Assignment 5

> 12011702 张镇涛

Q1

First switch checker to `firstfit_check_final`.

```
356 const struct pmm_manager default_pmm_manager = {
357     .name = "default_pmm_manager",
358     .init = default_init,
359     .init_memmap = default_init_memmap,
360     .alloc_pages = default_alloc_pages,
361     .free_pages = default_free_pages,
362     .nr_free_pages = default_nr_free_pages,
363     //.check = default_check,
364     // 合并空闲块之后，请将上面的check注释，下面的check解除注释，进行测试
365     .check = firstfit_check_final,
366 };
367
```

Code:

free blocks can be merged in case when there exists free pages **before or after** the current newly released free block. We need to check if the end of the prior node for previous node in list is the start of newly released block or the start of prior node for next node in list is the end of newly released block.

```
123
126 static void
127 default_free_pages(struct Page *base, size_t n) {
128     assert(n > 0);
129     struct Page *p = base;
130     for (; p != base + n; p ++) {
131         assert(!PageReserved(p) && !PageProperty(p));
132         p->flags = 0;
133         set_page_ref(p, 0);
134     }
135     base->property = n;
136     SetPageProperty(base);
137     nr_free += n;
138
139     if (list_empty(&free_list)) {
140         list_add(&free_list, &(base->page_link));
141     } else {
142         list_entry_t* le = &free_list;
143         while ((le = list_next(le)) != &free_list) {
144             struct Page* page = le2page(le, page_link);
145             if (base < page) {
146                 list_add_before(le, &(base->page_link));
147                 break;
148             } else if (list_next(le) == &free_list) {
149                 list_add(le, &(base->page_link));
150             }
151         }
152     }
153
```

```
154     //----------------------合并空闲块-------------------
155      //check if exists free block(p) after the newly released free block(base)
156
157      list_entry_t* le = list_next(&(base->page_link));
158      if (le != &free_list){
159         p = le2page(le, page_link);
160         //if it's consective then merge
161         if (base + base->property == p){
162             base->property = base->property + p->property; //update property of base
163             ClearPageProperty(p);
164             list_del(&(p->page_link));//delete p from list
165         }
166
167      }
168
169      //check if exists free block(p) before the newly released free block(base)
170      le = list_prev(&(base->page_link));
171      if(le != &free_list){
172         p = le2page(le, page_link);
173         //if it's consective then merge
174         if (p + p->property == base){
175            p->property = p->property + base->property;
176            ClearPageProperty(base);
177            list_del(&(base->page_link));
178            base = p; //reset base to node before
179         }
180
181
182      }
183
184
185     //-----------------------------------------------------
186
187 }
188
```

Result Screenshot:

```
os12011702@vmos-tony:~/oslab/Asg5$ make qemu
+ cc kern/mm/pmm.c
+ cc kern/mm/best_fit_pmm.c
+ ld bin/kernel
riscv64-unknown-elf-objcopy bin/kernel --strip-all -O binary bin/ucore.bin

OpenSBI v0.6
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs     : 8
Current Hart           : 0
Firmware Base          : 0x80000000
Firmware Size          : 120 KB
Runtime SBI Version    : 0.2

MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
os is loading ...
memory management: default_pmm_manager
physcial memory map:
  memory: 0x0000000007e00000, [0x0000000080200000, 0x0000000087ffffff].
check_alloc_page() succeeded!
```

Q2

Implementation

First we switch `pmm_manager` to `best_fit_pmm_manager`.

```
// init_pmm_manager - initialize a pmm_manager instance
static void init_pmm_manager(void) {
    //pmm_manager = &default_pmm_manager;
    pmm_manager = &best_fit_pmm_manager;
    cprintf("memory management: %s\n", pmm_manager->name);
    pmm_manager->init();
}
```

We only need to modify `best_fit_alloc_pages()` function based on default first fit. Other functions are similar to that so we just directly use it with little modification.

```
11
12 static void
13 best_fit_init(void)
14 {
15  //TODO
16   list_init(&free_list);
17   nr_free = 0;
18 }
19
20 static void
21 best_fit_init_memmap(struct Page *base, size_t n)
22 {
23  //TODO
24     assert(n > 0);
25     struct Page *p = base;
26     for (; p != base + n; p ++) {
27         assert(PageReserved(p));
28         p->flags = p->property = 0;
29         set_page_ref(p, 0);
30     }
31     base->property = n;
32     SetPageProperty(base);
33     nr_free += n;
34     if (list_empty(&free_list)) {
35         list_add(&free_list, &(base->page_link));
36     } else {
37         list_entry_t* le = &free_list;
38         while ((le = list_next(le)) != &free_list) {
39             struct Page* page = le2page(le, page_link);
40             if (base < page) {
41                 list_add_before(le, &(base->page_link));
42                 break;
43             } else if (list_next(le) == &free_list) {
44                 list_add(le, &(base->page_link));
45             }
46         }
47     }
48 }
```

In this part we need to modify code from first fit to best fit. Therefore, we wants to **find smallest node from the nodes large enough** for allocation. The code maintains current smallest property and updates based on comparation of newly found potential candidate value.

```
50 static struct Page *
51 best_fit_alloc_pages(size_t n)
52 {
53
54     //TODO
55     assert(n > 0);
56     if (n > nr_free) {
57         return NULL;
58     }
59     struct Page *page = NULL;
60     long long cursmallestprop = 9223372036854775807ll;
61     struct Page *res = NULL;
62     list_entry_t *le = &free_list;
63     while ((le = list_next(le)) != &free_list) {
64         struct Page *p = le2page(le, page_link);
65         if (p->property >= n) {
66             page = p;
67             if (res == NULL){ //first potential result found
68                 res = page;
69                 cursmallestprop = page->property;
70             }
71             else {//compare with already found minimum value
72                 if(page->property < cursmallestprop){
73                     res = page;
74                     cursmallestprop = page->property;
75                 }
76                 else{
77                     continue;
78                 }
79             }
80         }
81     }

32     if (res != NULL) {
33         list_entry_t* prev = list_prev(&(res->page_link));
34         list_del(&(res->page_link));
35         if (res->property > n) {
36             struct Page *p = res + n;
37             p->property = res->property - n;
38             SetPageProperty(p);
39             list_add(prev, &(p->page_link));
90         }
91         nr_free -= n;
92         ClearPageProperty(res);
93     }
94     return res;
95
96
97 }
98
99 static void
00 best_fit_free_pages(struct Page *base, size_t n)
01 {
02  //TODO
03     assert(n > 0);
04     struct Page *p = base;
05     for (; p != base + n; p ++) {
06         assert(!PageReserved(p) && !PageProperty(p));
07         p->flags = 0;
08         set_page_ref(p, 0);
09     }
10     base->property = n;
11     SetPageProperty(base);
12     nr_free += n;
13
```

```
113
114     if (list_empty(&free_list)) {
115         list_add(&free_list, &(base->page_link));
116     } else {
117         list_entry_t* le = &free_list;
118         while ((le = list_next(le)) != &free_list) {
119             struct Page* page = le2page(le, page_link);
120             if (base < page) {
121                 list_add_before(le, &(base->page_link));
122                 break;
123             } else if (list_next(le) == &free_list) {
124                 list_add(le, &(base->page_link));
125             }
126         }
127     }
128
129     //----------------------合并空闲块-------------------
130      //check if exists free block(p) after the newly released free block(base)
131
132      list_entry_t* le = list_next(&(base->page_link));
133      if (le != &free_list){
134         p = le2page(le, page_link);
135         //if it's consective then merge
136         if (base + base->property == p){
137             base->property = base->property + p->property; //update property of base
138             ClearPageProperty(p);
139             list_del(&(p->page_link));//delete p from list
140         }
141
142      }
143
144       //check if exists free block(p) before the newly released free block(base)
145       le = list_prev(&(base->page_link));
146       if(le != &free_list){
147          p = le2page(le, page_link);
148          //if it's consective then merge
149          if (p + p->property == base){
150             p->property = p->property + base->property;
151             ClearPageProperty(base);
152             list_del(&(base->page_link));
153             base = p; //reset base to node before
154          }
155
156
157       }
158
159
160      //-----------------------------------------------------
161
162 }
163
164 static size_t
165 best_fit_nr_free_pages(void)
166 {
167     return nr_free;
168 }
169
```

The **result** is shown in following screenshot:

```
os12011702@vmos-tony:~/oslab/Asg5$ make qemu
+ cc kern/mm/pmm.c
+ ld bin/kernel
riscv64-unknown-elf-objcopy bin/kernel --strip-all -O binary bin/ucore.bin

OpenSBI v0.6
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs     : 8
Current Hart           : 0
Firmware Base          : 0x80000000
Firmware Size          : 120 KB
Runtime SBI Version    : 0.2

MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
os is loading ...
memory management: best_fit_pmm_manager
physcial memory map:
  memory: 0x0000000007e00000, [0x0000000080200000, 0x0000000087ffffff].
check_alloc_page() succeeded!
```