.15
- ARM
  - Raspberry Pi 3 Model b - Cortex A53
  - Ubuntu Server 18 LTE
  - Kernel 4.15
- RISC-V
  - QEMU
  - Berkley bootloader, Busybear Root Filesystem
  - Kernel 4.18 e 5.4

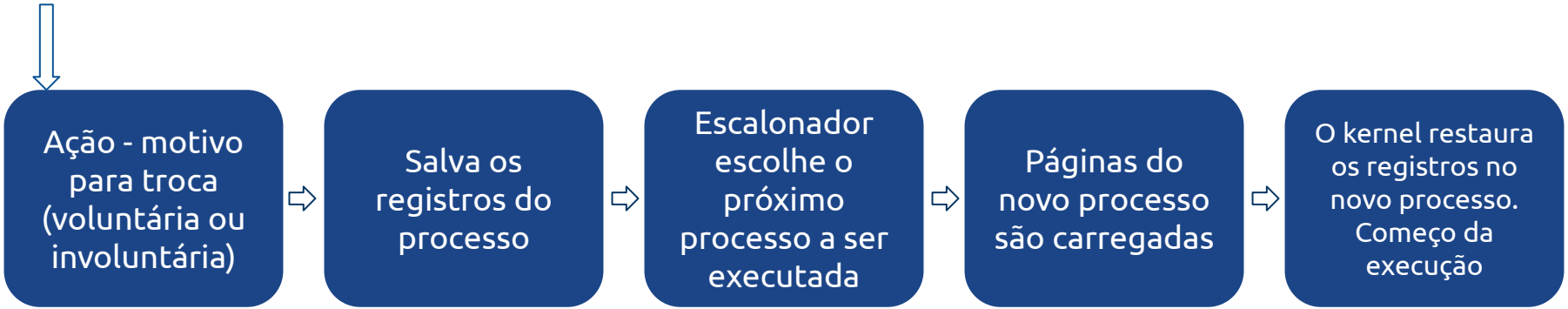
Metodologia - Benchmarks

- LMBench3
  - Troca de contexto entre diversos processos, variando quantidade e tamanho dos processos
- Eli Bendersky Benchmarks
  - Troca de variáveis entre processos
  - Ping-Pong usando Pipe

Problemas no Desenvolvimento

- Alta complexidade de ambientes de simulação (rv8 e QEMU)
  - Documentação escassa
  - Código desatualizado
  - Root filesystem limitado (busybear)
  - Problemas na configuração da interface de rede
- Dificuldades para mensurar troca de contexto

Problemas de mensurar a troca

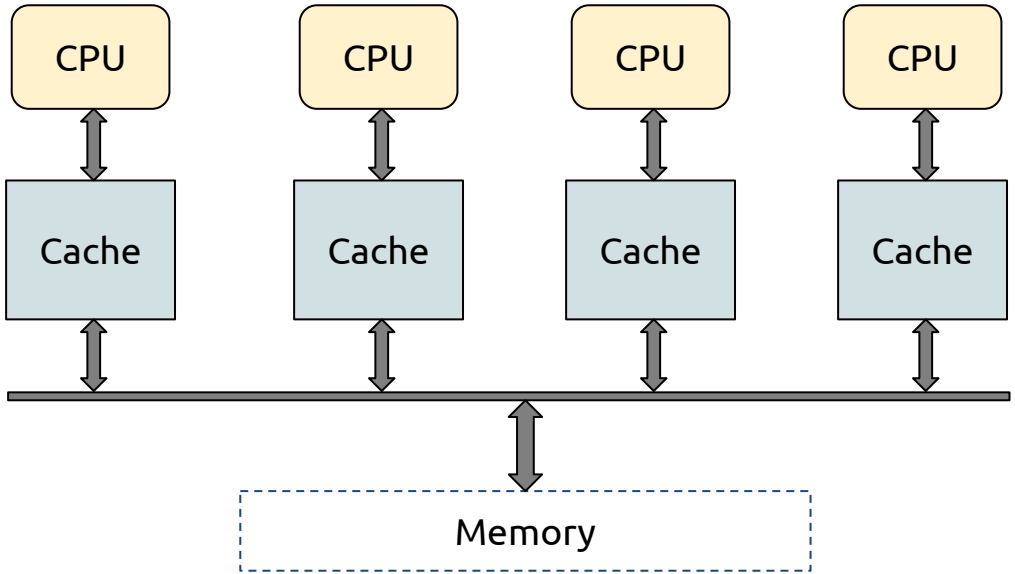


Problemas de mensurar a troca

1. Dados desnecessários do processo antigo substituídos gradativamente pelos dados do outro processo

2. Escalonando em um sistema multicore

R
- Fixar um processador
- Processos que trocam dados via pipe
- Processos que possuem uma variável em comum



Eli Bendersky - https://eli.thegreenplace.net/2018/measuring-context-switching-and-memory-overheads-for-linux-threads/

Estudo do Kernel

- Trocas de contexto divididas em duas etapas
  - context_switch() - Independente de arquitetura
  - **switch_to() - Dependente de arquitetura**

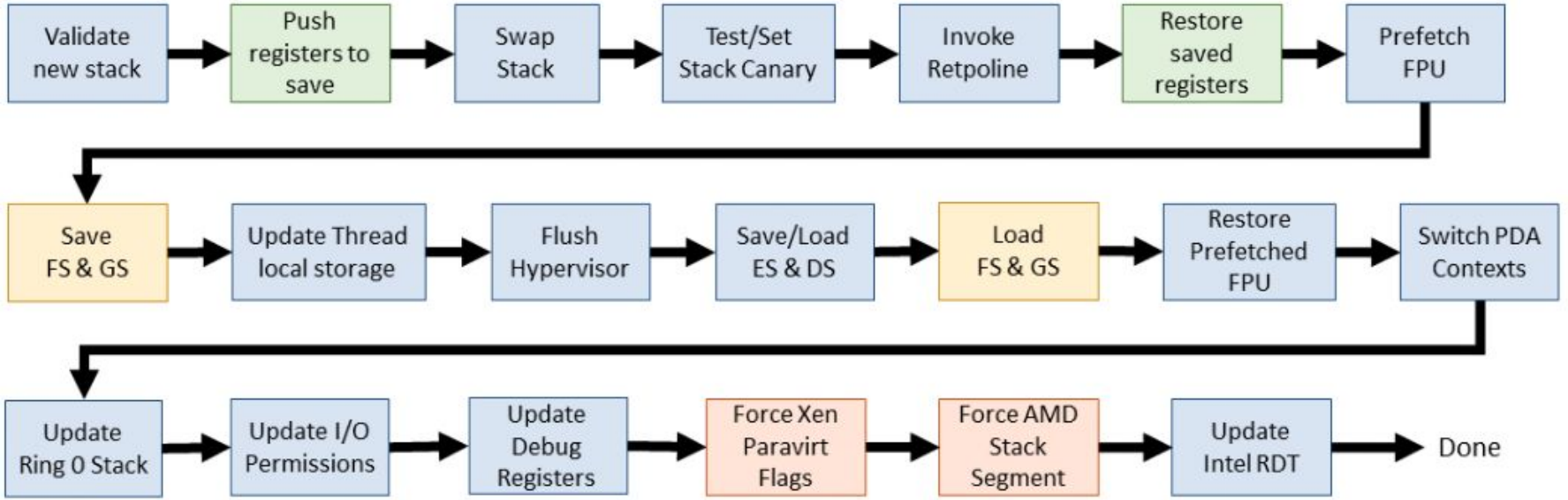
```
/*
 * context_switch - switch to the new MM and the new thread's register state.
 */
static __always_inline struct rq *
context_switch(struct rq *rq, struct task_struct *prev,
               struct task_struct *next, struct rq_flags *rf)
{
          struct mm_struct *mm, *oldmm;

          prepare_task_switch(rq, prev, next);

          mm = next->mm;
          oldmm = prev->active_mm;

          arch_start_context_switch(prev);

          if (!mm) {
                    next->active_mm = oldmm;
                    mmgrab(oldmm);
                    enter_lazy_tlb(oldmm, next);
          } else
                    switch_mm_irqs_off(oldmm, mm, next);

          if (!prev->mm) {
                    prev->active_mm = NULL;
                    rq->prev_mm = oldmm;
          }

          rq->clock_update_flags &= ~(RQCF_ACT_SKIP|RQCF_REQ_SKIP);

          prepare_lock_switch(rq, next, rf);

          /* Here we just switch the register state and the stack. */
          switch_to(prev, next, prev);
          barrier();

          return finish_task_switch(prev);
}
```

x86

```
#define switch_to(prev, next, last)
                  \
do {
            \
      prepare_switch_to(next);
                  \
                                                \
      ((last) = __switch_to_asm((prev),
(next)));               \
} while (0)
```

```
/*
 * %rdi: prev task
 * %rsi: next task
 */
ENTRY(__switch_to_asm)
        UNWIND_HINT_FUNC
        /*
         * Save callee-saved registers
         * This must match the order in inactive_task_frame
         */
        pushq   %rbp
        pushq   %rbx
        pushq   %r12
        pushq   %r13
        pushq   %r14
        pushq   %r15

        /* switch stack */
        movq    %rsp, TASK_threadsp(%rdi)
        movq    TASK_threadsp(%rsi), %rsp

#ifdef CONFIG_CC_STACKPROTECTOR
        movq    TASK_stack_canary(%rsi), %rbx
        movq    %rbx, PER_CPU_VAR(irq_stack_union)+stack_canary_offset
#endif
```

```
#ifdef CONFIG_RETPOLINE
        /*
         * When switching from a shallower to a deeper call stack
         * the RSB may either underflow or use entries populated
         * with userspace addresses. On CPUs where those concerns
         * exist, overwrite the RSB with entries which capture
         * speculative execution to prevent attack.
         */
        FILL_RETURN_BUFFER %r12, RSB_CLEAR_LOOPS, X86_FEATURE_RSB_CTXSW
#endif

        /* restore callee-saved registers */
        popq    %r15
        popq    %r14
        popq    %r13
        popq    %r12
        popq    %rbx
        popq    %rbp

        jmp     __switch_to
END(__switch_to_asm)
```


Validate new stack
Push registers to save
Swap Stack
Test/Set Stack Canary
Invoke Retpoline
Restore saved registers
Prefetch FPU
Save FS & GS
Update Thread local storage
Flush Hypervisor
Save/Load ES & DS
Load FS & GS
Restore Prefetched FPU
Switch PDA Contexts
Update Ring 0 Stack
Update I/O Permissions
Update Debug Registers
Force Xen Paravirt Flags
Force AMD Stack Segment
Update Intel RDT
Done

ARM

```
#define switch_to(prev,next,last)                                       \
do {                                                                    \
      __complete_pending_tlbi();                                        \
      last = __switch_to(prev,task_thread_info(prev), task_thread_info(next));      \
} while (0)
```

```
__notrace_funcgraph struct task_struct *__switch_to(struct task_struct *prev,
                                struct task_struct *next)
{
        struct task_struct *last;

        fpsimd_thread_switch(next); //contexto de instruções SIMD
        tls_thread_switch(next); /thread local storage
        hw_breakpoint_thread_switch(next); //breakpoints para debugging
        contextidr_thread_switch(next); //registrador de controle de identificador de thread (debugging)
        entry_task_switch(next); //restaura copia do SP
        uao_thread_switch(next); //user access override state (debugging)

        /*
         * Complete any pending TLB or cache maintenance on this CPU in case
         * the thread migrates to a different CPU.
         * This full barrier is also required by the membarrier system
         * call.
         */
        dsb(ish);

        /* the actual thread switch */
        last = cpu_switch_to(prev, next);

        return last;
}
```

```
ENTRY(cpu_switch_to)
	mov	x10, #THREAD_CPU_CONTEXT
	add	x8, x0, x10
	mov	x9, sp
	stp	x19, x20, [x8], #16		// store callee-saved registers
	stp	x21, x22, [x8], #16
	stp	x23, x24, [x8], #16
	stp	x25, x26, [x8], #16
	stp	x27, x28, [x8], #16
	stp	x29, x9, [x8], #16
	str	lr, [x8]
	add	x8, x1, x10
	ldp	x19, x20, [x8], #16		// restore callee-saved registers
	ldp	x21, x22, [x8], #16
	ldp	x23, x24, [x8], #16
	ldp	x25, x26, [x8], #16
	ldp	x27, x28, [x8], #16
	ldp	x29, x9, [x8], #16
	ldr	lr, [x8]
	mov	sp, x9
	msr	sp_el0, x1
	ret
ENDPROC(cpu_switch_to)
```

RISC-V - Kernel 4.19

```
#define switch_to(prev, next, last)                 \
do {                                                \
     struct task_struct *__prev = (prev);           \
     struct task_struct *__next = (next);           \
     __switch_to_aux(__prev, __next);               \
     ((last) = __switch_to(__prev, __next));        \
} while (0)
```

```
static inline void __switch_to_aux(struct task_struct *prev,
                                   struct task_struct *next)
{
      struct pt_regs *regs;

      regs = task_pt_regs(prev);
      if (unlikely(regs->sstatus & SR_SD))
            fstate_save(prev, regs);
      fstate_restore(next, task_pt_regs(next));
}
```

```
ENTRY(__switch_to)
        /* Save context into prev->thread */
        li    a4,  TASK_THREAD_RA
        add   a3, a0, a4
        add   a4, a1, a4
        REG_S ra,  TASK_THREAD_RA_RA(a3)
        REG_S sp,  TASK_THREAD_SP_RA(a3)
        REG_S s0,  TASK_THREAD_S0_RA(a3)
        REG_S s1,  TASK_THREAD_S1_RA(a3)
        REG_S s2,  TASK_THREAD_S2_RA(a3)
        REG_S s3,  TASK_THREAD_S3_RA(a3)
        REG_S s4,  TASK_THREAD_S4_RA(a3)
        REG_S s5,  TASK_THREAD_S5_RA(a3)
        REG_S s6,  TASK_THREAD_S6_RA(a3)
        REG_S s7,  TASK_THREAD_S7_RA(a3)
        REG_S s8,  TASK_THREAD_S8_RA(a3)
        REG_S s9,  TASK_THREAD_S9_RA(a3)
        REG_S s10, TASK_THREAD_S10_RA(a3)
        REG_S s11, TASK_THREAD_S11_RA(a3)
        /* Restore context from next->thread */
        REG_L ra,  TASK_THREAD_RA_RA(a4)
        REG_L sp,  TASK_THREAD_SP_RA(a4)
        REG_L s0,  TASK_THREAD_S0_RA(a4)
        REG_L s1,  TASK_THREAD_S1_RA(a4)
        REG_L s2,  TASK_THREAD_S2_RA(a4)
        REG_L s3,  TASK_THREAD_S3_RA(a4)
        REG_L s4,  TASK_THREAD_S4_RA(a4)
        REG_L s5,  TASK_THREAD_S5_RA(a4)
        REG_L s6,  TASK_THREAD_S6_RA(a4)
        REG_L s7,  TASK_THREAD_S7_RA(a4)
        REG_L s8,  TASK_THREAD_S8_RA(a4)
        REG_L s9,  TASK_THREAD_S9_RA(a4)
        REG_L s10, TASK_THREAD_S10_RA(a4)
        REG_L s11, TASK_THREAD_S11_RA(a4)
```

```
        /* Swap the CPU entry around. */
        lw a3, TASK_TI_CPU(a0)
        lw a4, TASK_TI_CPU(a1)
        sw a3, TASK_TI_CPU(a1)
        sw a4, TASK_TI_CPU(a0)
#if TASK_TI != 0
#error "TASK_TI != 0: tp will contain a 'struct thread_info', not a 'struct task_struct' so
get_current() won't work."
        addi tp, a1, TASK_TI
#else
        move tp, a1
#endif
        ret
ENDPROC(__switch_to)
```

RISC-V - Kernel 5.4

```
#define switch_to(prev, next, last)                     \
do {                                                    \
     struct task_struct *__prev = (prev);               \
     struct task_struct *__next = (next);               \
     if (has_fpu)                                       \
          __switch_to_aux(__prev, __next);              \
     ((last) = __switch_to(__prev, __next));            \
} while (0)
```

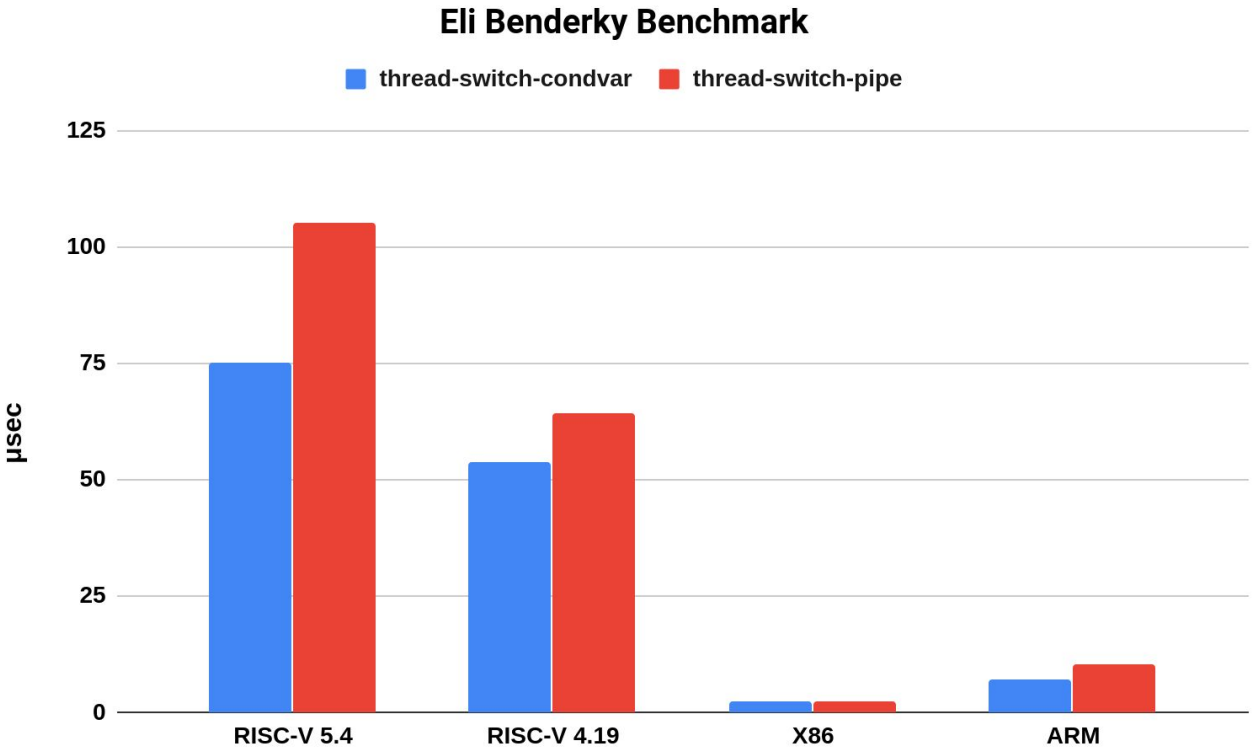
Resultados

- Experimentos
  - Eli Benderky Benchmark
  - LMBench
- Ferramentas do Linux para medir
  - Perf
  - pidstat
- Scripts
  - ctx_stats.sh
  - ctx_v0.sh

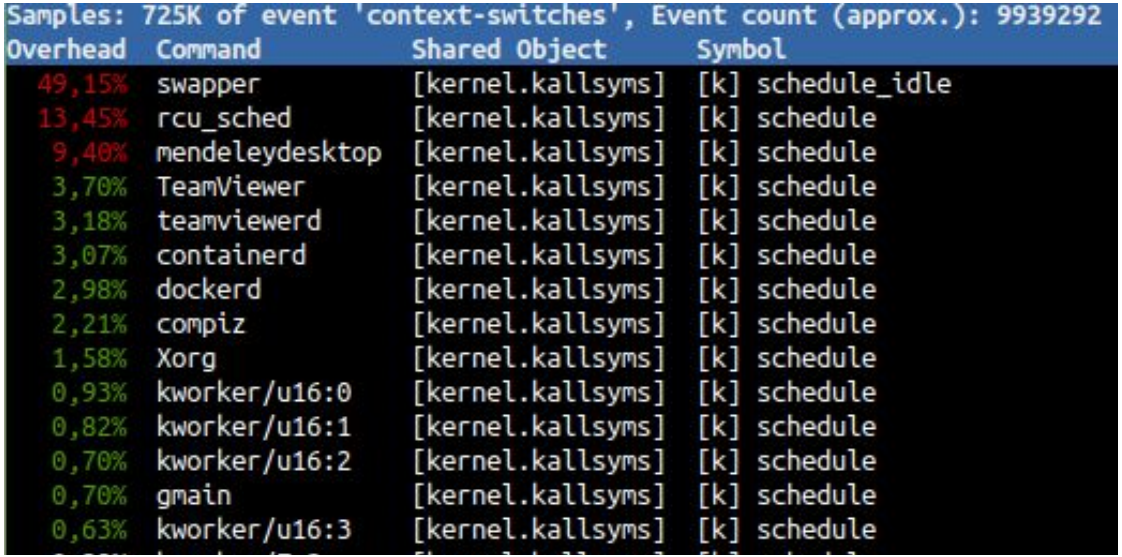
https://github.com/JEvSilv/sotopics

Resultados - Eli Benderky Benchmark

	Iterações	Switches
thread-switch-condvar	100000	200000
thread-switch-pipe	100000	200000



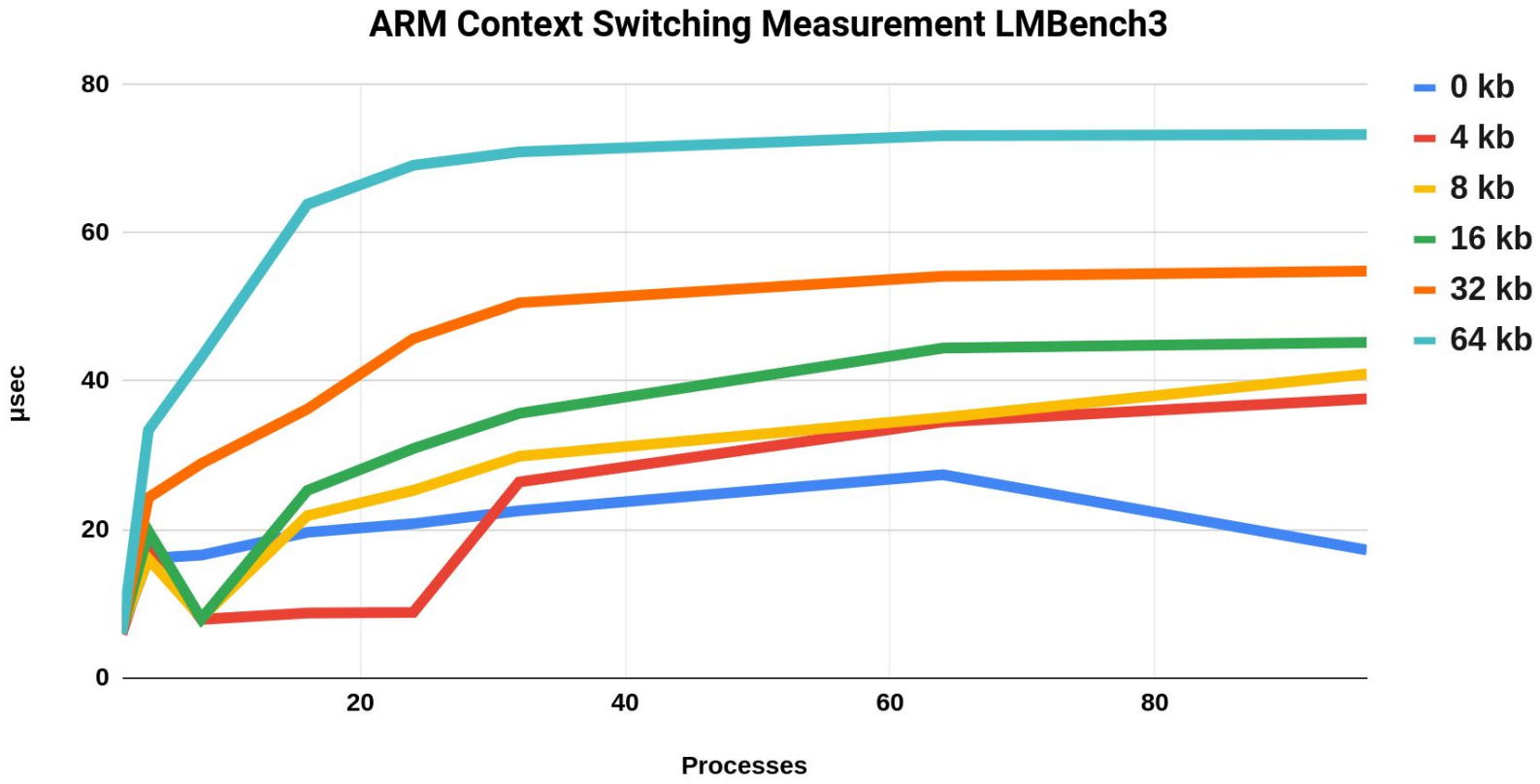
Resultados - Eli Benderky Benchmark + Projeção com Perf



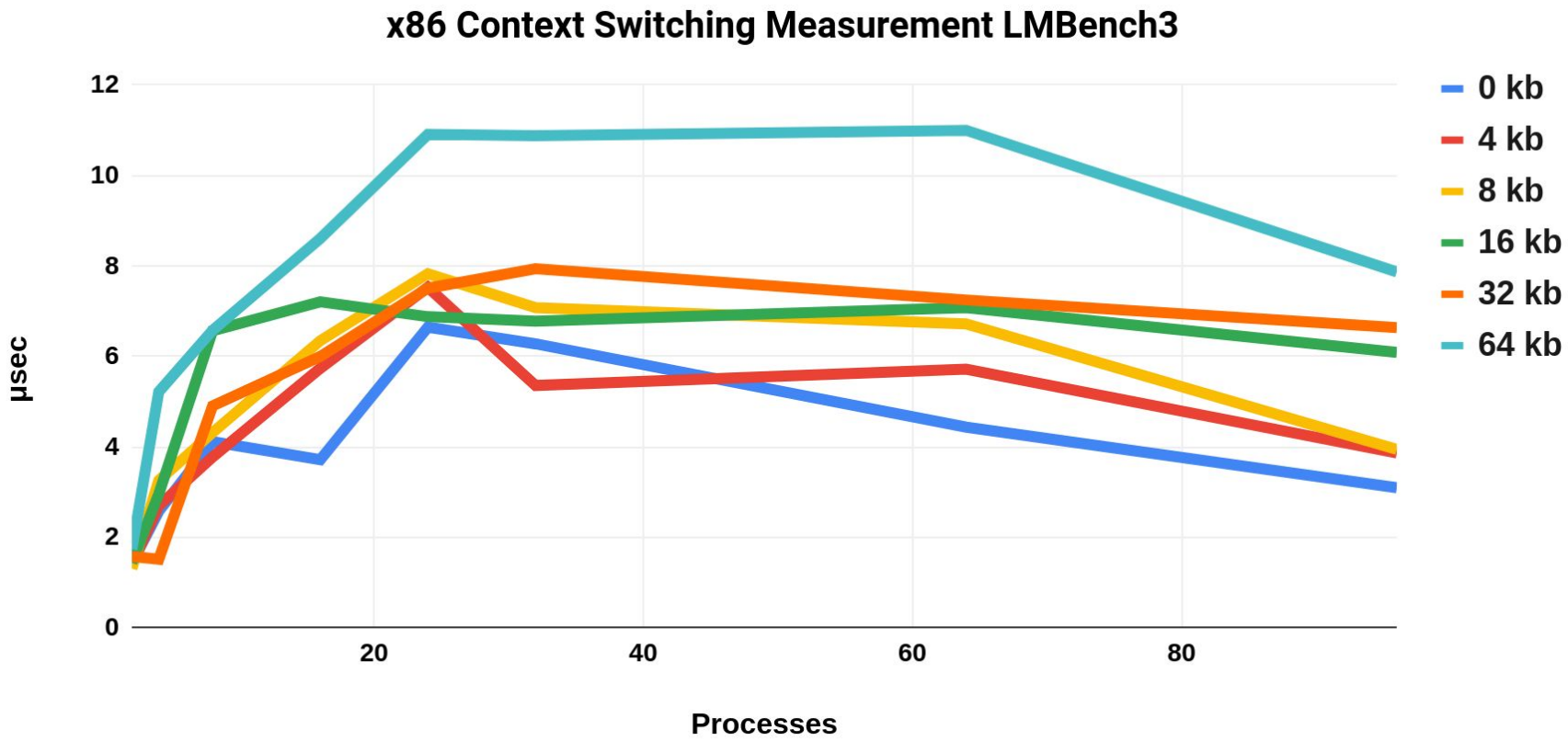
```
Samples: 725K of event 'context-switches', Event count (approx.): 9939292
Overhead  Command          Shared Object      Symbol
  49,15%  swapper          [kernel.kallsyms]  [k] schedule_idle
  13,45%  rcu_sched        [kernel.kallsyms]  [k] schedule
   9,40%  mendeleydesktop  [kernel.kallsyms]  [k] schedule
   3,70%  TeamViewer       [kernel.kallsyms]  [k] schedule
   3,18%  teamviewerd      [kernel.kallsyms]  [k] schedule
   3,07%  containerd       [kernel.kallsyms]  [k] schedule
   2,98%  dockerd          [kernel.kallsyms]  [k] schedule
   2,21%  compiz           [kernel.kallsyms]  [k] schedule
   1,58%  Xorg             [kernel.kallsyms]  [k] schedule
   0,93%  kworker/u16:0    [kernel.kallsyms]  [k] schedule
   0,82%  kworker/u16:1    [kernel.kallsyms]  [k] schedule
   0,70%  kworker/u16:2    [kernel.kallsyms]  [k] schedule
   0,70%  gmain            [kernel.kallsyms]  [k] schedule
   0,63%  kworker/u16:3    [kernel.kallsyms]  [k] schedule
```

	12h	1 dia	1 mês	1 ano
ARM	70.605 s	2.335 min	1.67 h	20,04 h
X86	23.003 s	1.1 min	30.30 min	6.06 h
RISC-V 4.19	533.248 s	8.887 min	4.443 h	2.22 dias
RISC-V 5.4	7479.665 s	4.16 h	5.2 dias	62.4 dias

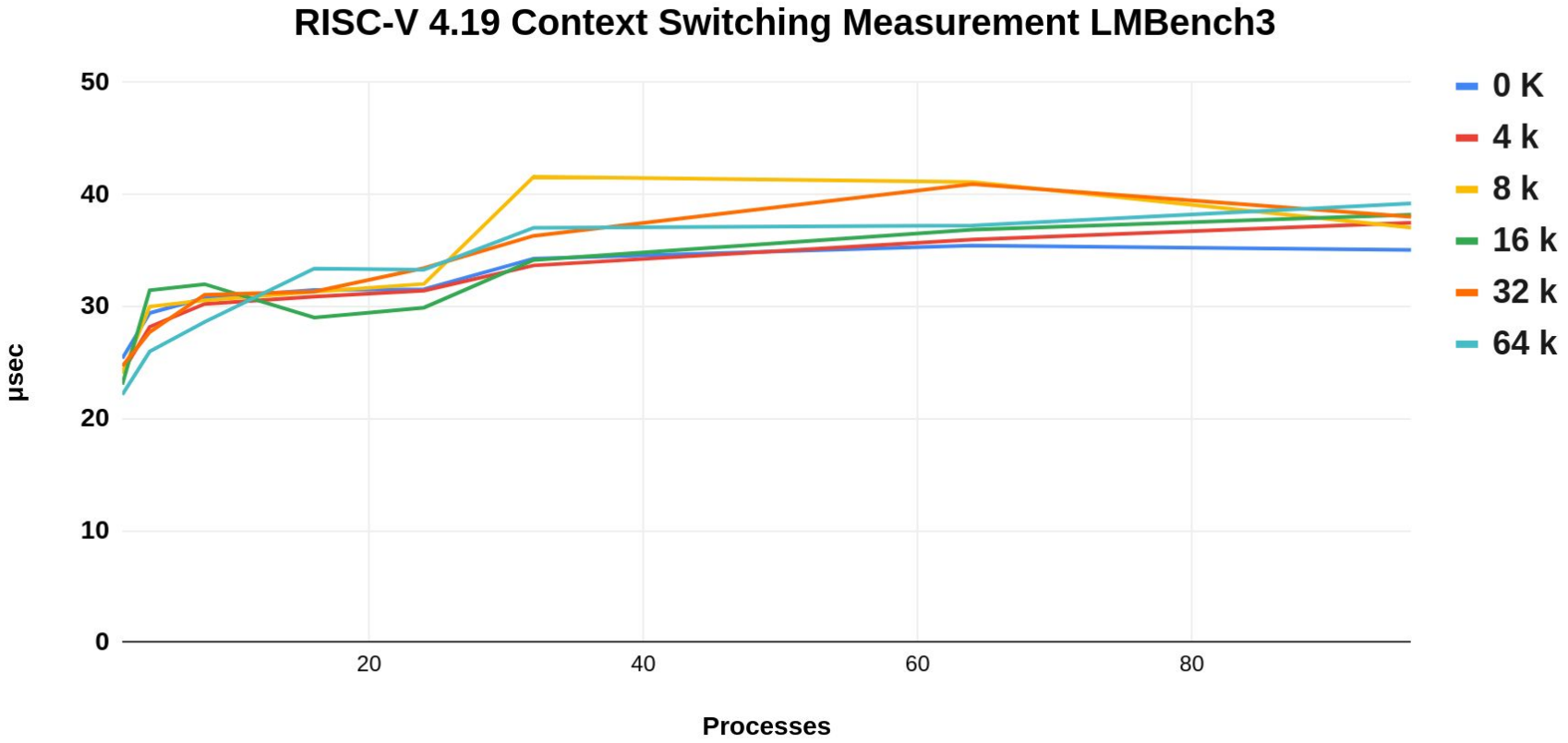
Resultados - LMBench-3.0a1



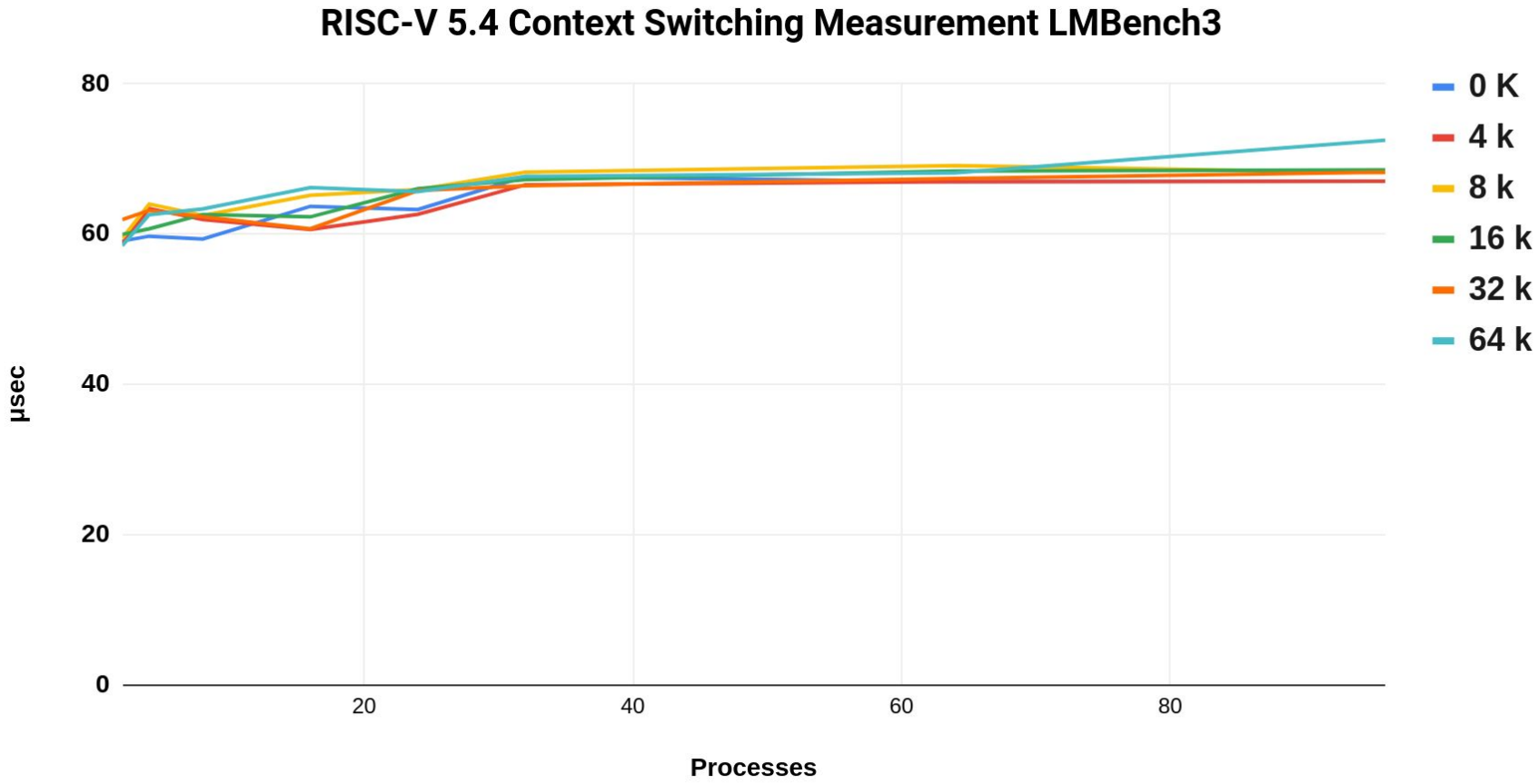
Resultados - LMBench-3.0a1



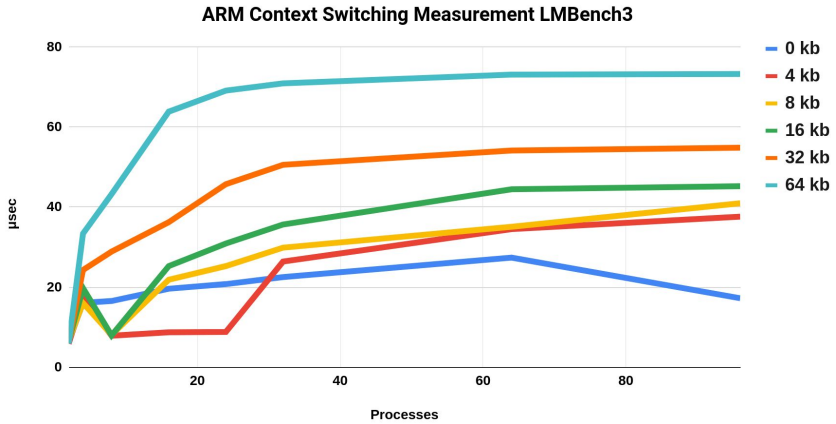
Resultados - LMBench-3.0a1

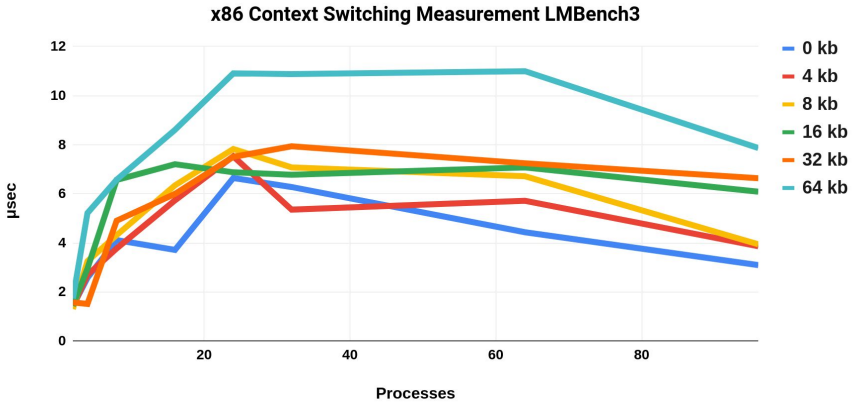


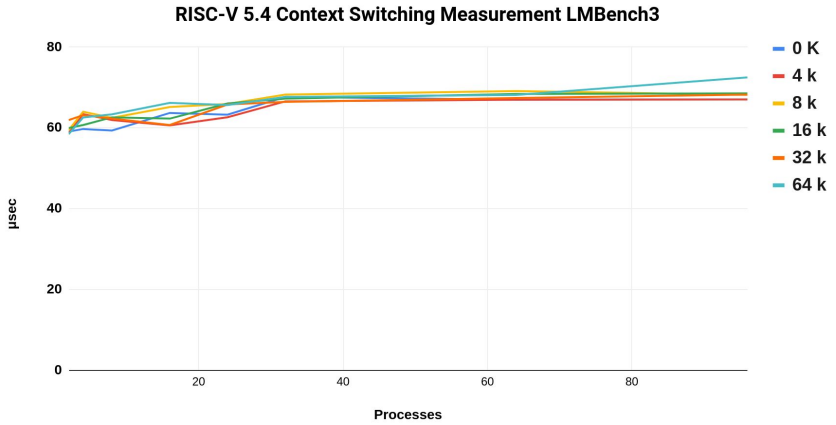
Resultados - LMBench-3.0a1

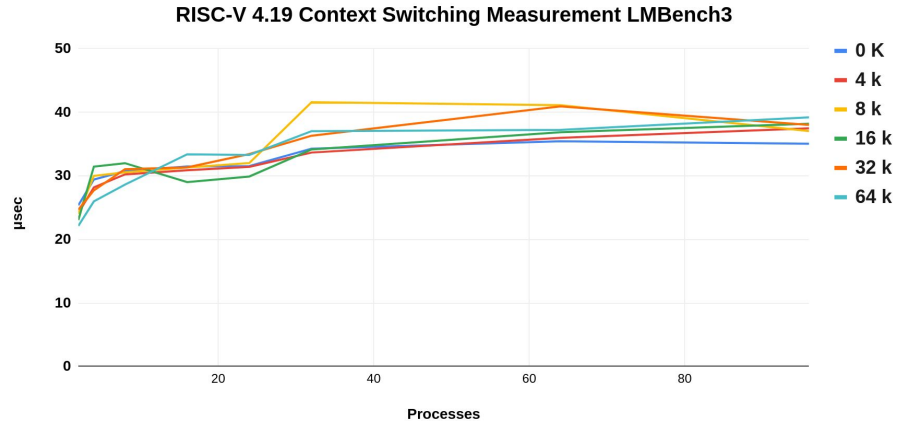


Resultados - Comparação









ctx_stats.sh

```
#!/bin/bash
`ls /proc/|cat| grep ^[0-9]*$ > pids`
NUM_PIDS=$(cat pids | wc -l)
for (( i=1; i<=$NUM_PIDS; i++))
do
   PID=$(cat pids | sed -n ${i}p)
   NAME=$(cat /proc/${PID}/status | grep Name)
   CTX=$(cat /proc/${PID}/status  | grep ctx)
   echo "----------------"
   echo $NAME
   echo $CTX
   echo "----------------"
done
rm pids
```

Conclusão

- Surpreendente velocidade do x86
- Difícil métrica de desempenho para simulação
- ARM e x86
  - Quando se transfere poucos dados entre poucos processos a latência não sofre tanto impacto
- X86
  - Melhor performance
  - Resultado surpreendente, apesar da complexidade da troca
- RISC-V: quantidade de processos não influenciaram tanto no resultado
  - Versão 4.19 chega ser melhor que ARM
  - Versão 5.4 perde para todos
    - Overhead de simulação
- Potencia do processador

Obrigado pela atenção!

Perguntas?