CSE 2431 **HOMEWORK 2 – SOLUTION** **SPRING 2025**

Electronic Submission: 11:59 pm, Thu Feb 13, 2025
Point: 25 points (5% of total grade)

Notes:
K = 2^{10}; M=2^{20}; G=2^{30}; index always starts from 0 (e.g. page 0 is the first page)

Question 1: Free Space Management [8 points]

Suppose the malloc library is using the mechanism described in our lecture (**Slide 31-38** of **SP25_CSE2431_Topic_5b_Virtual_Memory_Dynamic_Relocation** to manage free space. Assume that **base address is 0 (note that here we start from 0, different from your lecture slide)**, and virtual memory is **given from 0 up to max**. Assume the header **takes 8 bytes**.

Solution:
1KB = 1024 Bytes – 8 bytes (for header) = 1016.
[Size: 1016] ← Header points to 0
[next: 0]
[Empty space….]

1) Following 1), suppose the user calls `malloc(20)`. Draw the memory layout diagram. (Clearly state all the pointers)

Solution:
malloc(20): means that we need to allocate 28 bytes in total (20 for content and 8 for header)
[Size: 20]
[magic: 1234567]
[20 bytes **used**] ← Ptr points to 8

[size: 988] ← **Header points to 28**
[next: 0]
[**Empty** space]

2) Following 2), suppose the user **continue** by calling **`malloc(120)`**. Draw the memory layout diagram. (Clearly state all the pointers)

Solution:
malloc(120) means we need to allocate 128 bytes.
[Size: 20]
[magic: 1234567]
[20 bytes **used**] ← Ptr points to 8

[Size: 120]
[magic: 1234567]
[120 bytes **used**] ← sPtr points to 36

[size: 860] ← **Header points to 156 (28 + 128)**
[next: 0]
[**Empty** space]

3) Following 3), suppose the user **frees** the block from 2). Draw the memory layout diagram.

Solution:
Free the block from (2) means we are freeing the first 28 bytes.
[Size: 20] **← Header points to 0**
[**next: 156**]
[20 bytes **empty**]

[Size: 120]
[magic: 1234567]
[120 bytes **used**] ← sPtr points to 36

[size: 860]
[next: 0]
[**Empty** space]

Question 2: Memory Allocation [6 points]
Suppose we have 1000K of memory where the first 100K is reserved. Also suppose that the system uses contiguous allocation, and the following processes have been allocated memory in the following order:

P1: 100K, P2: 100K, P3: 25K, P4: 200K, P5: 200K, P6: 75K, P7: 100K.

Create a diagram of memory (ignore header information to maintain free space).

[0 - 100K]	Reserve
[100K - 200K]	P1
[200K – 300K]	P2
[300K – 325K]	P3
[325K – 525K]	P4
[525K – 725K]	P5
[725K – 800K]	P6
[800K - 900K]	P7
[900K - 1000K]	**[Empty]**

Suppose that P2, P4, and P6 finish, and the following processes are on the queue waiting for memory (assume FIFO):

P8: 150K, P9: 25K, P10: 50K, P11: 25K.

When P2, P4, and P6 finished:

[0 - 100K]	Reserve
[100K - 200K]	P1
[200K – 300K]	**[Empty]**
[300K – 325K]	P3
[325K – 525K]	**[Empty]**
[525K – 725K]	P5
[725K – 800K]	**[Empty]**
[800K - 900K]	P7
[900K - 1000K]	**[Empty]**

Create a diagram of memory for each of the following algorithms showing how the memory would be allocated:

a) First-Fit Policy

P8 will be fit in first, at [325K-475K]. Then followed by P9, P10, P11. Here we **luckily have** the free segment that fits all P9, P10, P11. But this is not always the case.

[0 – 100K]	Reserve
[100K – 200K]	P1
[200K – 225K]	**P9**
[225K – 275K]	**P10**
[275K – 300K]	**P11**
[300K – 325K]	P3
[325K – 475K]	**P8**
[475K – 525K]	**[Empty - 50K]**
[525K – 725K]	P5
[725K – 800K]	**[Empty – 75K]**
[800K - 900K]	P7
[900K - 1000K]	**[Empty – 100K]**

b) Best-Fit Policy

[0 - 100K]	Reserve
[100K - 200K]	P1
[200K – 300K]	**[Empty]**
[300K – 325K]	P3
[325K – 475K]	**P8**
[475K – 500K]	**P9**
[500K – 525K]	**P11**
[525K – 725K]	P5
[725K – 775K]	**P10**
[775K – 800K]	**[Empty]**
[800K - 900K]	P7
[900K - 1000K]	**[Empty]**

c) **Worst-Fit Policy**

[0 - 100K]	Reserve
[100K - 200K]	P1
[200K – 225K]	P9
[225K – 250K]	P11
[250K – 300K]	**[Empty]**
[300K – 325K]	P3
[325K – 475K]	P8
[475K – 525K]	**[Empty]**
[525K – 725K]	P5
[725K – 800K]	**[Empty]**
[800K - 900K]	P7
[900K – 950K]	P10
[950K - 1000K]	**[Empty]**

****** Note:*** *For worst-fit policy, all the resulted free memory segments are very small. Now we cannot fit any Process requiring 100K of memory!*

Notes:

- ***Assume that the algorithm starts at the smallest address of memory to determine each new allocation for First-Fit.***
- ***If there is a 'tie', that is, two locations in memory where a process could be loaded, use the smaller address.***

Question 3: **Address translation (Related to Question 2)** **[3 points]**
Following Question 2, suppose the system uses Best-Fit, translate the following virtual addresses into physical addresses:

1) Address 1K of P1
   Solution: Base = 100K, bounds = 100K, so 1K is **valid**. **PA = 100K + 1K = 101K**
2) Address 1K of P8
   Solution: Base = 325K, bounds = 150K, so 1K is **valid**. **PA = 325K + 1K = 326K**
3) Address 50K of P9
   Solution: Base = 475K, Bounds = 25K, so 50K is **invalid**. **Error!**

Question 4: **Paging (Basic)** **[8 points]**
Suppose a process can have a max virtual memory of 4KB, and a machine has a physical memory of 64KB. Suppose page/frame size is 1KB.

1) What is the minimal number of bits for virtual address (VA) and for physical address (PA)?
   Solution: **12 bits for VA** and **16 bits for Physical Address (PA)**
2) For a virtual address, how many bits should be used as offset and how many are for virtual page number (VPN)?
   Solution: **10 bits for offset** and **2 bits for VPN**
3) For physical address, how many bits should be used as offset and how many are for physical frame number (PFN)?
   Solution: **10 bits for offset** and **6 bits for PFN**
4) Suppose the following page table: page 0 -> frame 3; page 1 -> frame 4; page 2 -> frame 1; page 3 -> frame 10. Translate the following virtual addresses into physical addresses
   a. 0
   b. 1024
   c. 2000
   d. 4096

   Solution:
   0 (page 0, offset 0) → (frame 3, offset 0) = **3K (or 3072)**
   1024 (page 1, offset 0) → (frame 4, offset 0) = **4K (or 4096)**
   2000 (page 1, offset 976) → (frame 4, offset 976) = **4K + 976 = 5072**
   4096 (page 4, offset 0) → **Invalid. Error (page fault)**