Bilkent University
Department of Computer Engineering

CS 342 Operating Systems (Spring 2004)

Project 4 - Heap Management Library

Assignment Date: April 15, 2004
Due Date: April 29, 2004

1 Motivation and Goals

- *To design and implement a heap management C library.*
- *To learn how to build a C libary and how to link programs with a C library.*
- *Exercise with memory fit algorithms.*

The project has to be done on Linux platform using C language. The project has to be done individually.

2 Description

In this project, you will design and implement a heap management library, whose external interface is given with the header file included at the end of the section. We should be able to link a program with your library and use your library routines to allocate and free memory space dynamically from the heap segment of the program. In other words, your library routines will provide a functionality similar to `malloc()` and `free()` routines we have seen and used so far in our C programs to allocate memory dynamically.

We now describe each of the routines that are declared in the external interface (`heap.h`) of the heap library.

int heap_init()

This routine will be used by an application programmer (i.e. someone who would like to use your library) to initialize the heap. The job of this routine is to move the `end of data segment` pointer of the program to an address which is multiple of PAGESIZE (4096) [1]. This address will be the start address of the heap. You will then start growing the heap from this address when the programmer would like to allocate more memory.

This routine will also intialize other internal structures that your may want to use to manage the heap segment of a program.

void heap_info(struct heap *hp)

This routine will be used by a programmer to get information about the heap of the process. The information will be copied to the structure pointed by `hp`.

void *heap_malloc(usigned int n)

This routine will be called by a programmer to dynamically allocate a memory block of size `n` bytes. A pointer to the allocated memory block is returned as the return value of the function. If you can not allocate memory of requested size for some reason, you should return NULL.

[1] This part is already implemented as you can see in `heap.c`

As part of the implementation of this function, if you could not find an empty block of size at least `n` bytes, you should grow the heap if possible. How much to grow the heap depens on the value of `n`. If `n` is less than PAGESIZE, you should grow the heap by one page (PAGESIZE bytes). In general you should grow the heap by an amount which is equal to: `((n-1) / PAGESIZE) + 1` . Here `/` is integer division.

The parameter `n` has to be greater than zero to attempt to allocate memory.

void heap_free(void *p)

This routine will be called by a programmer to free a memory block pointed by `p` and allocated earlier dynamically with a call to `heap_malloc`.

void heap_print()

This routine will be called by a programmer to print information about the current state of the heap. When this routine is called, the heap may be in a state where it may constist of a sequence of memory blocks where each block is either empty or allocated. This routine should print out the following information about each of the blocks (empty or used):

`<BLOCK_TYPE> <START_ADDRESS> <SIZE>`

Here, `<BLOCK_TYPE>` is either `U` or `F`. `U` denotes a used block, `F` denotes a free block. For example, if we have 3 blocks, two of which are used and one of which is free, we can have such an output:

```
U    0   100
U  100    50
F  150   200
```

void heap_dump()

This routine will be called to get a hexadecimal dump of the heap (a sequence of bytes). Each byte value (an unsigned value) should be printed out in hexadecimal form. For example, if we have a very small heap of 6 bytes with values: 0x12 0x23 0x00 0xa5 0xbb 0xff, then the output should be as the following:

```
12 23 00 a5 bb ff
```

The output should be byte values in hexadecimal notation seperated by space characters (no newline character should exist inside the output).

int heap_getnextblock(int from_start, struct memblock *mblkp);

This routine will be used by an application programmer to traverse the memory blocks (used or free) that make the heap. The internal implementation of the blocks and the heap is hidden from the programmer. What the programmer get with this call is information about one memory block that is indexed by a current pointer (an integer or an address). The current pointer is also hidden to the application programmer.

If programmer calls this function with `from_start` being 1, then the information about the first block is copied to the structure pointed by `mblkp`, and the current pointer is incremented so that it now points to the next memory block in the heap.

If the progmmer calls this function again with `from_start` being 0 this time, then the information about the block pointed by the current pointer is copied into the structure pointed by `mblkp`.

After copying information about the current block, the current pointer is advaned. Ifthe copied block was the last block of the heap, then the current pointer will have a special value denoting the end of blocks. If programmer calls this function in this state, the return value of the function will be 0.

If the programmer calls this function with `from_start` being 0 and the current pointer pointing to the last block (which can be used or free), then the function should not copy any information, since there is no next block. The function should return a value of 0 in this case (denoting the end of blocks).

If there is an error condition occured, the function should return -1. Otherwise, if the function has copied information to the structure pointed by `mblkp` successfully, the return value should be 1 (success).

Next, we provide the content of the external interface of the heap libary, i.e. the content of the header file `heap.h`.

```
#define PAGESIZE 4096   /* growing of heap should in
   increments that is multiple of
   PAGESIZE, i.e. we can grow
   heap by k * PAGESIZE, where k >= 1.
*/
#define MEMBLK_FREE  1
#define MEMBLK_USED  2

/*
   structure to store information about a memory block in
   heap. can be used by application programmer.
*/
struct memblock
{
  int    type;   /* type of memory block - free or used */
  ulong  start;  /* start address of the memory block */
  long   size;   /* size of memory block  */
};

/*
   structure to store information about heap. can be used
   by an application programmer.
*/
struct heap
{
  ulong  start;         /* type could also be void* */
  ulong  end;
  long   size;
  int    num_memblks;
  int    num_freeblks;
  int    num_usedblks;
};
```

```
/*****************************************************************
              INTERFACE FUNCTION PROTOTYPES
*****************************************************************/

int    heap_init();   void    heap_info(struct    heap   *hp);    void
*heap_malloc(uint    size);   void    heap_free(void    *addr);   void
heap_print(); void  heap_dump(); int heap_getnextblock(int from_start,
struct memblock *mblkp);
```

3 Other Attachements of the Project

The website contains some source files that will be useful for you. The folowing are included in the website:

- the header file `heap.h`
- a C template file `heap.c` which will contain the implemetation of the routines described above.
- a program template `heaptest.c` that can include the calls to the functions implemented in the libray
- a Makefile to compile the library and test program and link them together.

4 Submission Requirements

We will post them later on the course website.