ECE 331 - HW 1 Blake Bourque

Due: 1/24/2013

@Note: This was written in markdown which compiles to html. The pound `#` is reserved as a keyword in markdown. In the below document $ is used to indicate a command prompt.

1.) Copy file from remote server with scp

$scp bbourque@hammer.eece.maine.edu:/usr/linux/public/ece331/fmemset-char.c .

2.) System Information

a.) Program:

```
////////////////////////////////////////////////////////////////////////////////
// ECE 331 - HW1
// Author: Blake Bourque
// Date: 1/24/2013
// Purpose: a small program to print some information about our RPi
////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <unistd.h>
#include <sys/utsname.h>

int main(void)
{
    long pages = sysconf(_SC_PHYS_PAGES);               //Physical Pages
    long pgsz = sysconf(_SC_PAGESIZE);                  //Physical Page size
    long proc = sysconf(_SC_NPROCESSORS_ONLN);  //Online Processors

    printf("Pages: %ld \n", pages);
    printf("Page Size: %ld \n", pgsz);
    printf("Processors: %ld \n", proc);

    struct utsname my_uname;
    if(uname(&my_uname) == -1)
        printf("uname call failed!");
    else
        printf("Machine:%s\n", my_uname.machine);//machine/hardware identifier
}
```

b.) The "Hardware identifier": The "Hardware identifier" `armv6l` is an indicator of the architecture of the processor. The RPi uses an ARM11 processor which is a RISC `(Reduced Instruction Set Computer)`. A desktop computer might report `x86_64` indicating that the computer uses the x86 64bit architecture. Note: Desktops are CISC `(Complex Instruction Set Computers)`.

3.) mset

```
////////////////////////////////////////////////////////////////////////////////
// ECE 331 - HW1
```

```
// Author: Blake Bourque
// Date: 1/24/2013
// Purpose: Implement a function `mset` which mimics the behavior of c's `memset`
// purely in c code. The below code uses two strategise to be faster than the
// sample code given:
// 1. Where possible set 64bits (8 bytes) of memory at a time
// 2. Pointers & Pointer arithmetic over indexing.
////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <stdint.h>

#define SIZE (128*1024*1024)

////////////////////////////////////////////////////////////////////////////////
// Forward Declarations
////////////////////////////////////////////////////////////////////////////////
void* mset(void *s, int c, size_t n);
void check(void *s, int c, size_t n);

////////////////////////////////////////////////////////////////////////////////
// Main
////////////////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
    char *m = (char *)malloc(SIZE);
    if (m==NULL) {
        perror("malloc");
        return errno;
    }

    mset(m, 0x8, SIZE);
    //check(m, 0x8, SIZE); //uncomment this to check the output. (adds time)

    free(m);
    return(0);
```

```
}
////////////////////////////////////////////////////////////////////////////
// Function to imitate c's memset function
////////////////////////////////////////////////////////////////////////////
void* mset(void *s, int c, size_t n)
//s is a pointer to the first byte
//c is the value of each byte
//n is number of bytes
{
    register uint64_t nVal = c;
    nVal |= (nVal << 8);
    nVal |= (nVal << 16);
    nVal |= (nVal << 32);

    register uintptr_t ptr = (uintptr_t)s;
    register uintptr_t end = (uintptr_t)(ptr + n);

    for (ptr; (ptr+8) < end; ptr+=8) {
        *((uint64_t *)(ptr)) = nVal;  //write 8 bytes
    }

    if( n % 8 != 0) {       //do the last few bytes individually
        ptr -= 8;               //Clear out the increment from the last loop check
        for (ptr; ptr<end; ptr++) {
            *((unsigned char *)ptr)=(unsigned char)c;
        }
    }

    return s;
}
////////////////////////////////////////////////////////////////////////////
// Function to ensure that the output is correct
////////////////////////////////////////////////////////////////////////////
void check(void *s, int c, size_t n)
{
    register int numERR =0;

    register uintptr_t ptr = (uintptr_t)s;
    register uintptr_t end = (uintptr_t)(ptr + n);
```

```
    for (ptr; ptr < end; ptr++) {
        if(*((unsigned char *)ptr)!=(unsigned char)c){
            numERR ++;
            printf("ERR @ Addy: %p\n", (void*)ptr);
        }
    }
    printf("Error: %d\n", numERR);
}
```

@Note: To test the fallback code change `#define SIZE (128*1024*1024)` to `#define SIZE (128*1024*1024-1)`

4.) Install top:

First one must find the package which contains top, then that package can be installed.

1. Find where top is installed:
   $ which top
2. determine what package contains top:
   $ dpkg -S /usr/bin/top
   - One can Shortcut steps 1&2 with:
     $ dpkg -S `which top`
3. Install package with apt
   $ apt-get install procps

5.) Exploring udevd

a.) udevd lives in: `/sbin/udevd`
b.) Udevd captures events from the kernel and forwards them to udev. udev is a programs that manages device nodes in the `/dev` directory.
c.) udevd & udev are a usability enhancement, without them a root user could manage the `/dev` directory. For the common user, udevd is important to have running as without it only statically defined devices can be accessed.

6.) Permissions:

a.) $ chmod g+rx,o+r opamp.m
b.) $ chmod og+r,u-w opamp.m