Fall 2010
CSC 4304 – Systems Programming
Homework Assignment #2

The due date is: September 30th, Thursday, before the class. Late submission is **not** allowed. Copy all of answers and your code into a text file, rename it to **hw2_yourlastname**, and email to thanks@csc.lsu.edu, cc to kosar@cct.lsu.edu

Problem 1:

Below are two implementations of the same function:

```
#include <fcntl.h>
#include "dlist.h"

Dlist file_to_dlist_1(char *fn)
{
  int n_ints;
  Dlist d;
  int i, j;
  int fd;

  fd = open(fn, O_RDONLY);

  d = make_dl();
  read(fd, &n_ints, sizeof(int));
  for (i = 0; i < n_ints; i++) {
    read(fd, &j, sizeof(int));
    dl_insert_b(d, j);
  }
  close(fd);
  return d;
}
```

```
#include <stdio.h>
#include "dlist.h"

Dlist file_to_dlist_2(char *fn)
{
  int n_ints;
  Dlist d;
  int i, j;
  FILE *f;

  f = fopen(fn, "r");

  d = make_dl();
  fread(&n_ints, sizeof(int), 1, f);
  for (i = 0; i < n_ints; i++) {
    fread(&j, sizeof(int), 1, f);
    dl_insert_b(d, j);
  }
  fclose(f);
  return d;
}
```

a) Which of the functions above will be more efficient if the file being read is large, and why? Explain in detail.

b) Write a function my_file_to_dlist() which is functionally equivalent to these two functions, and is at least as efficient as the best of these two. You may only use open(), close(), and read() and not fopen(), fclose() and fread(). Explain why this is efficient code? You may assume that the file being read is never bigger than the amount of available physical memory.

Problem 2:

Consider the following commands. Describe the state of the directory in terms of filenames, inodes and disk storage at the position of the arrow below. Explain what happens to this state during each rm command. Include when the disk storage associated with this file is freed. How does Unix know to free the disk storage when it does?

```
$   ls
$   cat > f1
This is file 1
<CTRL-D>
$ ln f1  f2
$ ln -s f1  f2
                           ←-------------
$   rm f1
$    rm f2
$   rm f3
```

Problem 3:

Implement my_popen()   and my_pclose() functions in C with following requirements:

NAME

my_popen, my_pclose -- open or close a pipe (for I/O) from and to a process

SYNOPSIS

```
int my_popen(char *command)
int my_pclose(int fd)
```

DESCRIPTION

The argument to my_popen() is a pointer to a null terminated shell command. my_popen() executes the command and returns a file descriptor. The command is hooked up to its calling process in the following way: Standard input of the command should come from standard output of the calling process. Standard output of the command should be readable by the calling process using the return value of my_popen() as a file descriptor.

my_pclose() should be called on the return value of my_popen(). This will wait for the command to complete, close all related open files, and restore the calling process's standard output fd to what it was before my_popen() was called. my_pclose() returns the status of the command executed by my_popen(). You may assume that my_popen() is never called twice without an intervening my_pclose().

<u>Problem 4:</u>

The subroutine free() frees a block of memory so that it can be reused by malloc(). However, free() is passed only a pointer to the block to be freed. Answer the following questions about free():

1. How does free() know how big this block is?
2. "Bad arguments" can be passed to free(). Describe what a bad argument to free() is and what are the possible results of passing bad arguments to free()?
3. Free() should be fixed so that bad arguments are recognized (or are almost always recognized). Discuss how this can be achieved.

<u>Problem 5:</u>

a.) What are the differences between POSIX signals and real-time signals?

b.) Write a program which blocks all of the other signals (except the ones which cannot be blocked) if a signal arrives (same one or different one) while your program is inside a signal handler. Hint use sigaction() function.