CSCE-313 Fall 2015 MP1: High Performance Linked List.

Due: Sun Sep20 11:59pm, 2015

Objective

In a traditional linked list implementation, memory is allocated for each newly inserted item. Again, when some item is deleted, the memory allocated for it is freed and given back to the system. Each memory allocation/de-allocation request interacts with the system memory manager and thus puts overhead on the linked list insertion/deletion operations. To solve this problem, we will develop a high performance linked list that will avoid interacting with the memory manager on a per-operation basis.

Implementation Details

We will obtain/reserve a fixed amount of memory (i.e., M bytes) from the memory manager in the beginning, which we call Memory Pool (MP). The unit of memory is given by a variable called *basic block size* and denoted by variable b. Thus, each item in the list takes b bytes and there can be at most $m = M/b$ items in the list. Further insertion requests will be rejected by the linked list. Keep a Head Pointer (HP) and a Free Pointer (FP) for managing the list, where the former points to the head of the list and the latter points to where the next insertion will happen.

Each linked list item can be separated into two sections – a header and a payload. The header contains necessary information for maintaining the list (e.g., the next pointer, the previous pointer for doubly linked list). On the other hand, the payload portion consists of a key-value pair, where the *key* is a 4 byte integer and the *value* is of variable length, but has a maximum size that is determined by the header and key size. Observe Fig. 1 that shows how a singly linked list should be organized. In the top, a linked list item is shown in detail. Note that the size of pointers depend on the machine/OS type (i.e., in a 64-bit machine, it is 8 bytes).

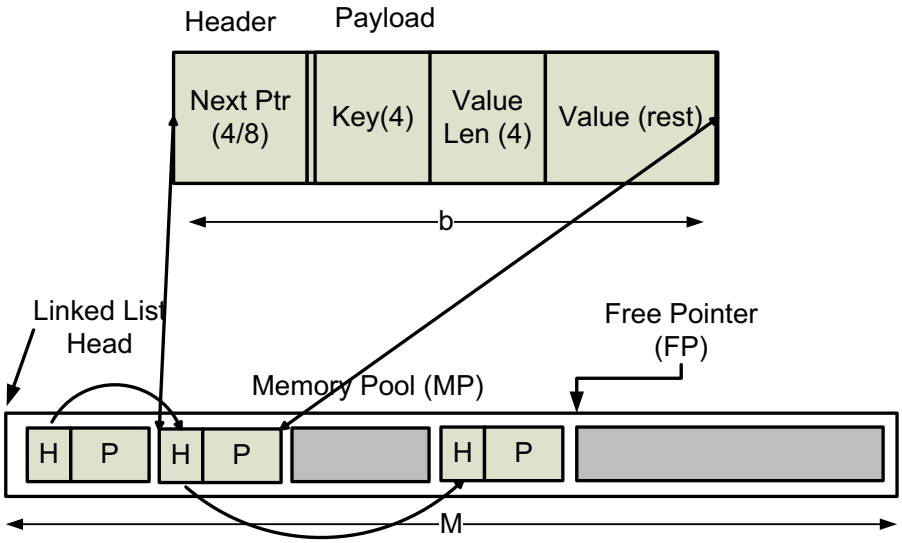

Figure 1: Structural view of a linked list in memory

Assignment: Part 1 – Singly-Linked List

You are to implement a singly linked list with the above mentioned features in C/C++.

- There will be 3 files in your program – `main.c` (contains main function), `linked_list.h` and `linked_list.c`. We will provide a sample main file that you will have to finish.
- Your implementation should contain at least the following functions (declared in the .h and defined in the .c file):
  - `Init(int M, int b)` – initially obtains M bytes of memory by calling library function `malloc()` and initializes the linked list
  - `Destroy()` – destroys the linked list and returns memory to the system by calling another library function `free()`
  - `Insert(int x, char * value_ptr, int value_len)` – inserts a key-value pairs where the key is an integer x and the value is some data pointed by the pointer value_ptr of length value_len. You should use library `memcpy()` function to copy the value rather than just copying the pointer. Each insert should consume b bytes from the MP
  - `Delete(int x)` – deletes the first item from the beginning with the key x
  - `Lookup(int x)` – finds out the key in the list and gives a pointer back
  - `PrintList()` – prints all the items' key-value sequentially starting from the head of the list. Print only the key and the value-length, do not print the actual value, because the value could contain binary/non printable data
- You will write a program called **`testlist`**, which reads the basic block size and the memory size (in bytes) from the command line, initializes the memory, makes some insertion/deletion calls, prints the list using the PrintList() function, and then destroys the allocated memory (using Destroy()). Note that all these will go inside your main function.
- Use the `getopt()` C library function to parse the command line for arguments. The synopsis of the `testlist` program is of the form:

```
testlist [-b <blocksize>] [-s <memsize>]
```

`-b <blocksize>`	Defines the basic block size b in bytes. Default is 128 bytes.
`-s <memsize>`	defines the size of the memory to be allocated, in bytes. Default is 512kB.

- Make sure that your program does not crash in any case. Here are a number of scenarios that your program should take care. In these cases, your program should just print an error, skip the given instruction, and continue to work (i.e., do not exit).
  - Deleting non-existent keys from the list
  - Trying to insert keys after the given memory is full
  - Trying to insert values that do not fit in the payload section

Assignment: Part 2 – Stratified Linked List

Modify the implementation in Part 1 to make a multi-tiered linked list that groups keys into a number of disjointed intervals and keeps numbers from those intervals in separate lists. Take another integer *t* as input that indicates how many levels/tiers you will have in the tiered linked list and divide the number space (i.e., $[0, 2^{32} - 1]$ for unsigned integers, but $[0, 2^{31} - 1]$ for signed integers, which we will be using here, no need to use negative numbers) into *t* regions. The default value for *t* is 16. Fig. 2 shows how this is organized in a 4-tier list. In addition, all numbers in tier *i* should be less than those in tier i+1. Thus, taking modulus operation on a number to choose which tier it goes to is not allowed. You should be using division/bit shift operations. The following are the required tasks (in either C or C++):

- There will be 3 files in your program – main.c, linked_list2.h and a linked_list2.c. Again, we will provide a sample main file that includes some test cases.
- Provide implementation for all the functions in part 1. Note that those functions will change in the way they work. In some cases, the function arguments will also change. For example the Init(M,b) function will change to `Init(M,b,t)` which should now create *t* tiers. Furthermore, the PrintList() will change as well. Do not print empty tiers (i.e., where no insertion happened)
- Name your program **`testlist2`** and this time, your program should accept the following command arguments:

```
testlist2 [-b <blocksize>] [-s <memsize>] [-t <tiers>]
```

REPORT DOCUMENTATION

Provide a report in PDF format describing your findings in both Parts 1 and 2 (note there is no separate report for Part 1). **For Part 1**, 1) do you notice any wastage of memory when items are deleted? If so, can we avoid such wastage and how? 2) Can you think of scenario, where there is space in the memory, but no insertion is possible? 3) What is the maximum size of the value when the pointers are 8 bytes? **For Part 2**, derive a general expression for the range of numbers that go into the i-th tier.

Figure 2: Organization of a tiered linked list

Assignment Submission Instructions

Submit a zipped folder that contains two folders named MP1part1 and MP1part2, and a report in PDF "MP1.pdf". Both folders (MP1part1 and MP1part2) should contain 3 c/cpp/h files. Demonstrate your work during lab meetings. Make sure that your program runs in CSE department's Linux server (linux.cse.tamu.edu).