lab2-1 Extra-offline

lab2-1 Extra 课上测试已经结束，若同学们未能在课上完成且有兴趣继续完善该内容，可以将完成后的代码推送到远程分支 `lab2-1-Extra-offline` 进行评测，具体提交方式可以参见本文最后的 `代码提交` 部分。

本任务没有截止时间，也不计入 OS 实验课的成绩。

创建并切换分支

```
git checkout lab2
git add .
git commit -m "xxxxx"
git checkout -b lab2-1-Extra
```

题目描述

请你对现有的物理内存管理机制进行修改，对 MOS 中 $64\ \mathrm{MB}$ 物理内存的高地址 $32\ \mathrm{MB}$ 建立伙伴系统。下面对**本题中**所需要实现的伙伴系统进行描述：

内存区间的初始化

伙伴系统将高地址 $32\ \mathrm{MB}$ 划分为数个内存区间，每个内存区间有两种状态：**已分配**和**未分配**。

每个内存区间的大小只可能是 $4 \times 2^i\ \mathrm{KB}$，其中 i 是整数且 $0 \leq i \leq 10$。

初始，共有 8 个 $4\ \mathrm{MB}$ 大小的内存区间，状态均为**未分配**。

内存区间的分配

每次通过伙伴系统分配 $x\ \mathrm{B}$ 的空间时，找到满足如下三个条件的内存区间：

- 该内存区间的状态为**未分配**。
- 其大小不小于 $x\ \mathrm{B}$。
- 满足上面两个条件的前提下，该内存区间的起始地址最小。

如果不存在这样的内存区间，则本次分配失败；否则，执行如下步骤：

1. 设该内存区间的大小为 $y\ \mathrm{B}$，若 $\frac{y}{2} < x$ 或 $y = 4\ \mathrm{K}$，则将该内存区间的状态设为**已分配**，将该内存区间分配并结束此次分配过程。
2. 否则，将该内存区间分裂成两个大小相等的内存区间，状态均为**未分配**。

3. 继续选择起始地址更小的那个内存区间，并返回步骤 1。

内存区间的释放

当一个内存区间使用完毕，通过伙伴系统释放时，将其状态设为**未分配**。

我们称两个内存区间 x 和 y 是**可合并**的，当且仅当它们满足如下四个条件：

1. x 和 y 的状态均为**未分配**。
2. x 和 y 是由**同一个**内存区间**一次分裂**所产生的两个内存区间。

若存在两个**可合并**的内存区间，则将两个内存区间合并，若合并后仍存在两个**可合并**的内存区间，则继续合并，直到不存在两个**可合并**的内存区间为止。

请你实现如下的三个函数：

初始化函数 `buddy_init`

- 函数原型为：`void buddy_init(void)`
- 调用此函数后，为 MOS 中 $64\ \mathrm{MB}$ 物理内存的高地址 $32\ \mathrm{MB}$ 初始化伙伴系统。初始化结束后，伙伴系统中仅有只有 8 个 $4\ \mathrm{MB}$ 的待分配内存区间。

分配函数 `buddy_alloc`

- 函数原型为：`int buddy_alloc(u_int size, u_int *pa, u_char *pi)`
- 调用此函数后，通过伙伴系统分配大小不小于 `size` 字节的空间，分配逻辑见上述描述。
  如果分配失败，返回 -1。否则，将 `pa` 指向所分配内存区间的起始地址，设所分配内存区间的大小为 $4 \times 2^i\ \mathrm{KB}$，令 `*pi = i`，并返回 0。

释放函数 `buddy_free`

- 函数原型为：`void buddy_free(u_int pa)`
- 调用此函数后，通过伙伴系统释放一个状态为**已分配**的内存区间，其起始地址为 `pa`。释放后的合并逻辑见上述描述。

注意事项

你需要先在 `include/pmap.h` 中加入如下三个函数定义：

```
void buddy_init(void);
int buddy_alloc(u_int size, u_int *pa, u_char *pi);
void buddy_free(u_int pa);
```

之后再在 `mm/pmap.c` 中实现这三个函数。

评测逻辑

评测过程中，**我们会将所有的 `Makefile` 文件、 `include.mk` 以及 `init/init.c` 替换为 lab2 初始配置**，接着将 `init/init.c` 中的 `mips_init` 函数改为如下形式：

```
void mips_init(){
        mips_detect_memory();
        mips_vm_init();
        page_init();

        buddy_init();
        buddy_test();

        *((volatile char*)(0xB0000010)) = 0;
}
```

最后的 `*((volatile char*)(0xB0000010)) = 0;` 会终止 Gxemul 模拟器的运行，避免占用评测资源。

`buddy_test` 是在评测过程中新添加到 `init/init.c` 的函数，其中仅包含以下两种操作：

- 调用 `buddy_alloc` 函数，我们保证 `size` 不为 0。
- 调用 `buddy_free` 函数，我们保证 `pa` 是之前某次调用 `buddy_alloc` 所得到的。

每调用一个函数算一次操作，我们保证总操作数不超过 1000。

运行 `make` 指令的最大时间为 10 秒，运行 Gxemul 模拟器的最大时间为 4 秒。

设伙伴系统管理的物理页数为 n，标程中 `buddy_alloc` 和 `buddy_free` 两个函数的时间复杂度均为 $O(n)$，请你尽量以此复杂度设计算法。

数据点说明

仅有一组数据，实现完全正确才能通过评测，通过后可以获得 100 分。

如果你的实现正确，评测机会返回 `Accepted!`，否则评测机会返回**第一个**错误之处：

- 若函数 `buddy_alloc` 的返回值有误，评测机会返回 `buddy_alloc return value error!`。
- 函数 `buddy_alloc` 会通过参数 `pa` 返回一个值，若该值有误，评测机会返回 `buddy_alloc pa error!`。
- 函数 `buddy_alloc` 会通过参数 `pi` 返回一个值，若该值有误，评测机会返回 `buddy_alloc pi error!`。
- 若你的程序出现其它错误，或未能在限定时间内执行全部程序，评测机会返回 `Other error!`。

测试样例

测试样例一

编写完成后，将 `init/init.c` 中的 `mips_init` 函数删除，并加入如下代码：

```
static void buddy_test(){
        u_int pa_1, pa_2;
        u_char pi_1, pi_2;
        buddy_alloc(1572864, &pa_1, &pi_1);
        buddy_alloc(1048576, &pa_2, &pi_2);
        printf("%x\n%d\n%x\n%d\n", pa_1, (int)pi_1, pa_2, (int)pi_2);
        buddy_free(pa_1);
        buddy_free(pa_2);
}

void mips_init(){
        mips_detect_memory();
        mips_vm_init();
        page_init();

        buddy_init();
        buddy_test();

        *((volatile char*)(0xB0000010)) = 0;
}
```

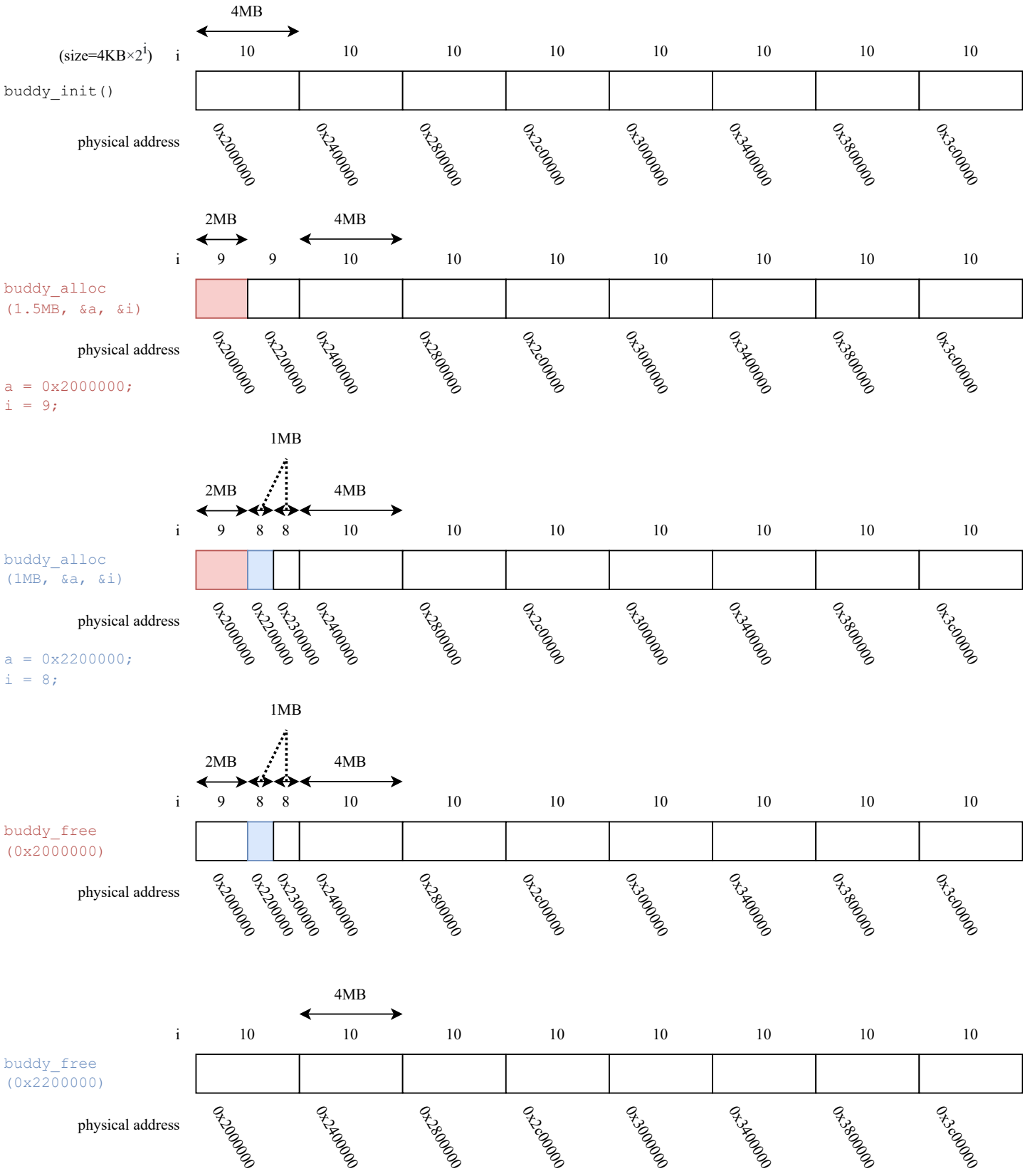
运行如下指令：

```
make clean && make && /OSLAB/gxemul -E testmips -C R3000 -M 64 gxemul/vmlinux
```

正确输出如下：

```
2000000
9
2200000
8
```

该样例的图解如下：



测试样例二

编写完成后，将 `init/init.c` 中的 `mips_init` 函数删除，并加入如下代码：

```
static void buddy_test(){
        u_int pa[10];
        u_char pi;
        int i;
        for(i = 0;i <= 9;i++){
                buddy_alloc(4096 * (1 << i), &pa[i], &pi);
                printf("%x %d\n", pa[i], (int)pi);
        }
        for(i = 0;i <= 9;i += 2) buddy_free(pa[i]);
        for(i = 0;i <= 9;i += 2){
                buddy_alloc(4096 * (1 << i) + 1, &pa[i], &pi);
                printf("%x %d\n", pa[i], (int)pi);
        }
        for(i = 1;i <= 9;i += 2) buddy_free(pa[i]);
        for(i = 1;i <= 9;i += 2){
                buddy_alloc(4096 * (1 << i) + 1, &pa[i], &pi);
                printf("%x %d\n", pa[i], (int)pi);
        }
        for(i = 0;i <= 9;i++) buddy_free(pa[i]);
        printf("%d\n", buddy_alloc(4096 * 1024, &pa[0], &pi));
        printf("%d\n", buddy_alloc(4096 * 1024 + 1, &pa[0], &pi));
}

void mips_init(){
        mips_detect_memory();
        mips_vm_init();
        page_init();

        buddy_init();
        buddy_test();

        *((volatile char*)(0xB0000010)) = 0;
}
```

运行如下指令：

```
make clean && make && /OSLAB/gxemul -E testmips -C R3000 -M 64 gxemul/vmlinux
```

正确输出如下：

```
2000000 0
2002000 1
2004000 2
2008000 3
2010000 4
2020000 5
2040000 6
2080000 7
2100000 8
2200000 9
2000000 1
2010000 3
2040000 5
2100000 7
2400000 9
2004000 2
2020000 4
2080000 6
2200000 8
2800000 10
0
-1
```

代码提交

```
git add .
git commit -m "xxxxx"
git push origin lab2-1-Extra:lab2-1-Extra-offline
```