Report-Project7

付昊源
517021910753

2019 年 6 月 12 日

1 Contiguous Memory Allocation

1.1 程序设计思想

本 project 用于模拟内存的连续分配。留给用户的接口有：为某一个进程申请一段连续的内存块，释放一个连续的内存块、将内存中的空闲空间集中为一整块、以及显示当前内存的整体状态。内存的范围由命令行输入，单位为字节。

在申请内存时，动态分配内存的方式有三种：First-Fit, Best-Fit, Worst-Fit，它们的算法已经在操作系统教材中给予说明，在此不再赘述。分配内存的方式由用户传入参数"F", "B", "W" 指定，因此在申请内存的函数中需要检查这一参数。

释放内存时，用户会输入所释放内存的进程名，只需要将该块空间变为 unused。但需要注意，如果该进程的空间左右也有空闲的空间，释放该进程后需要将连续的两块空闲空间归并为同一块。也就是说，在整个内存空间中始终不应该出现连续的两块空闲空间，任意两块空闲内存空间之间必有进程空间。

将内存中空间集中时，需要将所有进程占用的空间向前平移，从 0 号地址开始使之连续，剩余的空间即为一整块空闲空间。显示当前内存状态很简单，只需要将内存的每一块空间状态进行输出。

1.2 运行结果截图

为了测试连续内存分配，首先我们加入四块进程（P0, P1, P2, P3, P4）并显示状态。接下来，删除 P0, P2 和 P4，为测试 Best-Fit 和 Worst-Fit 创造条件。

```
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project7$ ./allocator 15000
allocator> RQ P0 1000 F
allocator> RQ P1 2000 F
allocator> RQ P2 3000 F
allocator> RQ P3 4000 F
allocator> RQ P4 5000 F
allocator> STAT
Addresses [0:999] Process P0
Addresses [1000:2999] Process P1
Addresses [3000:5999] Process P2
Addresses [6000:9999] Process P3
Addresses [10000:14999] Process P4
allocator> RL P0
allocator> RL P2
allocator> RL P4
allocator> STAT
Addresses [0:999] Unused
Addresses [1000:2999] Process P1
Addresses [3000:5999] Unused
Addresses [6000:9999] Process P3
Addresses [10000:14999] Unused
allocator> RQ P5 500 F
allocator> RQ P6 600 B
allocator> RQ P7 700 W
allocator> STAT
Addresses [0:499] Process P5
Addresses [500:999] Unused
Addresses [1000:2999] Process P1
Addresses [3000:3599] Process P6
Addresses [3600:5999] Unused
Addresses [6000:9999] Process P3
Addresses [10000:10699] Process P7
Addresses [10700:14999] Unused
allocator> RL P6
allocator> STAT
Addresses [0:499] Process P5
Addresses [500:999] Unused
Addresses [1000:2999] Process P1
Addresses [3000:5999] Unused
Addresses [6000:9999] Process P3
Addresses [10000:10699] Process P7
Addresses [10700:14999] Unused
allocator> C
allocator> STAT
Addresses [0:499] Process P5
Addresses [500:2499] Process P1
Addresses [2500:6499] Process P3
Addresses [6500:7199] Process P7
Addresses [7200:14999] Unused
```

图 1: Contiguous Memory Allocation

以 First-Fit 申请 P5，以 Best-Fit 申请 P6，以 Worst-Fit 申请 P7，显示状态后如图所示。回收 P6，测试其是否可以与之后的空闲空间合并。最后，归并所有空闲空间为一个大块并显示状态。

1.3 核心数据结构及代码解释

首先我们定义了一个新的结构体 hole，包含 bool 变量 flag 和 use，flag 用于表示该块内存是否有意义，use 用于表示该块内存是空闲还是被占用；还包含 int 变量 start 和 end，分别表示该块内存的起始地址和结束地址；以及一个 char 数组 name，用于记录该块空间进程的名字，如果该块内存是空闲区域，则名字无意义。

全局变量中我定义了 hole 类型数组 memory 用于模拟内存，以及一个 int 变量用于监视已使用的 hole 数量。

主函数中，首先从命令行读取内存的总空间，先将内存分配为一整块空间，并初始化 memory 中的所有其他元素，使之 flag 和 use 均为 false。接下来从用户的输入中获取用户行为，对于每一种操作指令（RQ、RL、C、STAT、X）分别调用不同函数。

对于 RQ 指令，调用 request() 函数，需要参数有进程名、内存块大小、连续内存分配模式。在 request() 函数中，首先根据连续内存分配的模式，在当前内存中找到符合条件的内存块首地址，如果没有找到，则输出提示语句，结束函数。需要注意的是，多数情况下，一块内存的大小不会正好与进程需要的内存大小相同，但要考虑特殊的情况，从而确定 memory 中下一个元素的 flag 是否使用。

对于 RL 指令，调用 release() 函数，所需的参数为进程名。函数的实现只需在 memory 中找到与进程名相同的进程，并将其释放。同时需要注意，该块内存所对应的 memory 中的元素，假设下标为 i，则还需要判断下标为 $i-1$ 和 $i+1$ 的元素是否为空闲内存，注意空闲内存快的合并。

对于 C 指令，调用 compact() 函数。实现这个函数时，首先遍历 memory 中所有 flag 为 true 的元素，判断其是否为某一进程，若是，则需记录其下标，便于之后的整合。当获取所有的进程元素下标后，在 memory 中改变各个元素，首先安放所有的进程块，最后，将所有其余内存均置为空闲内存，存为一大块，并将 memory 中其余不使用的元素 flag 置为 false。

对于 STAT 和 X 指令，分别为输出当前状态和退出程序，实现方式简单，不再赘述。