Per-cpu -4- (atomic operations)

2016-06-01 (http://jake.dothome.co.kr/per-cpu-atomic/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

Per-cpu -4- (atomic operations)

this_cpu_cmpxchg_double()

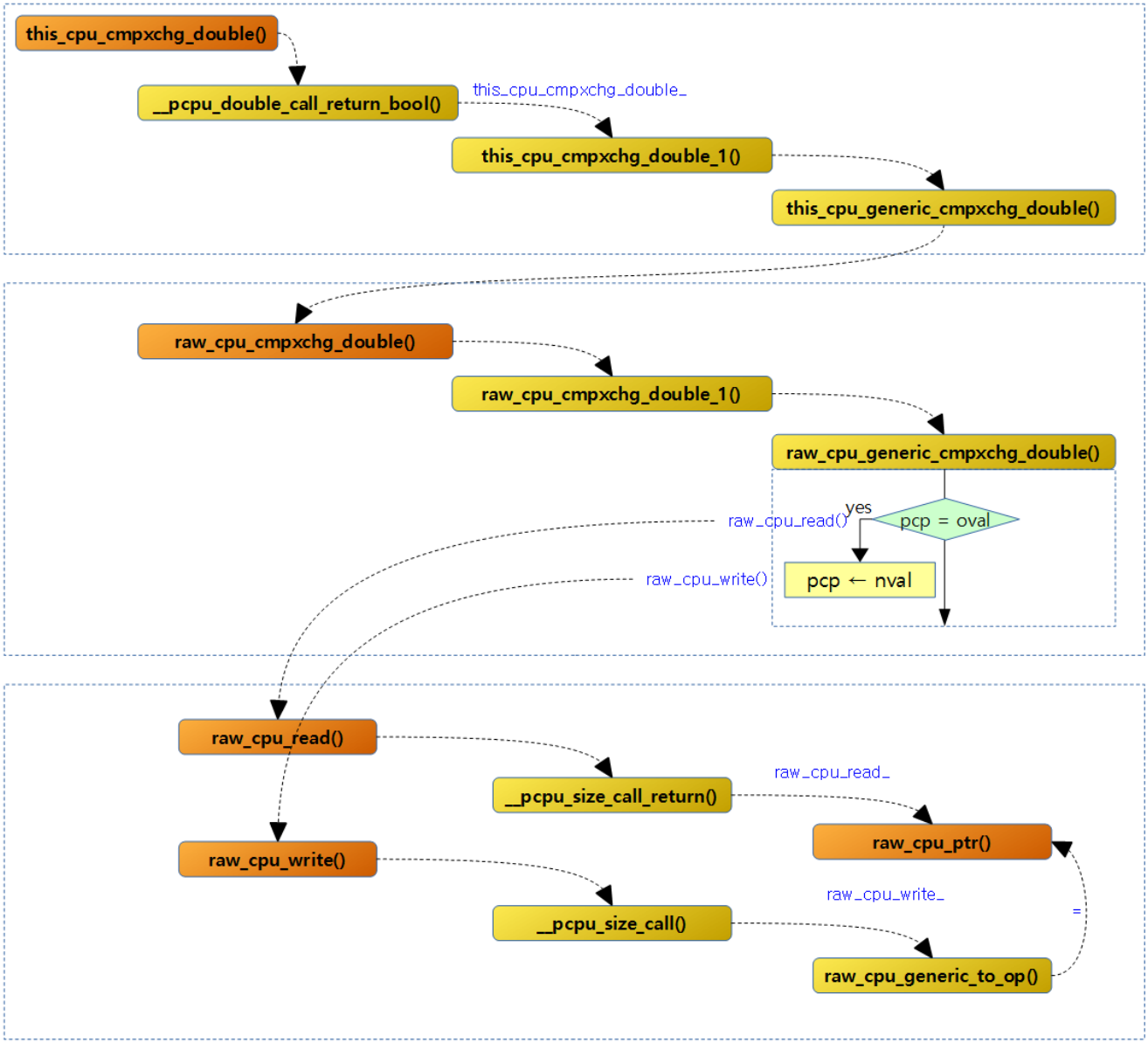
include/linux/percpu-defs.h

```
1 #define this_cpu_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1, nval2)
  \
2         __pcpu_double_call_return_bool(this_cpu_cmpxchg_double_, pcp1, p
  cp2, oval1, oval2, nval1, nval2)
```

If the per-CPU value of PCP(PCP1, PCP2) is the same as the old value (oval1, oval2), then the new value (nval1, nval2) is atomically substituted into the pcp.

- How it differs from the cmpxchg_double() function
  - Expect faster atomic operations to replace per-CPU values that don't need to compete with other CPUs.
    - In the ARM architecture, local irq is blocked only during atomic opeation operation.
    - In the arm64 architecture, preemption is prevented only during atomic operations.

The figure below shows how the this_cpu_cmpxchg_double() function is processed.



__pcpu_double_call_return_bool()

include/linux/percpu-defs.h

```
01 /*
02  * Special handling for cmpxchg_double.  cmpxchg_double is passed two
03  * percpu variables.  The first has to be aligned to a double word
04  * boundary and the second has to follow directly thereafter.
05  * We enforce this on all architectures even if they don't support
06  * a double cmpxchg instruction, since it's a cheap requirement, and it
07  * avoids breaking the requirement for architectures with the instructio
    n.
08  */
09 #define __pcpu_double_call_return_bool(stem, pcp1, pcp2, ...)
    \
10 ({
    \
11         bool pdcrb_ret__;
    \
12         __verify_pcpu_ptr(&(pcp1));
    \
13         BUILD_BUG_ON(sizeof(pcp1) != sizeof(pcp2));
    \
14         VM_BUG_ON((unsigned long)(&(pcp1)) % (2 * sizeof(pcp1)));
    \
```

```
15          VM_BUG_ON((unsigned long)(&(pcp2)) !=
\
16                    (unsigned long)(&(pcp1)) + sizeof(pcp1));
\
17          switch(sizeof(pcp1)) {
\
18          case 1: pdcrb_ret__ = stem##1(pcp1, pcp2, __VA_ARGS__); break;
\
19          case 2: pdcrb_ret__ = stem##2(pcp1, pcp2, __VA_ARGS__); break;
\
20          case 4: pdcrb_ret__ = stem##4(pcp1, pcp2, __VA_ARGS__); break;
\
21          case 8: pdcrb_ret__ = stem##8(pcp1, pcp2, __VA_ARGS__); break;
\
22          default:
\
23                  __bad_size_call_parameter(); break;
\
24          }
\
25          pdcrb_ret__;
\
26  })
```

If the value of PCP(PCP1, PCP2) is the same as the old value (oval1, oval2), substitute the new value (nval1, nval2) for the pcp.

- Depending on the data length, the argument will call stem1~stem8.
  - e.g. stem=this_cpu_cmpxchg_double_
    - this_cpu_cmpxchg_double_1, this_cpu_cmpxchg_double_2, this_cpu_cmpxchg_double_4, this_cpu_cmpxchg_double_8

this_cpu_cmpxchg_double_1()

include/asm-generic/percpu.h

```
01  #ifndef this_cpu_cmpxchg_double_1
02  #define this_cpu_cmpxchg_double_1(pcp1, pcp2, oval1, oval2, nval1, nval
    2) \
03          this_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2)
04  #endif
05  #ifndef this_cpu_cmpxchg_double_2
06  #define this_cpu_cmpxchg_double_2(pcp1, pcp2, oval1, oval2, nval1, nval
    2) \
07          this_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2)
08  #endif
09  #ifndef this_cpu_cmpxchg_double_4
10  #define this_cpu_cmpxchg_double_4(pcp1, pcp2, oval1, oval2, nval1, nval
    2) \
11          this_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2)
12  #endif
13  #ifndef this_cpu_cmpxchg_double_8
14  #define this_cpu_cmpxchg_double_8(pcp1, pcp2, oval1, oval2, nval1, nval
    2) \
15          this_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2)
16  #endif
```

If the value of PCP(PCP1, PCP2) is the same as the old value (oval1, oval2), substitute the new value (nval1, nval2) for the pcp.

- On a 32-bit arm, there is no operation to atomically process a double word, so it calls the generic function.

this_cpu_generic_cmpxchg_double()

include/asm-generic/percpu.h

```
#define this_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1, nval2) \
({                                                                              \
        int __ret;                                                              \
        unsigned long __flags;                                                  \
        raw_local_irq_save(__flags);                                            \
        __ret = raw_cpu_generic_cmpxchg_double(pcp1, pcp2,                      \
                        oval1, oval2, nval1, nval2);                            \
        raw_local_irq_restore(__flags);                                         \
        __ret;                                                                  \
})
```

If the value of PCP(PCP1, PCP2) is the same as the old value (oval1, oval2) while the interrupt is disabled for a while, the new value (NVL1, NVL2) is substituted for the PCP, and the interrupt is reversed.

raw_cpu_cmpxchg_double()

include/linux/percpu-defs.h

```
#define raw_cpu_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1, nval2) \
        __pcpu_double_call_return_bool(raw_cpu_cmpxchg_double_, pcp1, pcp2, oval1, oval2, nval1, nval2)
```

If the value of pcp(pcp1, pcp2) is equal to the old value (oval1, oval2), it returns true if it succeeds in substituting a new value (nval1, nval2) into the pcp.

raw_cpu_cmpxchg_double_1()

include/asm-generic/percpu.h

```
#ifndef raw_cpu_cmpxchg_double_1
#define raw_cpu_cmpxchg_double_1(pcp1, pcp2, oval1, oval2, nval1, nval2)        \
        raw_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1, nval2)
#endif
#ifndef raw_cpu_cmpxchg_double_2
#define raw_cpu_cmpxchg_double_2(pcp1, pcp2, oval1, oval2, nval1, nval2)        \
```

```
07          raw_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2)
08  #endif
09  #ifndef raw_cpu_cmpxchg_double_4
10  #define raw_cpu_cmpxchg_double_4(pcp1, pcp2, oval1, oval2, nval1, nval2)
    \
11          raw_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2)
12  #endif
13  #ifndef raw_cpu_cmpxchg_double_8
14  #define raw_cpu_cmpxchg_double_8(pcp1, pcp2, oval1, oval2, nval1, nval2)
    \
15          raw_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2)
16  #endif
```

If the value of pcp(pcp1, pcp2) is equal to the old value (oval1, oval2), it returns true if it succeeds in substituting a new value (nval1, nval2) into the pcp.

- On a 32-bit arm, there is no operation to atomically process a double word, so it calls the generic function.

raw_cpu_generic_cmpxchg_double()

include/asm-generic/percpu.h

```
01  #define raw_cpu_generic_cmpxchg_double(pcp1, pcp2, oval1, oval2, nval1,
    nval2) \
02  ({
    \
03          int __ret = 0;
    \
04          if (raw_cpu_read(pcp1) == (oval1) &&
    \
05                           raw_cpu_read(pcp2)  == (oval2)) {
    \
06                  raw_cpu_write(pcp1, nval1);
    \
07                  raw_cpu_write(pcp2, nval2);
    \
08                  __ret = 1;
    \
09          }
    \
10          (__ret);
    \
11  })
```

If the value of pcp(pcp1, pcp2) is equal to the old value (oval1, oval2), it returns true if it succeeds in substituting a new value (nval1, nval2) into the pcp.

raw_cpu_read()

include/linux/percpu-defs.h

```
1   #define raw_cpu_read(pcp)               __pcpu_size_call_return(raw_cpu_
    read_, pcp)
```

pcp value.

raw_cpu_write()

include/linux/percpu-defs.h

```
1 #define raw_cpu_write(pcp, val)         __pcpu_size_call(raw_cpu_write_,
pcp, val)
```

Substitute the val value for the pcp value.

__pcpu_size_call_return()

include/linux/percpu-defs.h

```
01 #define __pcpu_size_call_return(stem, variable)
\
02 ({
\
03         typeof(variable) pscr_ret__;
\
04         __verify_pcpu_ptr(&(variable));
\
05         switch(sizeof(variable)) {
\
06         case 1: pscr_ret__ = stem##1(variable); break;
\
07         case 2: pscr_ret__ = stem##2(variable); break;
\
08         case 4: pscr_ret__ = stem##4(variable); break;
\
09         case 8: pscr_ret__ = stem##8(variable); break;
\
10         default:
\
11                 __bad_size_call_parameter(); break;
\
12         }
\
13         pscr_ret__;
\
14 })
```

According to the size of the variable type, call stem1~8 specified as an argument and return the pcp value.

- e.g. stem=raw_cpu_read_
  - voice=raw_cpu_read_1, vote=raw_cpu_read_2, voice=raw_cpu_read_4, vote=raw_cpu_read_8

__pcpu_size_call()

include/linux/percpu-defs.h

```
01 #define __pcpu_size_call(stem, variable, ...)
\
02 do {
\
```

```
        __verify_pcpu_ptr(&(variable));
\
        switch(sizeof(variable)) {
\
                case 1: stem##1(variable, __VA_ARGS__);break;
\
                case 2: stem##2(variable, __VA_ARGS__);break;
\
                case 4: stem##4(variable, __VA_ARGS__);break;
\
                case 8: stem##8(variable, __VA_ARGS__);break;
\
                default:
\
                        __bad_size_call_parameter();break;
\
        }
\
} while (0)
```

Call STEM1~8 specified as an argument according to the size of the variable type and assign the val value to the PCP value.

- e.g. stem=raw_cpu_write_
  - voice=raw_cpu_write_1, vote=raw_cpu_write_2, voice=raw_cpu_write_4, vote=raw_cpu_write_8

raw_cpu_read_1()

include/asm-generic/percpu.h

```
#ifndef raw_cpu_read_1
#define raw_cpu_read_1(pcp) (*raw_cpu_ptr(&(pcp)))
#endif
#ifndef raw_cpu_read_2
#define raw_cpu_read_2(pcp) (*raw_cpu_ptr(&(pcp)))
#endif
#ifndef raw_cpu_read_4
#define raw_cpu_read_4(pcp) (*raw_cpu_ptr(&(pcp)))
#endif
#ifndef raw_cpu_read_8
#define raw_cpu_read_8(pcp) (*raw_cpu_ptr(&(pcp)))
#endif
```

Read the value of pcp.

raw_cpu_write_1()

include/asm-generic/percpu.h

```
#ifndef raw_cpu_write_1
#define raw_cpu_write_1(pcp, val)       raw_cpu_generic_to_op(pcp, val,
=)
#endif
#ifndef raw_cpu_write_2
#define raw_cpu_write_2(pcp, val)       raw_cpu_generic_to_op(pcp, val,
=)
#endif
#ifndef raw_cpu_write_4
#define raw_cpu_write_4(pcp, val)       raw_cpu_generic_to_op(pcp, val,
=)
```

```
09 #endif
10 #ifndef raw_cpu_write_8
11 #define raw_cpu_write_8(pcp, val)       raw_cpu_generic_to_op(pcp, val,
   =)
12 #endif
```

Store the val value in pcp.

raw_cpu_generic_to_op()

include/asm-generic/percpu.h

```
1 #define raw_cpu_generic_to_op(pcp, val, op)
  \
2 do {
  \
3         *raw_cpu_ptr(&(pcp)) op val;
  \
4 } while (0)
```

OP operations on pcp and val values.

- e.g. raw_cpu_generic_to_op(pcp, val, +=)
  - pcp += val

consultation

- per-cpu -1- (Basic) (http://jake.dothome.co.kr/per-cpu) | 문c
- per-cpu -2- (initialize) (http://jake.dothome.co.kr/setup_per_cpu_areas) | Qc
- per-cpu -3- (dynamic allocation (http://jake.dothome.co.kr/per-cpu-dynamic)) | Qc
- Per-cpu -4- (atomic operations) | Question C – Current Article

LEAVE A COMMENT

Your email will not be published. Required fields are marked with *

Comments

name *

email *

Website

WRITE A COMMENT

‹ Zonned Allocator -3- (Buddy Page Allocation) (http://jake.dothome.co.kr/buddy-alloc/)

Slub Memory Allocator -13- (slabinfo) › (http://jake.dothome.co.kr/slub-slabinfo/)

Munc Blog (2015 ~ 2024)