COP 3503 Programming Fundamentals for CIS Majors II, Fall 2014
Programming Assignment 2

Out: Oct. 06 (Monday), 2014

Due: 5pm on Oct. 20 (Monday), 2014

Problem Description

Write a C++ program that emulate the operating system's responsibility of allocating memory to certain programs. This will be a very simple page-based view of memory management. On startup, your program will have some 32 pages of contiguous, unused memory. Each page will be 4 KB long.

It should then allow the users (i.e., your TAs) to "run" programs that require chunks of this memory for some period of time. It should also allow the users to "kill" programs (i.e., "Ctrl-c" or "kill -9" in most OSs) that are in progress. The pages used by programs that are killed can then be re-used for future programs.

The purpose of this project is two-fold. First, you need to get practice with linked lists and pointers. Secondly, you should examine critically the results of different algorithms of allocating memory on the "fragmentation" of the memory.

Details

1. (Data Structures:) You are required to implement a linked-list data structure to represent the memory of the operating system. It is suggested that each node of the linked list should represent some "chunk" (page) of memory.

2. One strong suggestion is to use 2 linked lists in your program: one to maintain the free space and one to maintain the used space. Initially, the free space contains a single node representing the entire memory. The used list is then empty. When a program is added, you then have to "split" the node in the free list and add a new node to your used list.

3. Another way: Every node in your list is a page of memory with a given sequence number (Page 0 $\rightarrow$ Page 1 $\rightarrow$ Page 2, etc). So, when your program starts, your free list should contain all 32 pages in sequence. As more programs are added, the pages move from the free list into the used list. Note that the order of the lists must be maintained! [Note: This approach works OK for small memory size, but it will have scalability issue when the main memory is reaching gigabyte range (e.g., $1GB/4KB = 250,000$ page-nodes)]

4. (Algorithms:) When your OS/MM program is launched, it needs to decide which algorithm to allocate the **contiguous** memory for programs. To solve this issue, you are required to implement the best-fit and worst-fit algorithms when choosing pages.

5. When a program is "killed", your OS needs to reclaim that memory. This means that you will need to add the memory used by that program to your free list. Note that if there is free memory immediately adjacent to this memory, you may need to combine them.

6. (Computation:) At any given point, the users should be able to query your OS to get the number of "fragments" in memory. You will also certainly want some way of printing out your memory to the screen to ease the debugging process.

Example Run

Try to keep your output as close to the given format as possible:

```
> ./a.out worst
Using worst fit algorithm

    1. Add program
    2. Kill program
    3. Fragmentation
    4. Print memory
    5. Exit

choice - 1
Program name - P1
Program size (KB) - 8

Program P1 added successfully: 2 page(s) used.

choice - 1
Program name - P2
Program size (KB) - 7

Program P2 added successfully: 2 page(s) used.

choice - 1
Program name - P3
Program size (KB) - 9
```

```
Program P3 added successfully: 3 page(s) used.

choice - 3

There are 1 fragment(s).

choice - 4

P1   P1   P2   P2   P3   P3   P3   Free
Free Free Free Free Free Free Free Free
Free Free Free Free Free Free Free Free
Free Free Free Free Free Free Free Free

choice - 2
Program name - P2

Program P2 successfully killed, 2 page(s) reclaimed.

choice - 3

There are 2 fragment(s).

choice - 4

P1   P1   Free Free P3   P3   P3   Free
Free Free Free Free Free Free Free Free
Free Free Free Free Free Free Free Free
Free Free Free Free Free Free Free Free

choice - 1
Program name - P4
Program size (KB) - 3

Program P4 added successfully, 1 page(s) used.

choice - 4

P1   P1   Free Free P3   P3   P3   P4
Free Free Free Free Free Free Free Free
Free Free Free Free Free Free Free Free
```

```
Free Free Free Free Free Free Free Free

choice - 1
Program Name - P1
Program Size (KB) - 5

Error, Program P1 already running.

choice - 1
Program name - P5
Program size (KB) - 1000000

Error, Not enough memory for Program P5

choice - 5
```

Hints

1. You should get your linked list data structure working well before you try to move on to the memory management part. The memory management will inevitably just make use of certain methods in your linked list class.

2. Be wary of bad user input. Think about many test cases when the users could mess up (reasonably).

3. The TAs will perform at least two runs. One run is for the best-fit algorithm, and the other is for the worst-fit algorithm. The fragmentation can be different when different algorithms are used for the same add/kill-scenario.

Submission Guide-lines

1. You must finish this assignment with your individual effort. Your C++ source code file MUST be named as "pa2.cpp" and C++ header/class file should be named as "pa2.h".

2. Please test your program via g++ compiler (i.e., g++ -Wall) on the CISE machines to make sure it runs correctly.

3. Please upload the source code file(s) to SAKAI system as the attachment(s). Please submit the source code file(s) ONLY. NO need to compress the source code file(s) if the size is small.

4. Make sure you submit your assignment BEFORE the deadline. Late submission will NOT be graded !!

Grading Criteria

1. Successful Compilation (30%): Your source code should be able to compile using "g++ -Wall" command without any error or warning. The output should be a valid executable. See lab tutorial for the commends. Please note that we will be using g++ compiler on Linux to grade your programs. If you are using other compilers or IDE (e.g., Visual C++), it is recommended that you test the source codes with g++ before the SAKAI submission (i.e., make sure there is no warning).

2. Program Correctness (40%): The executable should be able to run correctly by giving out the required output.

3. Programming Style (30%): Good coding style is a key to efficient programming. We encourage you to write clear and readable codes. You should adopt a sensible set of coding conventions, including proper indentation, necessary comments and more. Here are some guidelines of good programming style: http://en.wikipedia.org/wiki/Programming_style

Final Notes

1. AGAIN, remember to start the programming assignments as soon as possible. Unlike the conventional assignments, programming assignments sometimes take un-predictable amount of time to finish. Thus, have the code running first, then polish it later with the extra time before the deadline.

2. Remember you should always write your own code and never copy-and-paste from other students' work or other sources. There are indeed many tools (like Stanford Moss) to detect the code similarity.

3. Programming assignments are usually designed by TAs. If you have any question or concern, please feel free to contact TAs. Our goal is to let you experience the fundamentals of computer science. If you like the programming experience, you are one of us !!

4. HAPPY CODING ... and GOOD LUCK !!