第 0006 讲 3 内核数据结构-链表和红黑树

1、Linux 内核源码当中，常用的数据结构为链表及红黑树。如链表主要解决元素可以动态创建并删除和插入，每个元素离散存放不需要占据连续的内存。

链表在 Linux 内核数据结构如下：

include > linux > C types.h > hlist_head > first

```
187
188 struct list_head {
189     struct list_head *next, *prev;
190 };
191
192 struct hlist_head {
193     struct hlist_node *first;
194 };
195
196 struct hlist_node {
197     struct hlist_node *next, **pprev;
198 };
199
```

链表静态和动态初始化操作：

include > linux > C list.h > ...

```
23 #define LIST_HEAD_INIT(name) { &(name), &(name) }
24 #define LIST_HEAD(name) \
25     struct list_head name = LIST_HEAD_INIT(name)
26
27 static inline void INIT_LIST_HEAD(struct list_head *list)
28 {
29     WRITE_ONCE(list->next, list);
30     list->prev = list;
31 }
32
```

添加节点到链表，内核直接提供 API 接口：

include > linux > C list.h > ...

```
79
80 static inline void list_add(struct list_head *new, struct list_head *head)
81 {
82     __list_add(new, head, head->next);
83 }
84
85
86 /**
87  * list_add_tail - add a new entry
88  * @new: new entry to be added
89  * @head: list head to add it before
90  *
91  * Insert a new entry before the specified head.
92  * This is useful for implementing queues.
93  */
94 static inline void list_add_tail(struct list_head *new, struct list_head *head)
95 {
96     __list_add(new, head->prev, head);
97 }
98
```

遍历节点内核提供 API 接口：

```
include > linux > C list.h > ...
476  /**
477   * list_for_each    -   iterate over a list
478   * @pos:    the &struct list_head to use as a loop cursor.
479   * @head:   the head for your list.
480   */
481  #define list_for_each(pos, head) \
482      for (pos = (head)->next; pos != (head); pos = pos->next)
483
```

具体实战案例分析如下：

```
vico@ubuntu: ~/Desktop/lab2
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
vico@ubuntu:~/Desktop/lab2$ ls
list.h  main  main.c
vico@ubuntu:~/Desktop/lab2$ gcc list.h main.c -o lm
vico@ubuntu:~/Desktop/lab2$ ls
list.h  lm  main  main.c
vico@ubuntu:~/Desktop/lab2$ ./lm

实现插入50个元素到链表.

输出链表的元素如下:
 1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50

链表释放成功.

vico@ubuntu:~/Desktop/lab2$
```

2、Linux 内核源码里面红黑树主要应用在内存管理及进程调度，将排序的元素组织到树，也正是因为红黑树具有的特色（搜索、删除、插入操作都可以在 O(logN)时间完成，其中 N 为树中的元素个数）。

Red Black Tree 具备二叉树特性：

- 结点是红色或黑色。
- 根结点是黑色。

- 所有叶子都是黑色。（叶子是 NIL 结点）
- 每个红色结点的两个子结点都是黑色。（从每个叶子到根的所有路径上不能有两个连续的红色结点）
- 从任一结点到其每个叶子的所有路径都包含相同数目的黑色结点。

Linux 内核红黑树数据结构如下：

include > linux > C rbtree.h

```
42
43  struct rb_root {
44      struct rb_node *rb_node;
45  };
46
```

include > linux > C rbtree.h > rb_node

```
35
36  struct rb_node {
37      unsigned long  __rb_parent_color;
38      struct rb_node *rb_right;
39      struct rb_node *rb_left;
40  } __attribute__((aligned(sizeof(long))));
41      /* The alignment might seem pointless, but allegedly CRIS needs it */
42
43  struct rb_root {
44      struct rb_node *rb_node;
45  };
```

具体实战案例分析如下：

【内核模块程序】

root@ubuntu: /home/vico/Desktop/Pros2/kernelmodule

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@ubuntu:/home/vico/Desktop/Pros2/kernelmodule# ls
Makefile  rbtreetest.c
root@ubuntu:/home/vico/Desktop/Pros2/kernelmodule# make
make -C /usr/src/linux-headers-5.4.0-110-generic    M=/home/vico/Desktop/Pros2/k
ernelmodule  modules
make[1]: 进入目录“/usr/src/linux-headers-5.4.0-110-generic”
  CC [M]  /home/vico/Desktop/Pros2/kernelmodule/rbtreetest.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/vico/Desktop/Pros2/kernelmodule/rbtreetest.mod.o
  LD [M]  /home/vico/Desktop/Pros2/kernelmodule/rbtreetest.ko
make[1]: 离开目录“/usr/src/linux-headers-5.4.0-110-generic”
root@ubuntu:/home/vico/Desktop/Pros2/kernelmodule# ls
Makefile       Module.symvers  rbtreetest.ko   rbtreetest.mod.c  rbtreetest.o
modules.order  rbtreetest.c    rbtreetest.mod  rbtreetest.mod.o
root@ubuntu:/home/vico/Desktop/Pros2/kernelmodule# insmod rbtreetest.ko
root@ubuntu:/home/vico/Desktop/Pros2/kernelmodule#
```

```
root@ubuntu: /home/vico/Desktop/Pros2/kernelmodule
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
[  767.324637] raid6: sse2x4   xor()  9779 MB/s
[  767.371705] raid6: sse2x2   gen() 13647 MB/s
[  767.420629] raid6: sse2x2   xor()  9407 MB/s
[  767.468553] raid6: sse2x1   gen() 12078 MB/s
[  767.516255] raid6: sse2x1   xor()  7160 MB/s
[  767.516256] raid6: using algorithm avx2x2 gen() 35341 MB/s
[  767.516256] raid6: .... xor() 22579 MB/s, rmw enabled
[  767.516257] raid6: using avx2x2 recovery algorithm
[  767.522925] xor: automatically using best checksumming function   avx
[  767.541550] Btrfs loaded, crc32c=crc32c-intel
[ 2516.977315] perf: interrupt took too long (12941 > 12895), lowering kernel.perf_event_max_sample_rate to 15250
[ 7031.099622] rbtreetest: loading out-of-tree module taints kernel.
[ 7031.099696] rbtreetest: module verification failed: signature and/or required key missing - tainting kernel
[ 7031.103937] key=0
[ 7031.103939] key=2
[ 7031.103940] key=4
[ 7031.103941] key=6
[ 7031.103941] key=8
[ 7031.103942] key=10
[ 7031.103943] key=12
[ 7031.103944] key=14
[ 7031.103945] key=16
[ 7031.103945] key=18
root@ubuntu:/home/vico/Desktop/Pros2/kernelmodule#
```

【用户应用程序】

```
vico@ubuntu: ~/Desktop/Pros2/userapplication
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
vico@ubuntu:~/Desktop/Pros2/userapplication$ ls
main.c  rbtree_augmented.h  rbtree.c  rbtree.h
vico@ubuntu:~/Desktop/Pros2/userapplication$ gcc main.c rbtree_augmented.h rbtree.c rbtree.h -o rbt
vico@ubuntu:~/Desktop/Pros2/userapplication$ ls
main.c  rbt  rbtree_augmented.h  rbtree.c  rbtree.h
vico@ubuntu:~/Desktop/Pros2/userapplication$ ./rbt
```

```
vico@ubuntu: ~/Desktop/Pros2/userapplication
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
nodes[9978].key=7875,nodes[9978].val=8469
nodes[9979].key=1745,nodes[9979].val=3476
nodes[9980].key=789,nodes[9980].val=9649
nodes[9981].key=4120,nodes[9981].val=2509
nodes[9982].key=4356,nodes[9982].val=3118
nodes[9983].key=2799,nodes[9983].val=4381
nodes[9984].key=1612,nodes[9984].val=9604
nodes[9985].key=2594,nodes[9985].val=9264
nodes[9986].key=7000,nodes[9986].val=7800
nodes[9987].key=8347,nodes[9987].val=7429
nodes[9988].key=4106,nodes[9988].val=3469
nodes[9989].key=850,nodes[9989].val=3581
nodes[9990].key=3215,nodes[9990].val=4480
nodes[9991].key=1953,nodes[9991].val=7446
nodes[9992].key=7075,nodes[9992].val=1426
nodes[9993].key=7627,nodes[9993].val=4949
nodes[9994].key=9894,nodes[9994].val=5723
nodes[9995].key=4776,nodes[9995].val=7034
nodes[9996].key=5372,nodes[9996].val=8896
nodes[9997].key=9542,nodes[9997].val=6079
nodes[9998].key=8365,nodes[9998].val=8692
nodes[9999].key=460,nodes[9999].val=6329

请输入要查找节点的数值，系统会根据红黑树算法搜索查询.
输入num数值必须(num>=0 and num<=10000).
用户输入-1退出应用程序?
等待=======>>:8692
搜索结果：在红黑树中找到与该节点匹配的数值:8692

请输入要查找节点的数值，系统会根据红黑树算法搜索查询.
输入num数值必须(num>=0 and num<=10000).
用户输入-1退出应用程序?
等待=======>>:
```