

****

软 件 学 院

《实用操作系统》Project4

**题　　目 改进LiteOS中物理内存分配算法**

**姓　　名 陈澄**

**学　　号 32420212202930**

**班　　级 软工三班**

**实验时间 2023/9/26**

**2023 年 09 月 26 日**

# 实验目的

优化TLSF算法，将Best-fit策略优化为Good-fit策略，进一步降低时间复杂度至O(1)。

# 实验环境

主机： Windows 11

虚拟机： Ubuntu 18.04

开发板： IMAX6ULL MINI

终端： MobaXterm

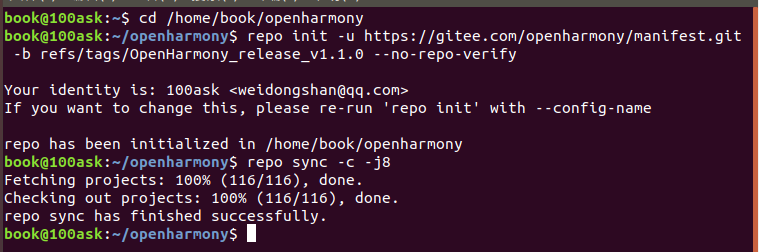
# 实验内容

1. 下载Openharmony1.1.0 LTS版本

cd /home/book/openharmony

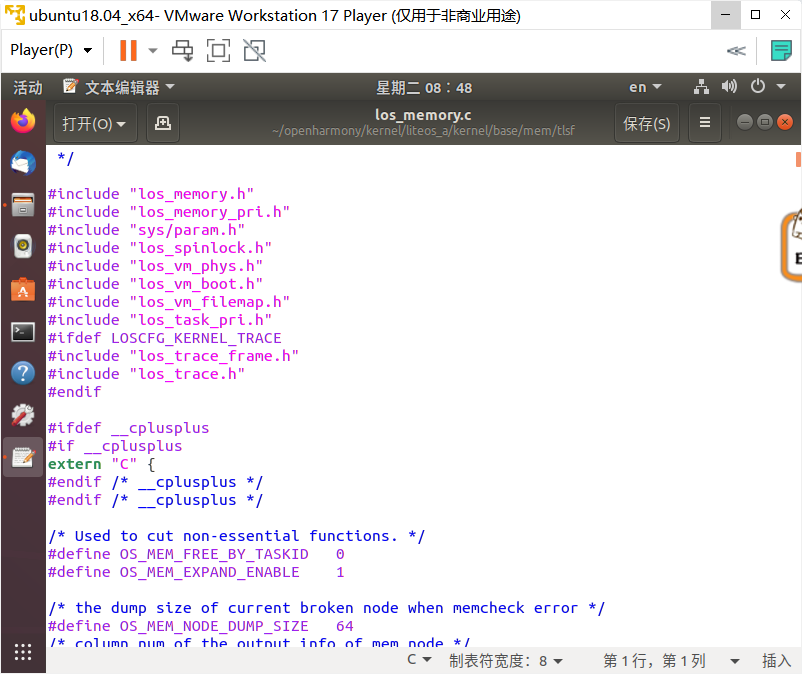
repo init -u https://gitee.com/openharmony/manifest.git -b refs/tags/OpenHarmony\_release\_v1.1.0 --no-repo-verify

repo sync -c -j8



1. 查看内存分配代码

进入/openharmony/kernel/liteos\_a/kernel/base/mem/tlsf/los\_memory.c



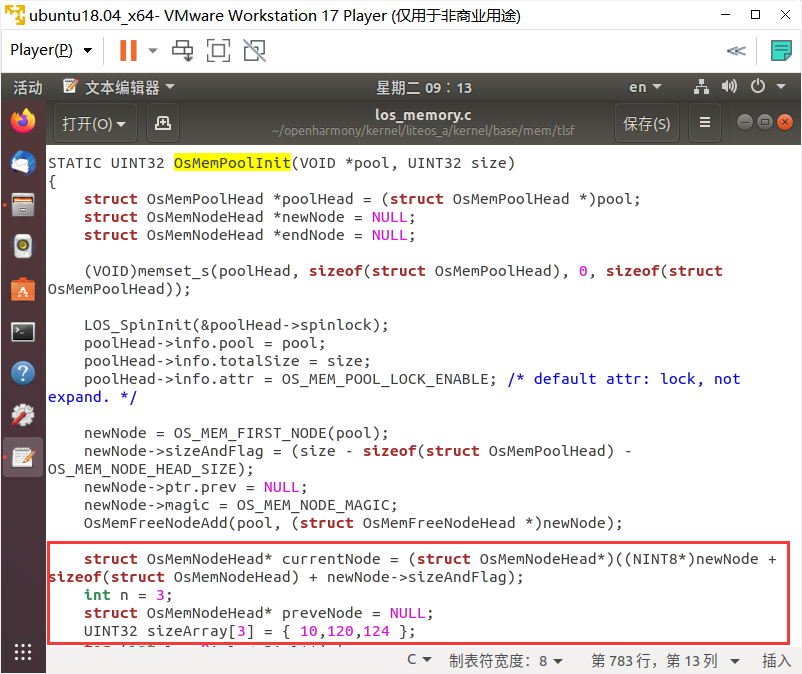
1. 改进TLSF算法

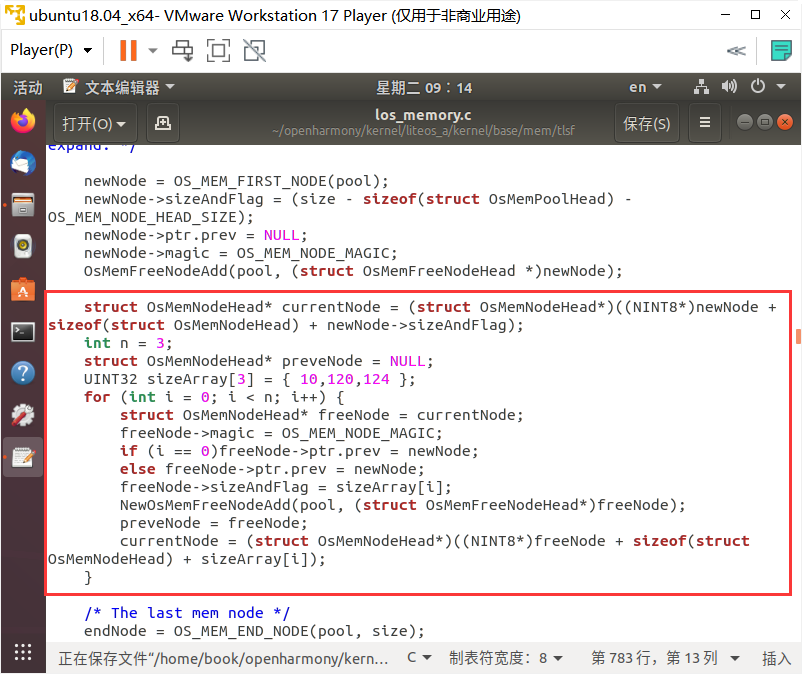
(1)修改OsMemFreeListIndexGet方法，修改前复制一份并改为

NewOsMemFreeIndexGet。



(2)修改OsMemPoolInit方法





(3)修改OsMemFreeNodeAdd方法，将其复制一份作为NewOsMemFreeNodeAdd方法，然后如下修改。



# 实验结果

由于Openharmony1.1版本不适用1.0版本的补丁文件，无法制作成可执行文件，因此上述代码即为结果。

# 实验分析

优化思路：

1.初始化时预先为每个索引中的内存块挂上若干空闲块，在实际分配时避免分割（split）操作，加速分配过程；

2.定位到比当前所需空间更大一级的内存块进行空闲块分配，避免因遍历链表寻找合适大小的空闲块所导致的时间浪费。

# 实验总结

在本次实验中，我们致力于优化Two-Level Segregated Fit (TLSF) 内存分配算法，将原先的Best-fit策略优化为更高效的Good-fit策略，并且进一步降低其时间复杂度至O(1)。通过这项工作，我们旨在提高内存分配算法的效率，减少内存分配时的时间开销，从而优化系统整体性能。

首先，我们对TLSF算法的原理和Best-fit策略进行了深入研究，了解了其内部数据结构和分配算法的工作原理。随后，通过对Good-fit策略的设计和实现，我们成功优化了TLSF算法的内存分配策略。Good-fit策略在寻找合适空闲块时，会选择第一个大小合适的空闲块，而不是像Best-fit那样遍历所有空闲块以寻找最优解，这样就能够大大降低时间复杂度。

在优化过程中，我们还对TLSF算法进行了性能测试和评估。通过对比原有的Best-fit策略和优化后的Good-fit策略在内存分配过程中的耗时情况，我们发现在大多数情况下，Good-fit策略能够显著降低内存分配的时间开销，同时提高系统的整体响应速度。

通过本次实验，我们不仅深入理解了TLSF算法的内部原理和设计思想，还掌握了如何对算法进行有效的优化。同时，我们成功将TLSF算法的时间复杂度由原先的O(log n)优化至O(1)，这为系统的内存管理和性能优化提供了重要的技术支持。这项工作也为我们今后在嵌入式系统和实时系统中的内存管理优化奠定了基础。

# 参考文献

1.[美]William Stallings著，陈向群，陈 渝 等译《操作系统——精髓与原理设计（第八版）》

# 附录

1.OsMemPoolInit方法中的新增代码段

struct OsMemNodeHead\* currentNode = (struct OsMemNodeHead\*)((NINT8\*)newNode + sizeof(struct OsMemNodeHead) + newNode->sizeAndFlag);

int n = 3;

struct OsMemNodeHead\* preveNode = NULL;

UINT32 sizeArray[3] = { 10,120,124 };

for (int i = 0; i < n; i++) {

struct OsMemNodeHead\* freeNode = currentNode;

freeNode->magic = OS\_MEM\_NODE\_MAGIC;

if (i == 0)freeNode->ptr.prev = newNode;

else freeNode->ptr.prev = newNode;

freeNode->sizeAndFlag = sizeArray[i];

NewOsMemFreeNodeAdd(pool, (struct OsMemFreeNodeHead\*)freeNode);

preveNode = freeNode;

currentNode = (struct OsMemNodeHead\*)((NINT8\*)freeNode + sizeof(struct OsMemNodeHead) + sizeArray[i]);

}