按照inode去写

enum node\_stat\_item {

NR\_LRU\_BASE,

NR\_FILE\_PAGES,

NR\_FILE\_DIRTY,

};

struct task\_struct {

int nr\_dirtied;

int nr\_dirtied\_pause;

unsigned long dirty\_paused\_when;

}

struct bdi\_writeback {

int dirty\_exceeded;

}

struct backing\_dev\_info {

struct bdi\_writeback wb;

};

struct dirty\_throttle\_control {

struct bdi\_writeback \*wb;

unsigned long avail;

unsigned long dirty;

};

static DEFINE\_PER\_CPU(int,bdp\_ratelimits);

DEFINE\_PER\_CPU(int,dirty\_throttle\_leaks) = 0;

balance\_dirty\_pages\_ratelimited(struct address\_space \*mapping)

{

struct backing\_dev\_info \*bdi = inode\_to\_bdi(inode);

int ratelimit;

struct bdi\_writeback \*wb = NULL;

int \*p;

wb = &bdi->wb;

ratelimit = current->nr\_dirtied\_pause;

if(wb->dirty\_exceeded)

ratelimit = min(ratelimit,32 >>(PAGE\_SHIFT-10));

p = this\_cpu\_ptr(&bdp\_ratelimits);

p = this\_cpu\_ptr(&dirty\_throttle\_leaks);

if(unlikely(current->nr\_dirtied >=ratelimit))

balance\_dirty\_pages(wb,current->nr\_dirtied);

}

#define GDTC\_INIT(\_\_wb) .wb = (\_\_wb), \

.wb\_completions = &(\_\_wb)->completions

balance\_dirty\_pages(wb,current->nr\_dirtied)

{

struct dirty\_throttle\_control gdtc\_stor = {GDTC\_INIT(wb)};

unsigned long nr\_reclaimable;

struct dirty\_throttle\_control \*const gdtc = &gdtc\_stor;

bool strictlimit = bdi->capabilities &BDI\_CAP\_STRICTLIMIT;

struct dirty\_throttle\_control mdtc\_stor ={MDTC\_INIT(wb,&gdtc\_stor)};

/\*

如果CONFIG\_CGROUP\_WRITEBACK未定义，mdtc\_valid则返回false，mdtc则是NULL

\*/

struct dirty\_throttle\_control \*const mdtc = mdtc\_valid(&mdtc\_stor)?

&mdtc\_stor:NULL;

for(;;) {

unsigned long now = jiffies;

unsigned long dirty,thresh,bg\_thresh;

nr\_reclaimable = global\_node\_page\_state(NR\_FILE\_DIRTY) + global\_node\_page\_state(NR\_UNSTABLE\_NFS);

/\*

\*/

gdtc->avail = global\_dirtyable\_memory();

gdtc->dirty = nr\_reclaimable +

global\_node\_page\_state(NR\_WRITEBACK);

domain\_dirty\_limits(gdtc);

dirty = gdtc->dirty;

thresh = gdtc->thresh;

bg\_thresh = gdtc->bg\_thresh;

/\*

dirty\_freerun\_ceiling,是thresh和bg\_thresh的一半

\*/

if(dirty <= dirty\_freerun\_ceiling(thresh,bg\_thresh)) {

unsigned long intv = dirty\_poll\_interval(dirty,thresh);

current->dirty\_paused\_when = now;

current->nr\_dirtied = 0;

current->nr\_dirtied\_pause = intv;

break;

}

if(!writeback\_in\_process(wb))

wb\_start\_backgroud\_writeback(wb);

if(!strictlimit)

wb\_dirty\_limits(gdtc);

}

}

global\_dirtyable\_memory()

{

unsigned long x;

x = global\_zone\_page\_state(NR\_FREE\_PAGES);

/\*

totalreserve\_pages是wmark high和lowmem\_reserve总和

\*/

x-=min(x,totalreserve\_pages);

x+=global\_node\_page\_state(NR\_INACTIVE\_FILE);

x+=global\_node\_page\_state(NR\_ACTIVE\_FILE);

return x+1;

}

/\*

根据vm\_dirty\_ratio 和dirty\_background\_ratio计算ratio和bg\_ratio，再计算thresh和bg\_thresh，然后将计算结果赋值给dtc

\*/

domain\_dirty\_limits(dtc)

{

unsigned long bytes = vm\_dirty\_bytes;

unsigned long bg\_bytes = dirty\_background\_bytes;

const unsigned long available\_memory = dtc->avail;

unsigned long ratio = (vm\_dirty\_ratio\*PAGE\_SIZE)/100;

unsigned long bg\_ratio = (dirty\_background\_ratio \*PAGE\_SIZE)/100;

struct task\_struct \*tsk;

/\*

如果CONFIG\_CGROUP\_WRITEBACK没有定义，则mdtc\_gdtc返回NULL

\*/

struct dirty\_throttle\_control \*gdtc = mdtc\_gdtc(dtc);

if(gdtc) {

unsigned long global\_avail = gdtc->avail;

if(bytes)

ratio = min(DIV\_ROUND\_UP(bytes,global\_avail),PAGE\_SIZE);

if(bg\_bytes)

bg\_ratio = min(DIV\_ROUND\_UP(bg\_bytes,global\_avail),PAGE\_SIZE)

bytes = bg\_bytes =0;

}

if(bytes)

thresh = DIV\_ROUND\_UP(bytes,PAGE\_SIZE)

else

thresh = (ratio \*available\_memory)/PAGE\_SIZE;

if(bg\_bytes)

bg\_thresh = DIV\_ROUND\_UP(bg\_bytes,PAGE\_SIZE);

else

bg\_thresh = (bg\_ratio\*available\_memory)/PAGE\_SIZE;

if(bg\_thresh>=thresh)

bg\_thresh = thresh/2;

tsk = current;

if(tsk->flags &PF\_LESS\_THROTTHLE || rt\_task(tsk)) {

bg\_thresh += bg\_thresh/4 + global\_wb\_domain.dirty\_limit/32;

}

dtc->thresh = thresh;

dtc->bg\_thresh = bg\_thresh;

}

wb\_dirty\_limits(dtc)

{

}