Lab2实验报告

2210204 夏雨鍩

2212879 朱馨瑞

2212437 冯言旭

练习1

描述程序在进行物理内存分配的过程以及各个函数的作用

在 `default_pmm.c` 中，程序实现了一个简单的基于“首次适应算法”（First-Fit Memory Allocation，FFMA）的物理内存分配管理器，通过一系列的函数来完成物理内存的初始化、分配和释放工作。以下是每个函数的作用及内存分配过程的具体说明：

1. **`default_init` 函数**

   `default_init` 函数用于初始化内存管理器中的空闲链表和空闲页数计数器。具体实现如下：

   - 使用 `list_init` 初始化 `free_list`，这个链表用于记录所有的空闲内存块。
   - 将 `nr_free` 设置为 0，这是一个全局计数器，用于记录当前空闲的内存页数量。
   - 初始化完成后，内存管理器准备好记录即将分配和释放的内存页块。

2. **`default_init_memmap` 函数**

   `default_init_memmap` 函数用于初始化内存页块，并将其加入到空闲链表中。它会对给定的内存块（由 `base` 指向）执行以下操作：

   - 首先检查页数是否大于0。
   - 对每一页设置默认状态（清空 `flags` 和 `property`，将引用计数 `ref` 设为0），并清除已保留页位（`PageReserved`）。
   - 对于内存块的第一个页，将其 `property` 设为该块的总页数 `n`，并将其标记为 `PageProperty`（表明这是一个有效的空闲块）。
   - 更新空闲页数计数器 `nr_free`。
   - 将该内存块按地址顺序插入到空闲链表 `free_list` 中，以便后续分配和合并操作时可以按顺序进行遍历。

3. **`default_alloc_pages` 函数**

   `default_alloc_pages` 函数用于分配请求数量 `n` 的连续页，采用的是首次适应算法的思想：

   - 首先检查空闲页数量 `nr_free` 是否足够，否则返回 `NULL`。
   - 遍历 `free_list` 寻找第一个满足请求的内存块（`p->property >= n`），如果找到这样的块，则记录其地址并退出循环。
   - 若找到的块比请求数量 `n` 大，则将剩余部分重新作为一个新的空闲块，重新插入空闲链表中。
   - 更新空闲页数计数器 `nr_free`，将分配的内存块标记为“已分配”状态（即清除 `PageProperty` 位）。
   - 返回分配的内存页的地址。

4. **`default_free_pages` 函数**

`default_free_pages` 函数用于释放一个内存块并尝试合并相邻的空闲块。其实现过程如下：

- 首先重置每个页的标志位和引用计数。
- 将释放的内存块插入空闲链表的合适位置，以保持链表按地址递增顺序排列。
- 检查该块的前后是否有相邻的空闲块：
  - 若前一块的结束地址与当前块相邻，则合并它们，并更新 `property` 位。
  - 若后一块的开始地址与当前块相邻，也进行合并，并更新 `property` 位。
- 合并操作可以减少链表中碎片化的空闲块数量，有助于提高后续内存分配的成功率和效率。

5. **`default_nr_free_pages` 函数**

   `default_nr_free_pages` 返回当前空闲页的数量，即 `nr_free` 的值，用于获取当前内存管理器中的空闲资源信息。

6. **`default_check` 和 `basic_check` 函数**

   这两个函数用于测试内存管理器的功能是否正确：

   - `basic_check` 函数通过一系列的分配和释放操作来验证基本的内存管理功能。
   - `default_check` 则进一步测试首次适应算法，验证内存管理器在特定条件下能否正确地合并和分配内存块。

总结

设计上，程序采用首次适应算法来管理内存，通过空闲链表 `free_list` 和 `nr_free` 来跟踪所有空闲块的分布情况。当请求内存时，从链表头部开始扫描，找到第一个足够大的空闲块来满足请求，并在必要时进行分割。释放内存时，按地址顺序插入空闲链表，并检查是否需要合并相邻的块，以减少碎片化。

改进方向

首次适应算法（First Fit）在内存分配中简单且高效，但也存在一些不足之处，可以从以下几个方面进行改进：

1. **循环首次适应（Next Fit）**
   - **改进思路**：在首次适应中，每次分配都从链表头开始扫描。循环首次适应则从上一次分配的位置继续查找。这样可以避免从头开始扫描，提高分配效率。
   - **优缺点**：虽然提高了速度，但可能导致空闲空间集中在链表前端，增加内存碎片。
2. **最佳适应（Best Fit）**
   - **改进思路**：最佳适应算法在空闲链表中查找最接近需求的内存块，以最小化剩余空间，减少空间浪费。
   - **优缺点**：可以减少碎片化，但需要遍历整个链表，导致分配时间较长。
3. **分段首次适应**
   - **改进思路**：将空闲块分为不同大小的区段，每个区段有对应的空闲链表，按请求大小进入相应的链表，从而减少分配时的扫描时间。
   - **优缺点**：分段管理可以加速分配过程，减少碎片，但需要更多的内存来维护多个链表。

练习2

设计实现过程，阐述代码是如何对物理内存进行分配和释放

在这个 `Best Fit` 内存管理实现中，我们使用双向链表 `free_list` 来记录所有空闲的内存块，采用“最佳适应”（Best Fit）算法进行物理内存的分配和释放，具体设计实现过程如下：

1. **初始化内存管理器**
   - 使用 `best_fit_init` 函数初始化空闲链表 `free_list` 并将空闲页数计数器 `nr_free` 置为0。
   - `best_fit_init_memmap` 函数用于初始化给定的一段空闲内存块，将其插入到 `free_list` 中。对于每个内存页，清除所有标志位，将引用计数设为0，然后将整个块按地址顺序插入到链表中，保证链表始终有序。
2. **内存分配（`best_fit_alloc_pages`）**
   - **查找最佳块**：当请求分配内存时，`best_fit_alloc_pages` 函数遍历 `free_list`，找到最小的、且满足请求大小 `n` 的空闲块（即“最佳适应”块）。
   - **分割内存块**：若找到的块比请求大小大，则将多余部分分割成新的空闲块，并插入回 `free_list`。
   - **更新状态**：从空闲链表中删除分配的块，并更新 `nr_free` 计数器和标志位，将该块标记为“已分配”状态。
   - **返回分配地址**：返回分配的内存块地址。
3. **内存释放（`best_fit_free_pages`）**
   - **插入释放块**：释放内存时，将释放的块按地址顺序插入 `free_list` 中，保持链表的有序性。
   - **合并相邻空闲块**：检查释放块前后的空闲块，如果相邻，则将它们合并成一个更大的块，减少碎片化并提高内存利用率。
   - **更新计数器和标志位**：更新 `nr_free` 计数器，将合并后的块标记为“空闲”状态，保证空闲内存块的信息一致性。
4. **空闲页计数和检查**
   - `best_fit_nr_free_pages`：返回当前的空闲页数，帮助了解系统中剩余的空闲资源。
   - **测试函数**：通过 `basic_check` 和 `best_fit_check` 进行基本的分配、释放和合并操作的验证，确保所有功能正常。

总结

整个设计通过维护一个按地址排序的空闲链表，实现了 `Best Fit` 的分配策略，避免较大内存块的频繁分割，减少了碎片化。通过合并相邻的空闲块，有效提升了内存利用率，实现了对物理内存的高效管理。

改进方向

`Best Fit` 算法虽然减少了内存碎片，但仍有改进空间，主要集中在以下方面：

1. **分段空闲列表**
   - **改进思路**：将空闲内存按大小分段，维护多个空闲列表（如 4KB、8KB、16KB 等），每个列表存储对应大小的内存块。分配时只搜索满足需求的最小分段，从而减少搜索时间。

- **优缺点**：这种方法可以显著减少遍历空闲块的时间，提升分配效率，但会增加内存管理的复杂度。

2. **延迟合并（Deferred Coalescing）**
   - **改进思路**：在释放内存时，不立即合并相邻块，而是等到内存紧张时才批量合并，减少频繁合并带来的开销。
   - **优缺点**：延迟合并可以降低合并操作的频率，提高系统响应速度，但可能增加碎片化的风险。在内存紧张时触发合并，可以在最需要的时候提升内存利用率。
3. **改进搜索策略**
   - **改进思路**：`Best Fit` 算法需遍历整个链表找到最佳块，但可以考虑从上次分配的块继续查找或优先查找较小的空闲块，以提高分配效率。
   - **优缺点**：这种方式保留了最佳块选择的精度，但可能减少遍历次数，提高分配速度。
4. **“二次适应”（Modified Best Fit）**
   - **改进思路**：修改 `Best Fit` 策略，只搜索略大于需求的块，以减轻小块空闲内存的浪费。例如，若需求为 `n`，则查找大小为 `n` 到 `n * 2` 的块，从中选最接近需求的块。
   - **优缺点**：可以减少搜索时间，避免产生极小碎片，但对碎片的控制效果可能不如传统 `Best Fit`。
5. **内存紧凑（Memory Compaction）**
   - **改进思路**：在碎片严重时，使用后台进程将已分配的块移动到低地址空间，腾出高地址的连续空闲块。
   - **优缺点**：虽然可以完全消除碎片，但移动内存会消耗大量资源，且只能用于某些内存管理机制（如非直接映射的分页系统）。
6. **合并启发式**
   - **改进思路**：智能化控制合并逻辑，按需求和内存情况决定是否合并。例如，设置一个碎片率阈值，达到阈值时才合并。
   - **优缺点**：在一定范围内减少合并开销，但设置的参数可能需要根据场景调优。

总结

`Best Fit` 算法可以通过改进搜索策略、引入分段管理、延迟合并等方法进一步优化，使内存分配和释放更加高效。这些改进在速度和碎片控制上有所提升，但需平衡复杂性与实际需求。

在这个实验中，涉及到两个重要的内存分配算法：**First-Fit**和**Best-Fit**。通过这两个算法的实现，结合代码分析，可以深入理解操作系统中物理内存的分配和管理。以下是对本实验中重要知识点的总结，以及与OS原理知识的对比和理解。

扩展练习Challenge：硬件的可用物理内存范围的获取方法（思考题）

1. **BIOS/UEFI 引导信息**：
   在系统启动过程中，BIOS或UEFI固件会检测物理内存，并提供一个内存映射。操作系统可以在引导加载阶段通过与 BIOS 或 UEFI 接口通信（例如调用BIOS的 `INT 15h` 或 UEFI 的 `GetMemoryMap` 函数）获取当前物理内存布局信息。该信息包括了系统内可用的物理内存区域、已保留的内存区域等内容，操作系统可以根据该信息了解可用的物理内存范围。

2. **硬件内存检测**：
   通过对内存地址进行读写测试以检测可用的物理内存范围。具体方法是尝试向不同的物理地址写入数据并读取，看是否存在错误。如果某个地址范围不可用或受保护，则可能会产生访问异常或返回错误值。

3. **ACPI 表**：
   在较新的系统中，ACPI（高级配置与电源接口）表中会包含系统内存的详细信息。操作系统可以解析ACPI表中的内存描述数据，以准确地识别不同类型的内存区域（如可用内存、保留内存、设备内存等），从而确定可用的物理内存范围。

实验中的重要知识点与OS原理的联系

1. **First-Fit算法的实现与改进**：
   - **含义**：First-Fit算法是从内存的起始地址开始扫描，找到第一个足够大的空闲内存块进行分配。它的特点是实现简单，能够快速分配内存，但容易造成内存碎片。
   - **OS原理中的对应**：在操作系统的内存管理模块中，First-Fit是一种经典的分配策略，主要用于连续内存分配。它在实际中较为高效，但容易产生外部碎片。
   - **改进空间**：可以通过"碎片整理"或"合并相邻空闲块"来减少内存碎片。另外，还可以使用更加优化的扫描策略，比如跳过一定大小的碎片。

2. **Best-Fit算法的实现与改进**：
   - **含义**：Best-Fit算法遍历所有空闲块，选择大小最接近所需空间的块。这种方法减少了大块内存被小任务占用的可能性，但可能增加查找的时间。
   - **OS原理中的对应**：Best-Fit也是内存管理中的一种重要分配策略。与First-Fit相比，Best-Fit可能减少外部碎片，但查找过程耗时更长。
   - **改进空间**：可以通过数据结构优化，如平衡树或链表，来减少查找时间。此外，也可以设定一定大小的"内存池"以加快查找效率。

3. **物理内存分配与释放**：
   - **含义**：实验中的`default_alloc_pages`和`default_free_pages`函数实现了物理内存的分配和释放。这些函数是内存分配器的核心，通过维护空闲内存块链表实现内存的动态管理。
   - **OS原理中的对应**：在操作系统中，物理内存的分配和释放是内存管理的核心功能。内存分配器需要维护一张空闲链表或位图来标记空闲和已分配的内存块。实验中的实现方法与OS原理中物理内存管理的方式直接对应。
   - **理解差异**：实验实现较为基础，并没有实现更高级的内存管理，如交换空间或分页管理。OS原理中会涉及多种内存管理技术，以更高效地利用内存资源。

4. **内存碎片管理**：
   - **含义**：实验中涉及到的连续内存分配容易产生碎片，尤其是外部碎片。通过合理的算法优化（如Best-Fit）可以一定程度上缓解碎片问题。
   - **OS原理中的对应**：操作系统中的碎片管理是一个重要课题。为了应对内存碎片，操作系统通常会采用分页或分段技术，以将内存分为较小的块来避免碎片化。
   - **理解差异**：实验只涉及连续分配，未涉及分页管理。而OS原理中的分页、分段等技术可以避免碎片化，提高内存利用率。

OS原理中重要但未在实验中体现的知识点

1.**交换空间（Swap Space）管理**：

- **概念**：当内存不足时，操作系统会将一些页面移到硬盘的交换空间，腾出内存以供其他程序使用。

- **作用**：交换空间扩展了物理内存的容量，使得系统能够在低内存环境下执行更大的程序。
- **实验中未体现**：实验未涉及到硬盘的交换空间，未实现交换页面至磁盘的功能。

2.**共享内存和进程间通信（IPC）**：

- **概念**：共享内存是一种高效的进程间通信方式，不同进程可以共享一块物理内存区域来交换数据。
- **作用**：在进程间需要快速共享大量数据时，使用共享内存可以避免数据复制的开销。
- **实验中未体现**：实验主要聚焦单一进程的内存分配和释放，没有涉及多个进程共享内存的机制。