Computer Science 159.335 - Operating Systems & Concurrent Programming Assignment 3 Due 12th October 2015

Lecturer: Martin Johnson

The purpose of this assignment is to write memory allocation functions to replace malloc and free.
Available on stream is a program called a3-2015.c. It is a simple memory allocator. The important parts of it are shown below:

```c
#include <stdio.h>
#include <time.h>
// 159.335 assignment 3
// This is a working memory allocation program
// but it is slow and uses lots of memory.
// Martin Johnson
// the following is fixed by the OS
// you are not allowed to change it
#define PAGESIZE 4096
// you may want to change the following lines if your
// machine is very fast or very slow to get sensible times
// but when you submit please put them back to these values.
#define NO_OF_POINTERS 2000
#define NO_OF_ITERATIONS 200000
// change the following lines to test the real malloc and free
#define MALLOC mymalloc
#define FREE myfree
// you are not allowed to change the following function
void *allocpages(int n) { // allocate n pages and return start address
   return VirtualAlloc(0,n * PAGESIZE,4096+8192,4);
}
// you are not allowed to change the following function
int freepages(void *p) { // free previously allocated pages.
   return VirtualFree(p,0,32768);
}
void *mymalloc(int n) { // very simple memory allocation
   void *p;
   p=allocpages((n/PAGESIZE)+1);
   if(!p) puts("Failed");
   return p;
}
int myfree(void *p) { // very simple free
   int n;
   n=freepages(p);
   if(!n) puts("Failed");
   return n;
}
unsigned seed=7652;
int myrand() { // pick a random number
   seed=(seed*2416+374441)%1771875;
   return seed;
}
```

```c
int randomsize() { // choose the size of memory to allocate
   int j,k;
   k=myrand();
   j=(k&3)+(k>>2 &3)+(k>>4 &3)+(k>>6 &3)+(k>>8 &3)+(k>>10 &3);
   j=1<<j;
   return (myrand() % j) +1;
}
int main() {
   int i,k;
   unsigned char *n[NO_OF_POINTERS]; // pointers to allocated memory
   int size;
   int s[5000]; // used to store sizes when testing
   int start_time;
   int start_mem;

   for(i=0;i<NO_OF_POINTERS;i++) {
      n[i]=0;     // initially nothing is allocated
   }

   start_time=cputime();
   start_mem=memory();
   for(i=0;i<NO_OF_ITERATIONS;i++) {
      k=myrand()%NO_OF_POINTERS; // pick a pointer
      if(n[k]) { // if it was allocated, then free it
         // check that the stuff we wrote has not changed
         if(n[k][0]!=(unsigned char)(n[k]+s[k]+k))
            printf("Error when checking first byte!\n");
         if(s[k]>1 && n[k][s[k]-1]!=(unsigned char)(n[k]-s[k]-k))
            printf("Error when checking last byte!\n");
         myfree(n[k]);
      }
      size=randomsize(); // pick a random size
      n[k]=(unsigned char *)mymalloc(size); // do the allocation
      s[k]=size; // remember the size
      n[k][0]=(unsigned char)(n[k]+s[k]+k); // put data in the first and
      if(size>1) n[k][size-1]=(unsigned char)(n[k]-s[k]-k); // last byte
   }
   // print some statistics
   printf("That took %.3f seconds and used %d bytes\n",
         ((float)(cputime()-start_time))/1000,
         start_mem-memory());
   return 1;
}
```

This program uses two win32 kernel functions to implement malloc and free. The function "allocpages" allocates an integer number of 4K pages and returns the start address. The function "freepages" frees a previously allocated set of pages. In this program, mymalloc simply calls allocpages for every allocation. This is wasteful of memory because 4K will be allocated even if the malloc is for a single byte. It is also slow because each allocation requires changes to the memory management information for the process.

Your task is to rewrite mymalloc and myfree using only calls to allocpages and freepages and any global variables you wish. If you are using a very fast or slow machine you may want to change the number of iterations in the loop. You can not use malloc, free, new or similar memory allocators anywhere in your program. All memory allocation must be done after the line which says: start_mem=memory(). It is acceptable to allocate a single large block of memory the first time mymalloc is called and then return portions of this block (never calling freepages). To see how well Microsoft can write a malloc, change **`#define MALLOC mymalloc`** to **`#define MALLOC malloc`** and **`#define FREE myfree`** to **`#define FREE free`**. You should be able to get similar or better results.

Submit your assignment electronically using stream. Your submission must be a single source file written in ANSI C. Please do not include any extra header files. Your program should include comments as documentation. Do not open any external data files. Your program must compile using the GCC compiler in the labs. You are not allowed to work in groups.

This assignment will be worth 10% of the complete paper. Marks will be awarded for fast programs that are not wasteful of memory. Marks will be subtracted for obvious copying and for late submissions.

M Johnson 2015