CS2106 Introduction to Operating Systems

Semester 1 2021/2022

Tutorial 7

Contiguous Memory Allocation

1. (Fixed partitioning) Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.

   Perform the allocation of processes using:

   a. First Fit Algorithm
   b. Best Fit Algorithm
   c. Worst Fit Algorithm

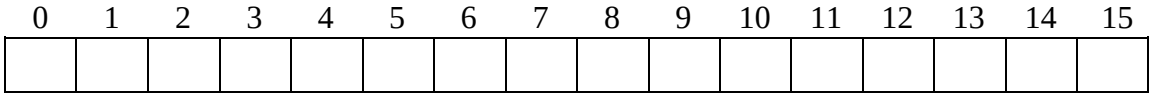
   Which algorithm makes the most efficient use of memory in this particular case? Which algorithm has the best average runtime? Which algorithm is the best to use overall?

2. (Dynamic partitioning) In the lecture, we used linked list to store partition information under the dynamic allocation scheme. One common alternative is to use bitmap (array of bits) instead.

   Basic idea: A single bit represents the smallest allocatable memory space, 0 = `free`, 1 = `occupied`. Use a collection of bits to represent the allocation status of the whole memory space. As a tiny example, suppose the memory size is 16KB and the smallest allocatable unit is 1KB. We need 16 bits (2 bytes) to keep track of the allocation status:

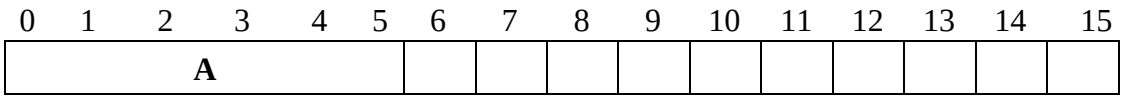
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

   Below is the corresponding physical memory layout at this point. Note that the allocatable unit != partition. At this point the whole memory is a single free partition. The boxes are drawn to illustrate the allocation unit clearly.



   After placing process **A** (6KB), the bitmap and the corresponding physical memory layout become:

1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0



   Give brief pseudo code to:

   a. Allocate X KB using the bitmap using **first-fit**.
   b. Deallocate (free) X KB with start location Y.
   c. Merge adjacent free space.

3. (Buddy System) Consider a buddy memory allocation system that has byte addressing. Memory space starts at `00000` and ends at `11111` = 2^5 = 32 bytes. Smallest allocatable block size is 1 byte. Assume that all possible merges are already done on buddy blocks.

a. Find the largest possible size of each of the free blocks.
b. Find the **minimum** sum of all allocated memory in the system.

The following are memory addresses (in binary) of free blocks:

```
[a] = 00000
[b] = 00110
[c] = 01000
[d] = 01100
[e] = 01110
[f] = 10000
[g] = 11100
```

Hint: Represent the memory as an array of 32 bytes. Split the blocks such that the free blocks will be the above addresses.

4. (Overhead of Bookkeeping Information) Regardless of the partitioning schemes, the kernel needs to maintain the partition information in some way (e.g. linked lists, arrays bitmaps etc). These kernel data, which is the overhead of the partitioning scheme, can consume considerable memory space.

   Given an initially free memory space of 16MB (2^{24} Bytes), briefly calculate the overhead for each of the scheme below. You should try to find a representation that reduces the overhead if possible.

   For simplicity, you can assume the following size during calculation:

   - Starting address, Size of partition or Pointer = 4 bytes each
   - Status of partition (occupied or not) = 1 byte

   a. Fixed-Size Partition: Each partition is 4KB size (2^{12}). What is minimum and maximum overhead?

   b. Dynamic-Size Partition (Linked List): The smallest request size is 1KB (2^{10}), the largest request size is 4KB. What is the minimum and maximum overhead using linked list? Allocations happen in multiples of 1 KB.

   c. Dynamic-Size Partition (Bitmap, see Q1): The smallest request size is 1KB (2^{10}), the largest request size is 4KB. What is the minimum and maximum overhead using bitmap?

Questions for own exploration

5. (Buddy System) Given a 1024KB memory with smallest allocatable partition of 1KB, use buddy system to handle the following memory requests.
    a. Allocate: Process A (240 KB)
    b. Allocate: Process B (60 KB)
    c. Allocate: Process C (100 KB)
    d. Allocate: Process D (128 KB)
    e. Free: Process A
    f. Free: Process C
    g. Free: Process B

    Show the physical memory layout as a way to track the results, i.e. below is the physical memory layout after request (a).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A [256KB (240)]				Free[256KB]				Free [512KB]							

    Note: For a more condensed representation, the numbers on the first row represent the starting address as multiples of 64KB. e.g. Process A starts at **0** x 64KB = 0, the free 256KB partition starts at **4** x 64KB = 256KB, the last partition is at **8** x 64KB = 512KB.

    You should try to maintain the actual array of linked lists (as shown in lecture) throughout to gain a better understanding of the algorithm. For consistency, we assume the partitions of the same size are arranged in ascending order by their starting address.

6. (First fit) Suppose we use the following allocation algorithm: for the first allocation traverses the list of free partitions from the beginning of the list until the first partition which can accommodate the request. The following allocations, however, do not will start from the beginning, but instead start from the partition where the previous allocation was performed. Compare this algorithm with First Fit in terms of runtime and efficiency of memory use.