Preprocessor Directives/malloc()/Errors

Preprocessor Directives

Preprocessor directives are commands for the preprocessor, `cpp`. Do not confuse the C preprocessor with C++, an object oriented language. All directive being with a `#` (pound) character. One you are already familiar with is `#include`. Source code can be conditionally included or excluded using `#ifdef` - `#endif`. Macros can be created with `#define`.

```
#include <stdio.h>

// Turn on debugging.
#define __DEBUG 1

int main(int argc, char * argv[], char * env[])
{

#ifdef __DEBUG
// Compile the following code if __DEBUG is defined
      int i;
      printf("Input arguments are:\n");
      for (i=0;i<argc;i++) {
            printf("\tArgc[%d] is %s\n",i,argc[i]);
      }
#endif //__DEBUG
      return(argc);
}
```

Defining `__DEBUG` as `0` (`#define __DEBUG 0`) still defines `__DEBUG`. Use `#undef` to undefine a name. Commenting out code is simpler using conditional directives.

```
j=atan(1);
#if 0
// This code will not be compile.
for (i=0;i<test;i++) {
      printf("This code will not run\n");
}
#endif //commented out
// Code below here will compile.
pointer=(char *)malloc(x);
```

Complex conditional statements are useful.

```
#if LINUX_VERSION_CODE < KERNEL_VERSION(2.2.0)
// If the kernel version is below 2.2.0 then do the following
kfree(buffer);
#else
// Kernel version 2.2.0and  above.
kafree(buffer)
#ifdef __DEBUG
// Nested conditional directive.
printk(KERN_NOTICE "buffer freed\n");
#endif //__DEBUG
#endif //LINUX_VERSION_CODE
```

Include files should only be included once. At the beginning of an include file, test to see if a name is not defined if not define it.

```
#ifndef __MY_INCLUDE_FILE_H__
#define __MY_INCLUDE_FILE_H__
// Include stuff here
#endif __MY_INCLUDE_FILE_H__
```

malloc()

`malloc()` is used to allocate memory. Using `malloc()` decreases the size of the executable because the memory is allocated at run time rather than compile time. Use `free()` to release the memory from a program when the memory is no longer needed.

```
//Reading data from a file.  Create a 1K buffer.
char * buf;

buf=(char *)malloc(1024);
while(!feof(myfile)) {
      //read the data, write it out the socket.
      data=read(fd,buff,1024);
      write(sh,buf,data);
}
free(buf);
```

Errors

To handles error in C, errno and perror(). In a program #incude <errno.h>. Then reference the global variable errno to find out the cause of an error. perror() print and error message based on the current value of errno.

```
#include <errno.h>

int fd;

fd=open("myfile",O_RONLY);
if (fd<0) {
      // open() returns -1 if an error occurs and sets
      // errno according to the error.
      printf("Errno is: %d\n",errno);
      perror("open(): ");
      exit(errno);
}
```

Write a program that uses `malloc()` and `free()`. Your program should allocate as much memory as possible without swapping in 64K blocks. Use an array of pointers to keep track of the blocks. Use `memset()` to zero the first block. Copy the zeroed first block to all other blocks. Use `sysinfo()` to get the total usable memory. You should check for and handle errors. Create an include file for you program. Be sure to use `#if` and `#define`.