Csci4061 : Introduction to Operating System

Recitation – 3

UNIX System Programming Using C

September 25, 2017

TA : Shalini Pandey

Agenda

- Compiling and executing programs on Unix
- System programming in UNIX using C
  - #include and #define
  - File handling
  - Basic Error Handling
- Makefiles
- Debugging using gdb

A simple C program – HelloWorld.c

```
#include<stdio.h>
int main(int argc, char** argv)
{
    printf("Hello, World!\n");
    return 0;
}
```

Compiling and Executing programs

- Compiling using ‘cc’ command
  - Type :

    cc HelloWorld.c
- An executable file with name ‘a.out’ will be created
- Execute a.out to run the program

  ./a.out

CC Command

- Compiler Options
  - -o : specify the output filename
  - -c : only compile the file, but do not link
  - -g : debugging information
- Examples

  cc –o HelloWorld HelloWorld.c

  cc –c HelloWorld.c

Exercise Problem 1

- Problem Statement :

1.Compile and run the helloworld.c code.

2.Generate an object file file named helloworld.o

- Resources : helloworld.c

Agenda

- Compiling and executing programs on Unix
- System programming in UNIX using C
  - #include and #define
  - File handling
  - Basic Error Handling
- Makefiles
- Debugging using gdb

C Program Structure

- #include
  - To include other header or .c files into your program
  - #include directive tells preprocessor to scan and process the files specified in #include directive before processing the rest of the program
  - Two types of includes
    - #include<…>:this form tells the preprocessor to search for include files in the system directories - /usr/include
      - Used to include system files such as stdio.h
    - #include ”…” :this form tells the preprocessor to search for include files in the same directory

C Program Structure

- #define
  - To define macros

    #define PI 3.14

  - To define compile-time flags

    #define DEBUG

  - #ifdef and #ifndef

Example

```
#include<stdio.h>
#define CHECK
int main(int argc,char** argv)
{
   #ifdef CHECK
     printf("CHECK is defined");
   #else
     printf("CHECK is not defined");
   #endif
}
```

File Handling

- #include <stdio.h>
- Opening file using fopen

  FILE* fp = fopen(“file.txt”, “r”);

  “r”- reading; “w”- writing; “a”- append
- Reading from file –
  - fgetc to read a single character/byte from file

    int c = fgetc(fp);
  - fgets to read a string of characters from file

    char buffer[100];

    char* retcode = fgets(buffer,100,fp);
- Writing to file –

  fputc(c,fp);

  fputs(“Hello World”,fp);

Error Handling

- errno
  - type int and is defined in <errno.h>
  - errno is used by many functions to return error values
  - Do man errno – to find out error messages and codes

- perror()
  - perror() function maps the error number accessed through the symbol errno to a language-dependent error message
  - Prints the string passed followed by colon and space
  - Then prints the error message returned by strerror()

Example- File handling and error checking

- Program description:
  - File copy from source file to destination file
  - If source file does not exist, give an error
  - If destination file is a directory, give an error

- Learn how to:
  - Read and write file
  - Use errno and perror

Example- File handling and error checking

```
1   #include "check.h"
2   #include<errno.h>
3   #include<stdio.h>
4   int main(int argc, char **argv) {
6       FILE *fpr, *fpw;
7       int letter;
8       if(argc != 3)
9           printf("Usage: filecopy <filename> <copy-filename> \n");
10  /*open the first file for reading*/
11      if( ( fpr = fopen( argv[1], "r")) == NULL ) {
12  #ifdef CHECK
13          perror("Cannot open the file\n");
14  #else
15          printf("No error printing enabled!\n");
16  #endif
17          exit( 0 );
18      }
```

Example- File handling and error checking(Contd)

```
19     if ( (fpw=fopen(argv[2], "w"))==NULL ) {
20 #ifdef CHECK
21         if (errno == EISDIR)
22             printf("Entered name is for a directory\n");
23         perror("Cannot open the file\n");
24 #else
25         printf("No error printing enabled!\n");
26 #endif
27         exit(0);
28     }
29     while( ( letter = fgetc( fpr ) ) != EOF){
30         fputc(letter, fpw);
31     }
32
33     fclose(fpr);
34     fclose(fpw);
35 }
```

Exercise Problem 2

- Problem Statement :

1. Copy an existing file to another file.

2. From the given code remove the #include”check.h” and answer how output of the code changes.

3. Try reading from a non-existing file and give the textual representation of the using perror().

- Resources : filecopy.c

Pointers to Textbook Sections

- Please see Robins and Robins book for these topics
- From Chapter 2
  - Section 2.4 (Examples 2.4 and 2.5) for discussion on errno, perror, and strerror
  - Section 2.5 for #define and #ifdef
  - Section 2.6 for argument array
  - Omit Sections 2.7 and 2.8 at this stage
- Form Chapter 4
  - Section 4.2 on reading and writing files: See the Example 4.7 (copyfile.c) and Program Code 4.2 (copyfile1.c)
  - Section 4.3 on opening and closing files

Agenda

- Compiling and executing programs on Unix
- System programming in UNIX using C
  - #include and #define
  - File handling
  - Basic Error Handling
- Makefiles
- Debugging using ddd

Makefile

- What is the utility of makefiles
  - You don't have to compile each file in your project separately
  - When you change only a subset of the files in your project, you don't have to compile all the files.
  - Makefiles take care of compiling only the changed files
- Makefile contains instructions and rules to compile the source-files in your project and build executables

Makefile (Contd)

- 'make' is a utility which reads the makefiles and builds target/executables according to the rules specified in makefile
- Type – make –f <name of makefile>
  - If you don't use option -f, by default make reads the file with name 'makefile'

Makefile (Contd)

- Rules
  - A rule tells when and how to make a target e.g to compile a source file or build an executable
  - Rules syntax –

```
target : prerequisites
    command
helloworld : helloworld.o
    gcc –o helloworld helloworld.o
helloworld.o : helloworld.c
    gcc –w –c helloworld.c
```

  - Here helloworld.c will be compiled only when modification time of helloworld.c is more recent than helloworld.o

Makefile Example

```
#the C compiler
CC = gcc
CFLAGS = -g

#this rule is to link the object code into an executable
helloworld: helloworld.o
    $(CC) -o HelloWorld HelloWorld.o

#this rule is to compile the source code
helloworld.o: helloworld.c
    $(CC) $(CFLAGS) -c HelloWorld.c

clean:
    rm -f *.o
    rm -f HelloWorld
```

Agenda

- Compiling and executing programs on Unix
- System programming in UNIX using C
  - #include and #define
  - File handling
  - Basic Error Handling
- Makefiles
- Debugging using gdb

Debugging using GDB

- GDB (GNU Debugger) is the standard command-line debugger for the GNU operating system
- Type – gdb <executable-filename> to invoke
- Compile source files with –g option to enable debugging
  - -g option tells compiler to put debug information in the object file
  - cc –g –o HelloWord HelloWorld.c
  - gdb ./HelloWorld
- GDB allows –
  - To execute and stop your program at specified points
  - Examine what has happened and inspect your program after stop-point
  - Make changes to variables and run

Example-Debugging using GDB

```
#include<stdio.h>
int main(int argc,char** argv)
{
        int arr[5];
        for(int i=0; i<=6; i++)
                arr[i] = i;
}
```

Exercise Problem 3

- Problem Statement : Debug the above code using gdb and try the following commands.

1. bt.
2. where.
3. frame.
4. info locals.

- Resources: gdb.c

http://darkdust.net/files/GDB%20Cheat%20Sh

Watch the Execution of Program

Q: How do I stop execution?

A: You can stop execution using the **Ctrl-C** key combination.

Q: How do I continue execution?

A: Use the **continue** command to restart execution of your program whenever it is stopped.

Q: How do I see where my program stopped?

A: Use the **list** command to have gdb print out the lines of code above and below the line the program is stopped at.

Q: How do I step through my code line-by-line?

A: First stop your program by sending it signals or using breakpoints. Then use the **next** and **step** commands.

Q: How do I examine variables?

A: Use the **print** command with a variable name as the argument.

Q: How do I modify variables?

A: Use the **set** command with a C assignment statement as the argument.

Use the Call Stack

Q: What is call statck?

A: The call stack is where we find the stack frames that control program flow. When a function is called, it creates a stack frame that tells the computer how to return control to its caller after it has finished executing. Stack frames are also where local variables and function arguments are 'stored'. We can look at these stack frames to determine how our program is running.

Q: How do I get a backtrace?

A: Use the gdb command **backtrace**.

Q: How do I change stack frames?

A: Use the gdb command **frame**. Pass the number of the frame you want as an argument to the command.

Q: How do I examine stack frames?

A: To look at the contents of the current frame, there are 3 useful gdb commands.

info frame displays information about the current stack frame.

info locals displays the list of local variables and their values for the current stack frame.
info args displays the list of arguments.

Use Breakpoints

Q: How do I set a breakpoint on a line?

A: The command to set a breakpoint is **break**. If you have more than one file, you must give the **break** command a filename as well:

(gdb) break test.c:19

Q: How do I set a breakpoint on a C function?

A: To set a breakpoint on a C function, pass it's name to break.

(gdb) break func1

Q: How do I set a temporary breakpoint?

A: Use the **tbreak** command instead of break. A temporary breakpoint only stops the program once, and is then removed.

Q: How do I get a list of breakpoints?

A: Use the **info breakpoints** command.

Q: How do I disable breakpoints?

A: Use the **disable** command. Pass the number of the breakpoint you wish to disable as an argument to this command.