DMA

#define LRU\_BASE 0

#define LRU\_ACTIVE 1

#define LRU\_FILE 2

enum lru\_list {

LRU\_INACTIVE\_ANON = LRU\_BASE,

LRU\_ACTIVE\_ANON = LRU\_BASE + LRU\_ACTIVE,

LRU\_INACTIVE\_FILE = LRU\_BASE + LRU\_FILE,

LRU\_ACTIVE\_FILE = LRU\_BASE + LRU\_FILE + LRU\_ACTIVE,

NR\_LRU\_LISTS,

};

Struct lruvec {

Struct list\_head lists[NR\_LRU\_LISTS];

};

Struct pglist\_data {

Struct zone node\_zones[MAX\_NR\_ZONES];

Struct page \*node\_mem\_map;/\*page 管理数组\*/

Unsigned long node\_spanned\_pages;/\*total size of physical page range,including hole\*/

Unsigned long node\_start\_pfn;

Struct lruvec lruvec;

};

Enum zone\_type {

ZONE\_NORMAL,

ZONE\_MOVABLE,

\_\_MAX\_NR\_ZONES

};

Enum migratetype {

MIGRATE\_UNMOVABLE,

MIGRATE\_MOVABLE,

MIGRATE\_RECLAIMABLE,

MIGRATE\_PCPTYPES,

MIGRATE\_HIGHATOMIC = MIGRATE\_PCPTYPES,

MIGRATE\_CMA,

MIGRATE\_ISOLATE,

MIGRATE\_TYPES

};

ZONE\_DMA,ZONE\_NORMAL,ZONE\_HIGHMEM,ZONE\_MOVABLE,ZONE\_DEVICE,\_\_MAX\_NR\_ZONES

Struct zone {

Struct free\_area free\_area[MAX\_ORDER];

Unsigned long zone\_start\_pfn;

Unsigned long spanned\_pages;

Unsigned long present\_pages;

Unsigned long \*pageblock\_flags;

};

Struct migratetype {

MIGRATE\_UNMOVABLE,

MIGRATE\_MOVABLE,

MIGRATE\_RECLAIMABLE

};

Struct free\_area {

Struct list\_head free\_list[MIGRATE\_TYPES];

Unsigned int nr\_free;

};

Struct page {

Unsigned long flags;

Struct list\_head lru;

Atomic\_t \_refcount;

Atomic \_mapcount;

};

Of\_reserved\_mem

Of\_reserved\_mem的属性nomap的含义是从memory中移除，不做任何page的分配，如果在低端内存，则不做映射。

## Page的分配

Setup\_arch->

Paging\_init->

bootmem\_init->

zone\_sizes\_init->

Free\_area\_init\_node->

Alloc\_node\_mem\_map

在allloc\_node\_mem\_map中分为CONFIG\_FLAT\_NODE\_MEM\_MAP和非CONFIG\_FLAT\_NODE\_MEM\_MAP

Alloc\_node\_mem\_map（struct pglist\_data \*pgdat）

{

Unsigned long start =0;

If(!pgdat->node\_spanned\_pages)

Return;

Start = pgdat->node\_start\_pfn & ~(MAX\_ORDER\_NR\_PAGES -1);

offset = pgdat->node\_start\_pfn - start;

if(!pgdat->node\_mem\_map) {

unsigned long size,end;

struct page \*map;

end = pgdat\_end\_pfn(pgdat);

}

Mem\_map = pgdat->node\_mem\_map;

}

## Page的初始化

**Mm/page\_alloc.c**

**Static unsigned long nr\_all\_pages \_\_initdata;**

**Static unsigned long nr\_kernel\_pages \_\_initdata;**

**Int page\_group\_by\_mobility\_diasble \_\_read\_mostly;**

**Include/linux/pageblock-flags.h**

**#define pageblock\_order 10**

**#define pageblock\_nr\_pages (1UL << pageblock\_order)**

Setup\_arch->

Paging\_init->

bootmem\_init->

zone\_sizes\_init->

Free\_area\_init\_node->

Free\_area\_init\_core(pgdat)

Free\_area\_init\_core(struct pglist\_data \*pgdat)

{

Enum zone\_type j;

Pgdat\_init\_internals(pgdat);

For(j=0;j<MAX\_NR\_ZONES;j++)

{

Struct zone \*zone = pgdat->node\_zones +j;

Unsigned long zone\_start\_pfn = zone->zone\_start\_pfn;

Unsigned long size,freesize,memmap\_pages;

Size = zone->spanned\_pages;

Freesize = zone->present\_pages;

Memmap\_pages = calc\_memmap\_size(size,freesize);

Zone\_init\_internals(zone,j,nid,freesize);

/\*

如果CONFIG\_HUGETLB\_PAGE\_SIZE\_VARIABLE没定义，则set\_pageblock\_order为空

\*/

Set\_pageblock\_order();

/\*

如果CONFIG\_SPARESEMEM不为空，则setup\_usemap会定义，否则为空

Setup\_usemap分配zone的pageblock\_flags

\*/

Setup\_usemap(pgdat,zone,zone\_start\_pfn,size);

Init\_currently\_empty\_zone(zone,zone\_start\_pfn,size);

Memmap\_init(size,nid,j,zone\_start\_pfn);

}

}

Pgdat\_init\_internals(pgdat)

{

Lruvec\_init(node\_lruvec(pgdat));

}

Init\_currently\_empty\_zone(zone,zone\_start\_pfn,size)

{

Struct pglist\_data \*pgdat = zone->zone\_pgdat;

Zone->zone\_start\_pfn = zone\_start\_pfn;

Zone\_init\_free\_lists(zone);

}

Zone\_init\_free\_lists(zone)

{

Unsigned int order,t;

For\_each\_migratetype\_order(order,t) {

INIT\_LIST\_HEAD(&zone->free\_area[order].free\_list[t]);

Zone->free\_area[order].nr\_free =0;

}

}

Memmap\_init(size,nid,j,zone\_start\_pfn)

->memmap\_init\_zone(size,nid,zone,start\_pfn,MEMMAP\_EARLY,NULL)

{

Unsigned long pfn,end\_pfn =start\_pfn+size;

For(pfn = start\_pfn;pfn<end\_pfn;pfn++)

{

If(context == MEMMAP\_EARLY)

{

}

Page = pfn\_to\_page(pfn);

\_\_init\_single\_page(page,pfn,zone,nid);

/\*

如果pfn是pageblock\_nr\_pages大小对齐

\*/

If(!(pfn & (pageblock\_nr\_pages-1))) {

/\*

\*/

Set\_pageblock\_migratetype(page,MIGRATE\_MOVABLE);

Cond\_resched();

}

}

}

\_\_init\_single\_page(page,pfn,zone,nid)

{

/\*

Mem\_zero\_struct\_page的作用是 memset page

\*/

Mm\_zero\_struct\_page(page);

/\*

初始化page的zone和nid，page的zone和nid记录在page的flags上\*/

Set\_page\_links(page,zone,nid,pfn);

/\*

初始化page的\_refcount为1

\*/

Init\_page\_count(page）;

/\*

初始化\_mapcount为-1

\*/

Page\_mapcount\_reset(page);

INIT\_LIST\_HEAD(&page->lru);

}

Set\_pageblock\_migratetype(page,MIGRATE\_MOVABLE)

{

/\*

page\_group\_by\_mobility\_disabled由build\_all\_zonelists决定

\*/

If(unlikely(page\_group\_by\_mobility\_disabled)&&

Migratetype < MIGRATE\_PCPTYPES)

Migratetype = MIGRATE\_UNMOVABLE;

Set\_pageblock\_flags\_group(page,(unsigned long)migratetype,PB\_migrate,PB\_migrate\_end);

}

Set\_pageblock\_flags\_group(page,(unsigned long)migratetype,PB\_migrate,PB\_migrate\_end)

{

}

## Page释放到伙伴系统

## zone和gfp选项间的转换

zone是根据配置的不同而个数不同，所以ZONES\_SHIFT也是根据MAX\_NR\_ZONES的不同而不同，我们看一下默认的ZONE配置:

enum zone\_type {

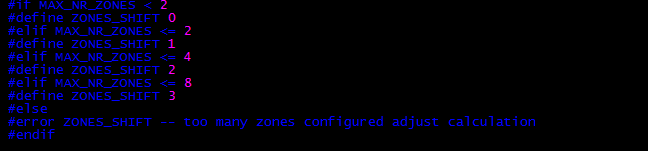
ZONE\_NORMAL,

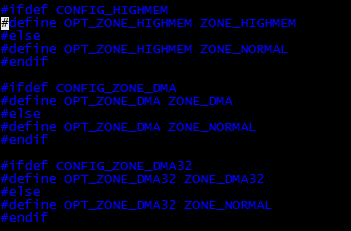
ZONE\_MOVABLE,

\_\_MAX\_NR\_ZONES,

};

只有ZONE\_NORMAL和ZONE\_MOVABLE,所以MAX\_NR\_ZONES是2，所以ZONES\_SHIFT是1，如下所示:





#define GFP\_ZONEMASK (\_\_GFP\_DMA | \_\_GFP\_HIGHMEM | \_\_GFP\_DMA32 | \_\_GFP\_MOVABLE)

#define GFP\_ZONES\_SHIFT ZONES\_SHIFT

#define GFP\_ZONE\_TABLE ( \

(ZONE\_NORMAL << 0\*GFP\_ZONES\_SHIFT) \

| (OPT\_ZONE\_DMA << \_\_\_\_GFP\_DMA \*GFP\_ZONES\_SHIFT) \

| (OPT\_ZONE\_HIGHMEM << \_\_\_\_GFP\_HIGHMEM \*GFP\_ZONES\_SHIFT) \

| (OPT\_ZONE\_DMA32 << \_\_\_\_GFP\_DMA32 \*GFP\_ZONES\_SHIFT) \

| (ZONE\_NORMAL <<\_\_\_\_GFP\_MOVABLE \*GFP\_ZONES\_SHIFT) \

| (OPT\_ZONE\_DMA << (\_\_\_\_GFP\_MOVABLE | \_\_\_\_GFP\_DMA)\*GFP\_ZONES\_SHIFT)\

| (ZONE\_MOVABLE <<(\_\_\_\_GFP\_MOVABLE | \_\_\_\_GFP\_HIGHMEM)\*GFP\_ZONES\_SHIFT) \

| (OPT\_ZONE\_DMA32 <<(\_\_\_\_GFP\_MOVABLE | \_\_\_\_GFP\_DMA32)\*GFP\_ZONES\_SHIFT)

enum zone\_type gfp\_zone(gfp\_t flags)

{

enum zone\_type z;

int bit = (\_\_force int)(flags & GFP\_ZONEMASK);

z = (GFP\_ZONE\_TABLE >>(bit \*GFP\_ZONES\_SHIFT)) &

((1<<GFP\_ZONES\_SHIFT)-1);

return z;

}

**例子**

gfp\_zone(GFP\_USER)

#define GFP\_USER (\_\_GFP\_RECLAIM | \_\_GFP\_IO | \_\_GFP\_FS | \_GFP\_HARDWALL)

GFP\_USER的zone是GFP\_NORMAL

## Zone watermark的初始化

struct pglist\_data {

struct zonelist node\_zonelists

};

struct zone {

unsigned long \_watermark[NR\_WMARK];

};

int min\_free\_kbytes =1024;

int user\_min\_free\_kbytes = -1;

#define GFP\_KERNEL (\_\_GFP\_RECLAIM | \_\_GFP\_IO | \_\_GFP\_FS)

#define SWAP\_CLUSTER\_MAX 32UL

int watermark\_scale\_factor = 10;

在page\_alloc中会通过core\_initcall调用init\_per\_zone\_wmark\_min,init\_per\_zone\_wmark\_min的调用过程如下所示:

init\_per\_zone\_wmark\_min(void)

{

unsigned long lowmem\_kbytes;

int new\_min\_free\_kbytes;

lowmem\_kbytes = nr\_free\_buffer\_pages()\*(PAGE\_SIZE >>10);

new\_min\_free\_kbytes = int\_sqrt(lowmem\_kbytes \*16);

if(new\_min\_free\_kbytes > user\_min\_free\_kbytes)

{

min\_free\_kbytes = new\_min\_free\_kbytes;

if(min\_free\_kbytes < 128)

min\_free\_kbytes=128;

if(min\_free\_kbytes > 65536)

min\_free\_kbytes = 65536;

setup\_per\_zone\_wmarks();

/\*

如果没定义SMP,refresh\_zone\_stat\_thresholds实现为空。

\*/

refresh\_zone\_stat\_thresholds();

setup\_per\_zone\_lowmem\_reserve();

}

}

/\*计算大于high water mark的page的总大小,在init\_per\_zone\_wmark\_min时wmark high为0，所以nr\_free\_zone\_pages即是zone的manged大小\*/

unsigned long nr\_free\_buffer\_pages()

->nr\_free\_zone\_pages(gfp\_zone(GFP\_USER))

{

struct zoneref \*z;

struct zone \*zone;

unsigned long sum =0;

struct zonelist \*zonelist = node\_zonelist(numa\_node\_id(),GFP\_KERNEL);

for\_each\_zone\_zonelist(zone,z,zonelist,offset) {

unsigned long size = zone\_managed\_pages(zone);

unsigned long high = high\_wmark\_pages(zone);

if(size > high)

sum += size-high;

}

return sum;

}

setup\_per\_zone\_wmarks()->

\_\_setup\_per\_zone\_wmarks() {

unsigned long lowmem\_pages =0;

unsigned long pages\_min = min\_free\_kbytes >> (PAGE\_SHIFT -10);

for\_each\_zone(zone) {

if(!is\_highmem(zone))

lowmem\_pages += zone\_managed\_pages(zone);

}

for\_each\_zone(zone) {

u64 tmp;

tmp = (u64) pages\_min\*zone\_managed\_pages(zone);

do\_div(tmp,lowmem\_pages);

if(is\_highmem(zone)) {

unsigned long min\_pages;

min\_pages = zone\_managed\_pages(zone) /1024;

/\*

clamp的含义是如果min\_pages在SWAP\_CLUSTER\_MAX和128之间，则clamp返回min\_pages，如果小于SWAP\_CLUSTER\_MAX，则取SWAP\_CLUSTER\_MAX，如果大于128UL，则取128UL

\*/

min\_pages = clamp(min\_pages,SWAP\_CLUSTER\_MAX,128UL);

zone->\_watermark[WMARK\_MIN] = min\_pages;

} else {

/\*

pages\_min 按照zone占总zone的比例来计算

\*/

zone->\_watermark[WMARK\_MIN] = tmp;

}

/\*

mult\_frac的含义是乘以watermark\_scale\_factor再除以10000,能尽可能减少损失

\*/

tmp = max\_t(u64,tmp>>2,

mult\_frac(zone\_managed\_pages(zone),watermark\_scale\_factor,10000));

zone->\_watermark[WMARK\_LOW] = min\_wmark\_pages(zone) +tmp;

zone->\_watermark[WMARK\_HIGH] = min\_wmark\_pages(zone)+tmp\*2;

zone->watermark\_boost = 0;

}

calculate\_totalreserve\_pages();

}

/\*

随着内存分配，内存的大小会分别进入WATERMARK\_HIGH ,WATERMARK\_LOW,WATERMARK\_MIN三个值。

\*/

calculate\_totalreserve\_pages()

{

struct pglist\_data \*pgdat;

enum zone\_type I,j;

for\_each\_online\_pgdat(pgdat) {

for(i = 0；i< MAX\_NR\_ZONES;i++)

{

struct zone \*zone = pgdat->node\_zones + i;

unsigned long managed\_pages = zone\_manged\_pages(zone);

long max =0;

for(j = i;j<MAX\_NR\_ZONES；j++)

{

if(zone->lowmem\_reserve[j] >max)

{

max = zone->lowmem\_reserve[j];

}

max += high\_wmark\_pages(zone);

pgdat->totalreserve\_pages +=max;

reserve\_pages+=max;

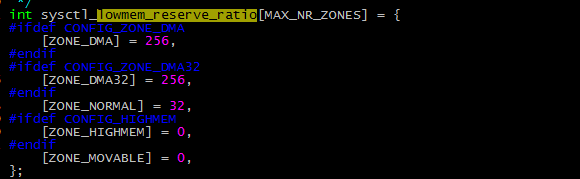
}

}

}

totalreserver\_pages =reserve\_pages;

}



/\*

sysctl\_lowmem\_reserve\_ratio记录着当前zone可以预留给下一个zone的比例，预留page的个数取决于下一个zone的manged page的个数。

当前zone的lowmem\_reserve记录着为下面的zone预留的个数。

\*/

setup\_per\_zone\_lowmem\_reserve()

{

struct pglist\_data \*pgdat;

enum zone\_type j,idx;

for\_each\_online\_pgdat(pgdat) {

for(j=0;j<MAX\_NR\_ZONES;j++) {

struct zone \*zone = pgdat->node\_zones +j;

unsigned long managed\_pages = zone\_managed\_pages(zone);

zone->lowmem\_reserve[j] =0;

idx = j;

while(idx) {

struct zone \* zone = lower\_zone;

idx—;

lower\_zone = pgdat->node\_zones + idx;

lower\_zone->lowmem\_reserve[j] = managed\_pages/sysctl\_lowmem\_reserve\_ratio[idx];

managed\_pages += zone\_managed\_pages(lower\_zone);

}

}

}

}

**如果CONFIG\_HIGHMEM没有定义怎么办，总内存超过lowmem 768MB怎么办**

zone\_sizes\_init(min,max\_low)->

free\_area\_init\_node->

calculate\_node\_totalpages(pgdat,start\_pfn,end\_pfn,zones\_size,zholes\_size)

## arm\_lowmem\_limit

arm\_lowmem\_limit是物理地址。