Operating System CH9

Susie Glitter

2025 年 7 月 7 日

注：本次实验使用了 VMware 中的 ubuntu-16.04.6-desktop

1 任务一：实现连续内存分配器

1.1 管理内存使用的数据结构

```
typedef struct SEG
{
    int s;
    int t;
    int used;
    char *name;
    struct SEG *prev;
    struct SEG *next;
    struct SEG *smaller;
    struct SEG *bigger;
}seg;
```

我们使用如上数据结构来表示一段连续的内存，其中 s 表示起始地址，t 表示结束地址的下一地址，used 表示该段内存是否被使用，若被使用，则在 name 中储存使用该段地址的进程名字

使用两个带头尾节点的双向列表将各段地址进行串联，其中 prev 与 next 指针对用于按地址大小顺序遍历，smaller 与 bigger 指针用于按空间大小遍历，具体在下文进行说明

1.2 初始化

初始化一块 1MB 的内存，标记为空，连接至头尾节点之间即可

1.3 请求内存

使用请求内存函数 request() 为某一进程请求一定大小的空间，经过某种策略（见后文）选定一块空且大小足够的内存段，进一步使用 request_at() 函数对该内存段进行处理，若大小恰好，直接修改 used 与 name 即可，若需要的空间小于该段大小，则将该内存进行切分，选用第一段进行分配，此时需要维护两个双向链表

1.4 释放内存

使用释放内存函数 release()，按地址进行遍历，释放使用中的对应进程名的内存，这里使用 release_at() 函数，将 used 修改，释放 name 字符串防止内存泄漏，这时检查前后是否有相连的空内存段，若有，进行合并（只用判断一次），并且维护两个双向链表

1.5 信息展示

按照地址大小遍历展示每一个内存段的使用情况

2 任务二：三种内存分配策略

request() 函数根据指定策略决定对哪一块地址使用 request_at() 函数

2.1 First fit

沿 next 指针方向，由小地址向大地址遍历，选择第一个可用地址

2.2 Best fit

沿 bigger 指针方向，由小块向大块内存遍历，选择第一个可用地址

2.3 Worst fit

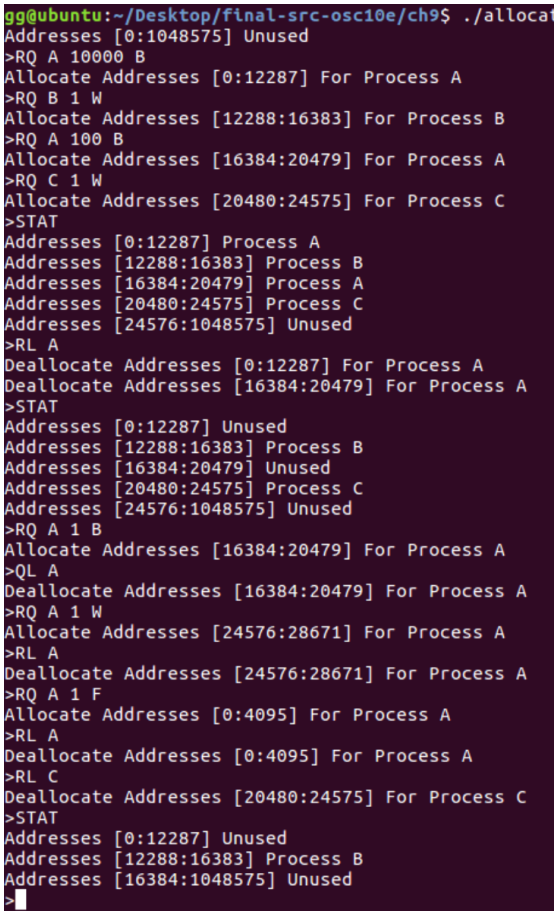
沿 smaller 指针方向，由大块向小块内存遍历，选择第一个可用地址

2.4 任务三：First Fit 与 Worst Fit 的加速

已经加速，原理为使用 bigger 与 smaller 链接的双链表，可以进行有序查找，快速找到第一个符合的内存段

因此在内存段的大小改变时，需要重新维护这一个双链表，时机有两个：申请内存时的切分与释放内存时的合并

3 效果展示



运行内容展示了三种分配策略的区别，展示了三种功能的正常运行，包括空内存段的合并