CS360 Midterm 2 - March 28, 2019 - James S. Plank

Put all answers on the answer sheet. In all of these questions, please assume the following:

- Pointers and longs are 4 bytes.
- Don't worry about header files in any of this work. Just assume that they are there.
- The machine is little endian: If an integer is 0xabcdef89, then its 1st byte is 0x89, and its last byte is 0xab.
- There are no segmentation violations or bus errors in any of this code.

Question 1

Behold the following procedure:

```
int write_double_array(int fd, int *ids, double *vals, int n)
{
    int i;

    for (i = 0; i < n; i++) {
        if (write(fd, ids+i, sizeof(int)) != sizeof(int)) return -1;
        if (write(fd, vals+i, sizeof(double)) != sizeof(double)) return -1;
    }
    return 0;
}
```

Part A: Suppose n is large (over 1 million). Why is this procedure going to run exceptionally slowly? I don't want a one-sentence answer here – give me three or four sentences, and maybe even some numbers to illustrate!

Part B: Your job is to rewrite **write_double_array()** so that it runs much faster. You are not allowed to use the standard I/O library, and your procedure must have the same output as the original. Limit your extra memory usage to no more than 12,000 bytes.

In this part, give me a few sentences to describe how you're going to do this rewrite. The point of this part is for you to communicate to me that you know what you're doing, without having to write code.

Part C: Now write the code.

Part D: Give me two examples of why **write_double_array()** as written above would return -1.

Question 2

Write the procedure **hlqef**, with the following prototype:

```
int hleqf(char *f, char *d);
```

In this procedure, **f** is the name of a file, and **d** is the name of a directory. **hleqf()** should return the number of files in the directory **d** that are hard links to **f**. Note, **f** does not have to be in **d**. If it is, you can simply include it in your number of files with hard links to **f**. You may assume that the filenames inside of a directory are no longer than 255 bytes.

If you encounter any errors, simply return -1.

CS360 Midterm 2 - March 28, 2019 - James S. Plank

Question 3

When we execute the following snippet of code, we know the following things:

- Pointers are four bytes.
- **malloc()** and **free()** work as described in lecture notes.
- The user has called **malloc()** and **free()** a bunch.
- **sbrk(0)** will return 0x5268.
- The free list is doubly linked and NULL terminated.
- The heap starts at address 0x5000.
- The pointer to the head of the free list is at address 0x4000. (It's a global variable, so it is not on the heap).
- Every byte between 0x5000 and the end of the heap is either allocated or free.
- The value at address 0x4000 is 0x5100.
- The value at address 0x5000 is 0x100 (256).
- The value at address 0x5100 is 0x80 (128).
- The value at address 0x5104 is 0.
- The value at address 0x5108 is 0.
- The value at address 0x5200 is 0x68. (104).

Part A: Please answer the following questions. You may answer in decimal or hexadecimal. If you answer in hexadecimal, then use "0x" please.

- **A-1**: How many bytes are in the heap?
- **A-2**: How many bytes are on the free list?
- **A-3**: What is the minimum number of allocated chunks in the heap?
- **A-4**: What is the smallest value of x for which **malloc(x)** will require us to call **sbrk()**?

Part B: Now, suppose I call **free(0x5208)**. Please tell me which values in memory change, and what their values change to. Simply list them in the table on the answer sheet.

Question 4

Please give me the "jassem" assembly code for the following procedures. As always, do not optimize.

```
int a(int *p, int b)
{
  return p[b];
}
```

```
void c()
{
  int d;

  d = e(4) + f(5, 6);
}
```

```
int g()
{
  int h[20];

  h[10] = 5;
  return (int) &(h[10]);
}
```

Useful Typedefs and Prototypes

You don't need to worry about putting the proper include files in the code that you write.

```
char *strcpy(char *to, char *from);
char *strcat(char *to, char *from);
void *memcpy(void *to, void *from, int num);

struct stat {
          mode_t   st_mode;     /* File mode (see mknod(2)) */
          ino_t    st_ino;      /* Inode number */
          nlink_t  st_nlink;    /* Number of links */
          uid_t    st_uid;      /* User ID of the file's owner */
          gid_t    st_gid;      /* Group ID of the file's group */
          off_t    st_size;     /* File size in bytes */
          time_t   st_atime;    /* Time of last access */
          time_t   st_mtime;    /* Time of last data modification */
          time_t   st_ctime;    /* Time of last file status change */
         /* Plus some other stuff */
};

struct dirent {
        char d_name[256];    /* name must be no longer than this */
        /* Plus other stuff */
};

int     open(const char *path, int flags, mode_t mode);
int    close(int d);
size_t  read(int d, const void *buf, size_t nbytes);
size_t write(int d, const void *buf, size_t nbytes);
int     stat(const char *path, struct stat *sb);
mode_t umask(mode_t numask);

DIR           *opendir(const char *filename);
struct dirent *readdir(DIR *dirp);
long           telldir(DIR *dirp);
void           seekdir(DIR *dirp, long loc);
void         rewinddir(DIR *dirp);
int           closedir(DIR *dirp);

extern JRB make_jrb();   /* Creates a new rb-tree */

extern JRB jrb_insert_str(JRB tree, char *key, Jval val);
extern JRB jrb_insert_int(JRB tree, int ikey, Jval val);
extern JRB jrb_insert_dbl(JRB tree, double dkey, Jval val);
extern JRB jrb_insert_gen(JRB tree, Jval key, Jval val, int (*func)(Jval,Jval));

/* The following return NULL if there is no such node in the tree */

extern JRB jrb_find_str(JRB root, char *key);
extern JRB jrb_find_int(JRB root, int ikey);
extern JRB jrb_find_dbl(JRB root, double dkey);
extern JRB jrb_find_gen(JRB root, Jval, int (*func)(Jval, Jval));
```