

# 实验一：Unikraft 内存分配策略分析

姓名：潘梓月 学号：22551192 指导老师：赵新奎 实验日期：2025/9/22

## 实验目标

本实验旨在通过 `qemu` 和 `gdb` 调试 Unikraft 内核，深入分析其内存分配过程。你需要：

- 跟踪不同大小内存请求的分配流程。
- 分析当前 Buddy 和 Slab 分配器的协作机制。
- 找出当前内存分配策略 / 协作机制中可能存在的不合理之处，例如内存碎片或正确性问题。
- 将你的分析和调试过程记录下来，形成一份完整的实验报告。

## 任务与思考题

### 1. 内存分配大小分析

在调试过程中，通过观察内存状态，填写下表，记录不同请求大小对应的实际分配大小。

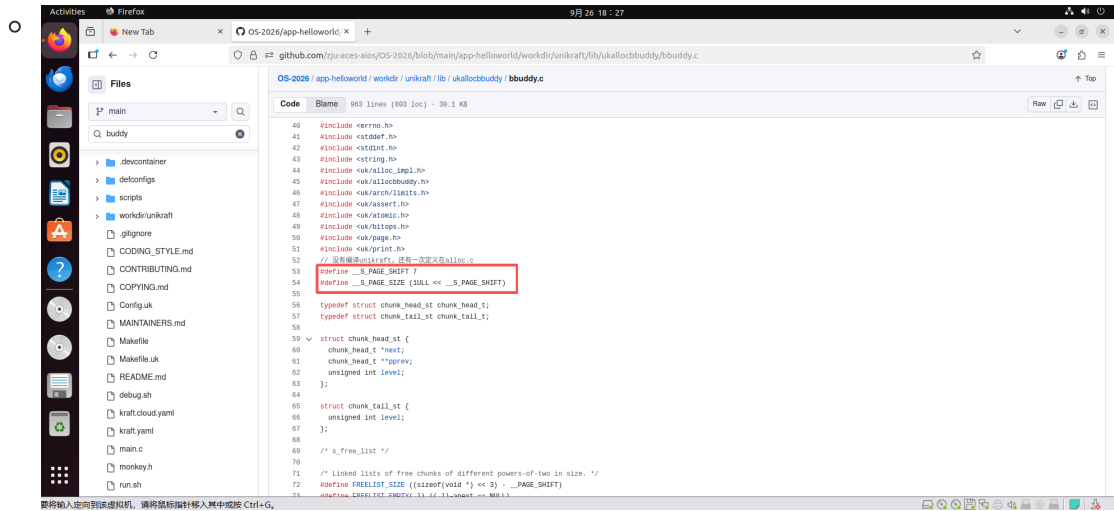
请求分配大小	实际分配大小	分析与说明
96 字节	128 字节	<code>uk_malloc</code> 在用户请求的96字节基础上增加32字节元数据，总需求为128字节。这个大小恰好等于 <code>salloc</code> 的一个最小分配单元，因此 <code>salloc</code> 分配一个128字节的区块，没有任何碎片。
128 字节	256 字节	<code>uk_malloc</code> 在用户请求的128字节基础上增加32字节元数据，总需求为160字节。为满足160字节的需求， <code>salloc</code> 必须向上取整，分配2个128字节的单元，即256字节。
256 字节	384 字节	<code>uk_malloc</code> 在用户请求的256字节基础上增加32字节元数据，总需求为288字节。为满足288字节的需求， <code>salloc</code> 必须向上取整，分配3个128字节的单元，即384字节。
4064 字节	4096 字节	<code>uk_malloc</code> 在用户请求的4064字节基础上增加32字节元数据，总需求为4096字节。这个大小不再由 <code>salloc</code> 处理，而是由 <code>palloc</code> 接管。由于总需求恰好等于一个标准页，即4096字节。
4096 字节	8192 字节	<code>uk_malloc</code> 在用户请求的4096字节基础上增加32字节元数据，总需求为4128字节。这个请求由 <code>palloc</code> 处理。因为4128字节超过了一个页面的大小，所以至少需要2个页面才能容纳。Buddy System会将页面请求向上取整到最近的2的幂次方，实际大小为 $2 \times 4096 = 8192$ 字节。

### 2. 核心问题

请在报告中回答以下问题：

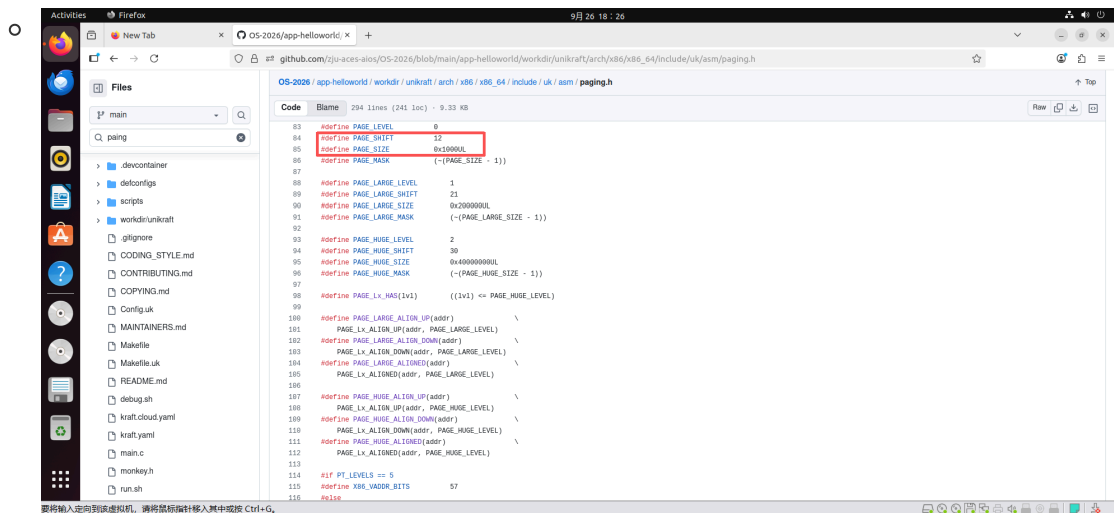
- 最小分配单元**：Unikraft 两种内存分配策略的最小单元是多少？它是如何定义的？  
`salloc` (小块内存分配器):
  - 最小单元：128 字节。

- 定义方式：这个值是在 `bbuddy.c` 文件中通过宏明确定义的。



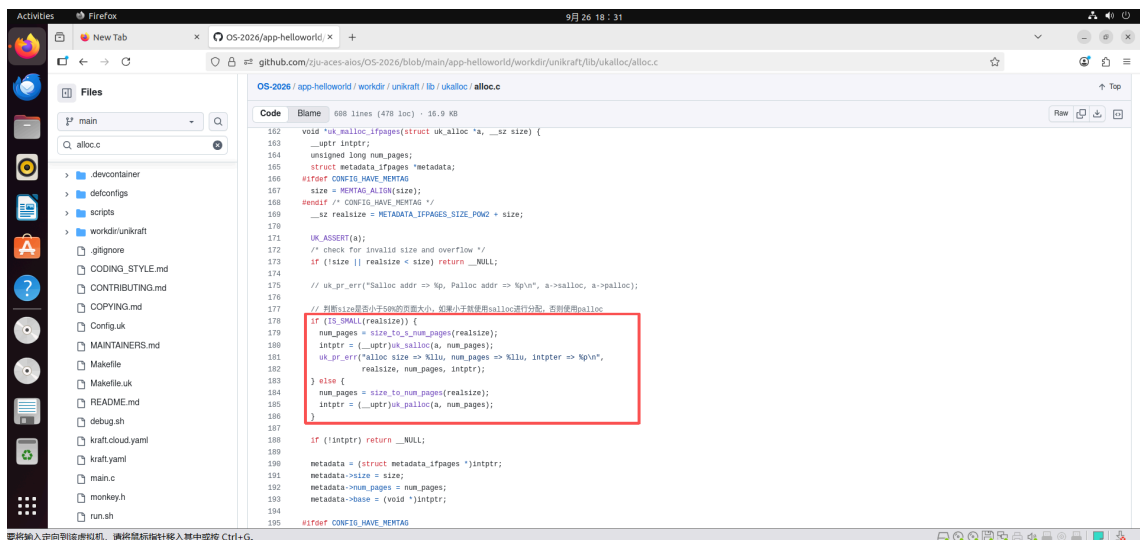
## pallocc (页分配器):

- 最小单元：4096 字节。
- 定义方式：这个值是在 `paging.h` 文件中通过宏明确定义的。



## 2. 分配器选择: `uk_malloc()` 函数在何种条件下会选择 `pallocc`，又在何种条件下会选择 `sallocc`？

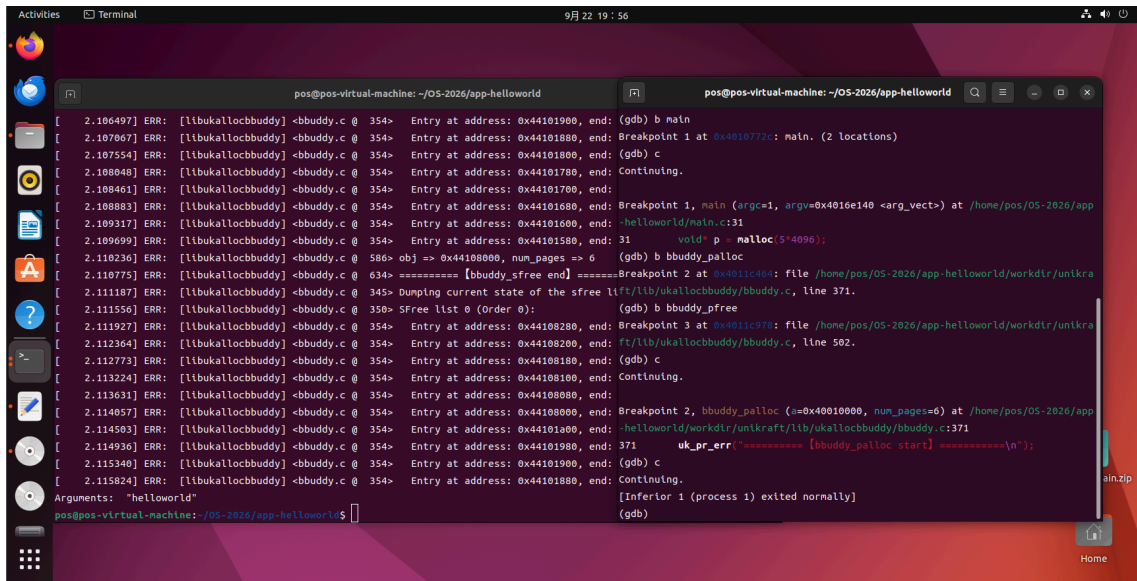
分配器的选择，是基于用户请求的大小加上系统自身管理开销（元数据）后的总大小来进行判断的。



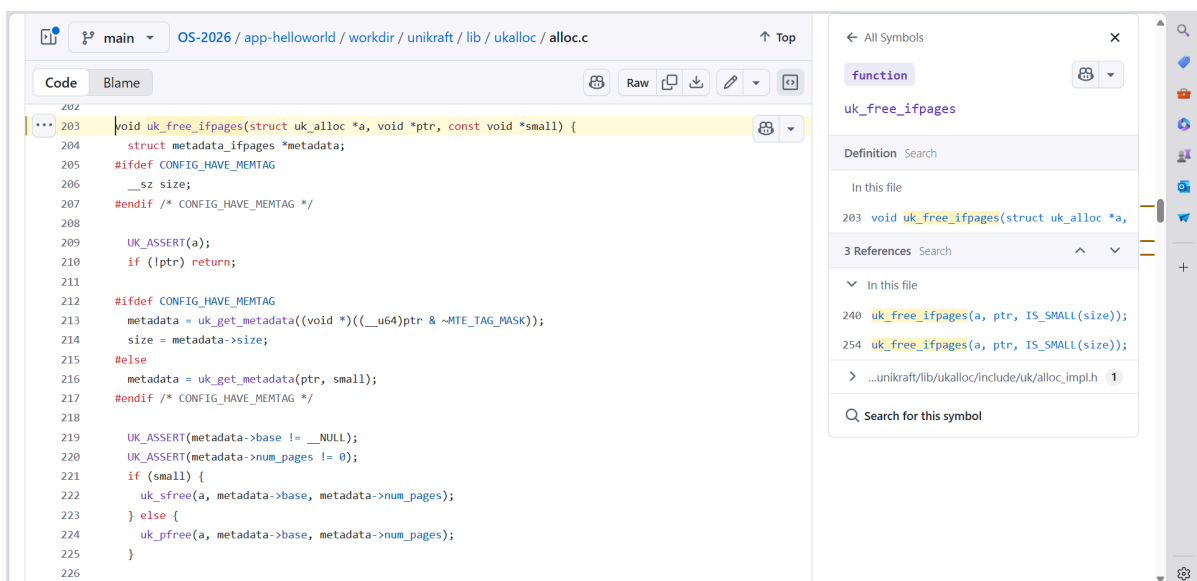
选择 `sallocc` 的条件：当 (用户请求大小 + 32字节元数据) < (页面大小 / 5) 时，系统会调用 `sallocc` 进行小块内存分配。

选择 `malloc` 的条件：当  $(\text{用户请求大小} + 32\text{字节元数据}) \geq (\text{页面大小} / 5)$  时，系统会调用 `malloc` 进行页分配。

3. **大内存分配问题**：当前 `malloc` 在处理大内存（例如，一次性分配多个页面）的分配与回收时，存在一个已知的设计问题。请定位该问题，并尝试在 GDB 中通过 `set` 命令修改相关变量，模拟正确的 `free` 过程，并截图记录结果。



在 `void* p = malloc(5*4096);` 的情况下，设置 `bbuddy_pfree` 断点，但观察到并没有触发 `bbuddy_pfree`。



释放路径是由 `uk_free_ifpages()` 决定走 `uk_sfree` 还是 `uk_pfree`。关键参数是 `small`，它必须正确反映对象当初是“small-page 分配”还是“page 分配”。

目前的实现问题是 `small` 没有被正确传递/初始化。如果大内存分配情况下，`small != 0`，就错误地走进了 `sfree`，把大块当成小页块拆页释放。

模拟正确的实验过程：

1. 在 `uk_free_ifpages` 打断点：

```
b uk_free_ifpages
```

2. 程序停下后，打印 `small`：

```
p small
```

如果不是 0 (但实际上应该是 0, 因为 6 页本该 page alloc), 说明它是垃圾值。

### 3. 手工修正:

```
set small = 0  
c
```

### 4. 观察结果: 正确进入 `bbuddy_pfree`

