Chufeng Jiang (19949)

HW-1A: There are 9 questions in this Homework.

Edit programs 0 through 6 in your Unix/Linux OS and show the outputs that you see after running programs 1 through 6

Table of Content

Color and symbol specification in Terminal output of this document:

1. `gcc  hw1_h.c` No shading color: command line;
2. `Hello World` Yellow shading color: program output;
3. `d` Blue highlight with yellow shading: (waiting) user input;
4. $ --- command prompt for NON-root account;
5. # --- command prompt for Root account.

The program source code and executable files can be found on Github:

https://github.com/Chufeng-Jiang/SFBU-CS510-Advanced-Linux-Programming/tree/main/Homework/Week01

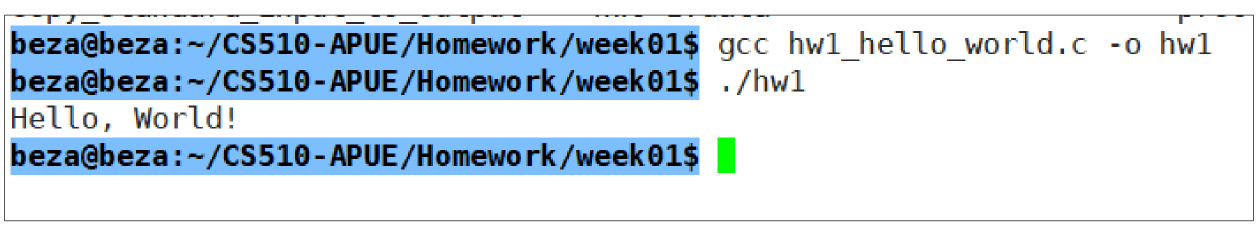
Q1. Hello_world program: hello_world.c

hello_world.c
This program is to print “Hello, World!” to the terminal.

```
#include <stdio.h> //header file
int main(void) {
// a print statement to print "Hello, World"
	printf("Hello, World! \n");
	return 0;
}
```

[Terminal Outputs]

Terminal	
$	gcc hw1_hello_world.c -o hw1
$	./hw1
	Hello, World!



```
beza@beza:~/CS510-APUE/Homework/week01$ gcc hw1_hello_world.c -o hw1
beza@beza:~/CS510-APUE/Homework/week01$ ./hw1
Hello, World!
beza@beza:~/CS510-APUE/Homework/week01$
```

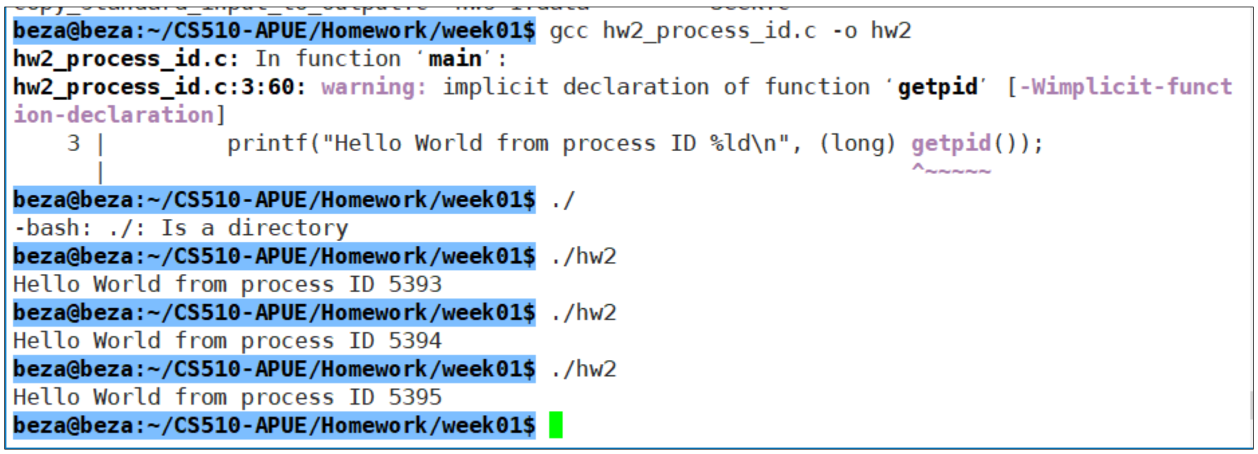
Q2. Process ID program: process_id.c

hw2_process_id.c
This program is to print out the pid of the current process.

```c
#include <stdio.h>
int main(void) {
// here the program used the getpid() function to get the process id
// a print statement is used to output the string and pid.
        printf("Hello World from process ID %ld\n", (long) getpid());
        return(0);
}
```

Terminal

```
$   gcc hw2_process_id.c -o hw2
    hw2_process_id.c: In function ‘main’:
    hw2_process_id.c:3:60: warning: implicit declaration of function ‘getpid’ [-Wimplicit-
    function-declaration]
        3 |         printf("Hello World from process ID %ld\n", (long) getpid());
          |                                                            ^~~~~~

$   ./hw2
    Hello World from process ID 5393

$   ./hw2
    Hello World from process ID 5394

$   ./hw2
    Hello World from process ID 5395
```



```
beza@beza:~/CS510-APUE/Homework/week01$ gcc hw2_process_id.c -o hw2
hw2_process_id.c: In function ‘main’:
hw2_process_id.c:3:60: warning: implicit declaration of function ‘getpid’ [-Wimplicit-funct
ion-declaration]
    3 |         printf("Hello World from process ID %ld\n", (long) getpid());
      |                                                            ^~~~~~
beza@beza:~/CS510-APUE/Homework/week01$ ./
-bash: ./: Is a directory
beza@beza:~/CS510-APUE/Homework/week01$ ./hw2
Hello World from process ID 5393
beza@beza:~/CS510-APUE/Homework/week01$ ./hw2
Hello World from process ID 5394
beza@beza:~/CS510-APUE/Homework/week01$ ./hw2
Hello World from process ID 5395
beza@beza:~/CS510-APUE/Homework/week01$
```

Q3. User ID, Group ID program: user_id_group_id.c

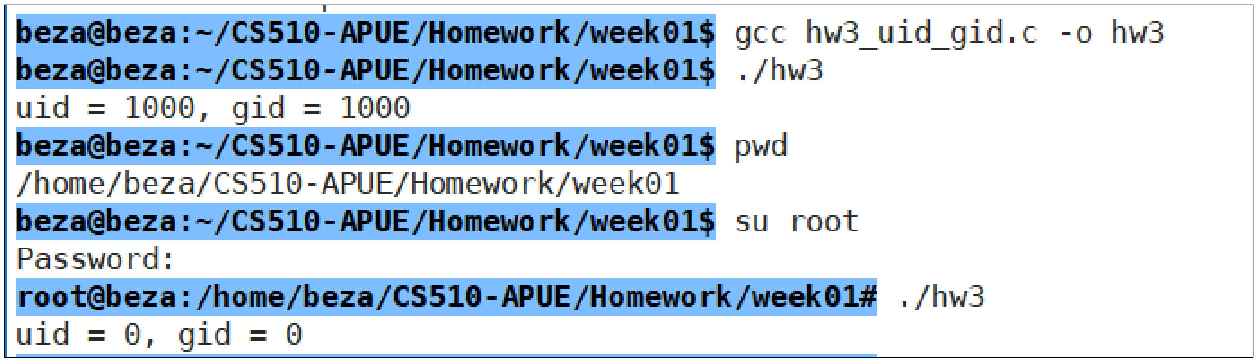
hw3_uid_gid.c
This program is to print the process user's user ID and group ID.

```
#include "apue.h"

Int main(void)
{
// here the program used the getuid() function to get the userid, and the getgid() function to get the group id that user belong to
// a print statement is used to output the string and pid,
        printf("uid = %d, gid = %d\n", getuid(), getgid());
        return(0);
}
```

Terminal

```
$    gcc hw3_uid_gid.c -o hw3
     ./hw3
     uid = 1000, gid = 1000

$    su root
     Password:

#     ./hw3
     uid = 0, gid = 0
```

We can find that when using the personal account and the root account, different uid and gid are applied. More specifically, the uid for root user is 0, and the gid for the root user is 0.



```
beza@beza:~/CS510-APUE/Homework/week01$ gcc hw3_uid_gid.c -o hw3
beza@beza:~/CS510-APUE/Homework/week01$ ./hw3
uid = 1000, gid = 1000
beza@beza:~/CS510-APUE/Homework/week01$ pwd
/home/beza/CS510-APUE/Homework/week01
beza@beza:~/CS510-APUE/Homework/week01$ su root
Password:
root@beza:/home/beza/CS510-APUE/Homework/week01# ./hw3
uid = 0, gid = 0
```

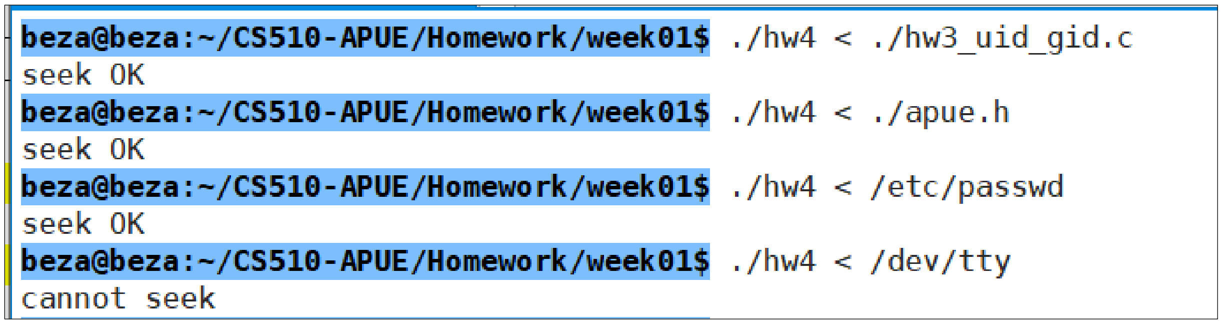
Q4. Seek.c program:

hw4_seek.c
This program is to determine whether a file is seekable or not, some files have specific accecc restriction that making the file is not open to the users thus is not seekable.

```
#include "apue.h"  // User Defined Header File
#include <stdio.h> //Standard Header File

int main(void){
          // If a file is not direct to the STDIN, the program print "cannot seek".
   if (lseek(STDIN_FILENO, 0, SEEK_CUR) == -1)
      printf("cannot seek\n");
          // else "seek OK".
   else
      printf("seek OK\n");
     // If a file is not exist, then print "No such file or directory"
          exit(0);
 }
```

Terminal

```
$ | gcc hw4_seek.c -o hw4
$ | ./hw4 < ./hw3_uid_gid.c
  | seek OK
$ | ./hw4 < ./apue.h
  | seek OK
$ | ./hw4 < /etc/passwd
  | seek OK
$ | ./hw4 < /dev/tty0
  | cannot seek
```



```
beza@beza:~/CS510-APUE/Homework/week01$ ./hw4 < ./hw3_uid_gid.c
seek OK
beza@beza:~/CS510-APUE/Homework/week01$ ./hw4 < ./apue.h
seek OK
beza@beza:~/CS510-APUE/Homework/week01$ ./hw4 < /etc/passwd
seek OK
beza@beza:~/CS510-APUE/Homework/week01$ ./hw4 < /dev/tty
cannot seek
```

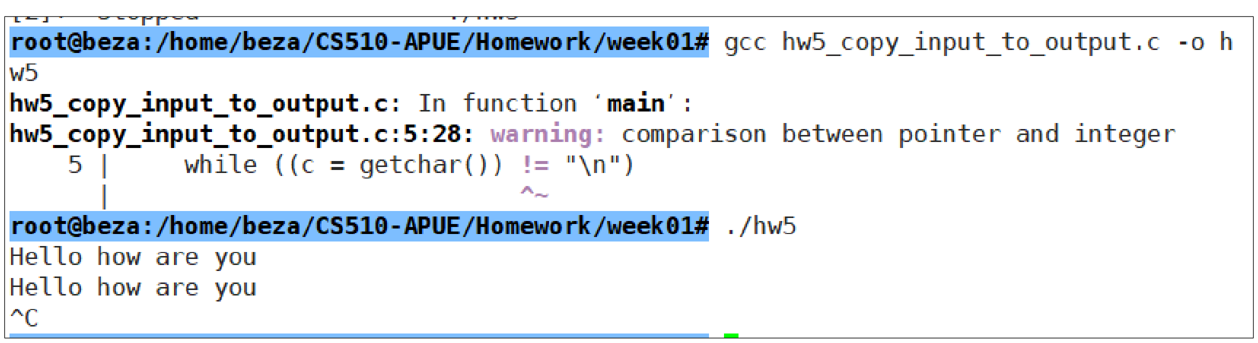
Q5. Copy input to output program: copy_input_to_output.c

hw5_copy_input_to_output.c
This program is to read data from the keyboard using getchar() function, and then print the data to the terminal using putchar() function.

```
#include "apue.h"
int main(void)
{
        int c;
// using getchar() to read character from the keyboard  until a change line symbol is input; then the putchar(c); function will display what we type in the keyboard to the screen.
        while ((c = getchar()) != "\n")
                putchar(c);
        return(0);
}
```

Terminal

```
$   gcc hw5_copy_input_to_output.c -o hw5
    hw5_copy_input_to_output.c: In function ‘main’:
    hw5_copy_input_to_output.c:5:28: warning: comparison between pointer and integer
       5 |     while ((c = getchar()) != "\n")
         |                            ^~

$   ./hw5
    Hello how are you  ← user input from keyboard
    Hello how are you
    ^C  ← signal to stop this process
```



```
root@beza:/home/beza/CS510-APUE/Homework/week01# gcc hw5_copy_input_to_output.c -o h
w5
hw5_copy_input_to_output.c: In function ‘main’:
hw5_copy_input_to_output.c:5:28: warning: comparison between pointer and integer
    5 |     while ((c = getchar()) != "\n")
      |                            ^~
root@beza:/home/beza/CS510-APUE/Homework/week01# ./hw5
Hello how are you
Hello how are you
^C
```

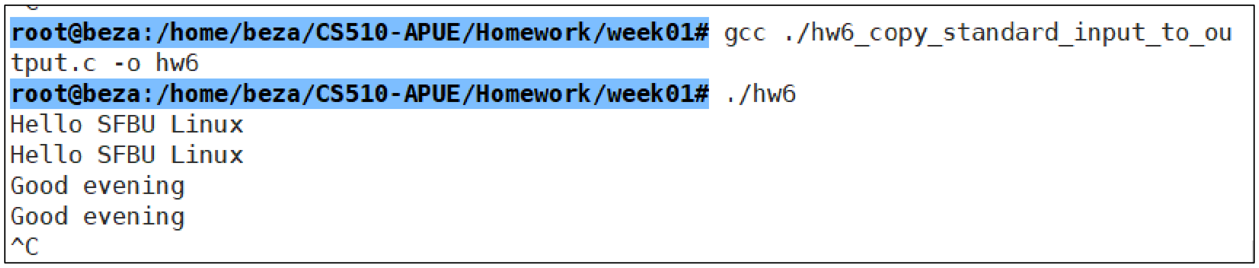
Q6. Copy standard input to output program:

This program copies data from STDIN to buf and writes to STDOUT from the buf.

```
#include "apue.h"
#define BUFFSIZE 4096

int main(void)
{
        int     n;
        char    buf[BUFFSIZE];

//loop to read from standard input (STDIN) and write to standard output (STDOUT).
// read BUFFSIZE bytes data from the STDIN to the buf
        while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
// if read more than 0 bytes, then  write to the STDOUT
// write n bytes of data already read in the buf, and send them to the STDOUT to print
            if (write(STDOUT_FILENO, buf, n) != n)
                printf("write error");

         if (n < 0)
             printf("read error");

        return (0); }
```

Terminal

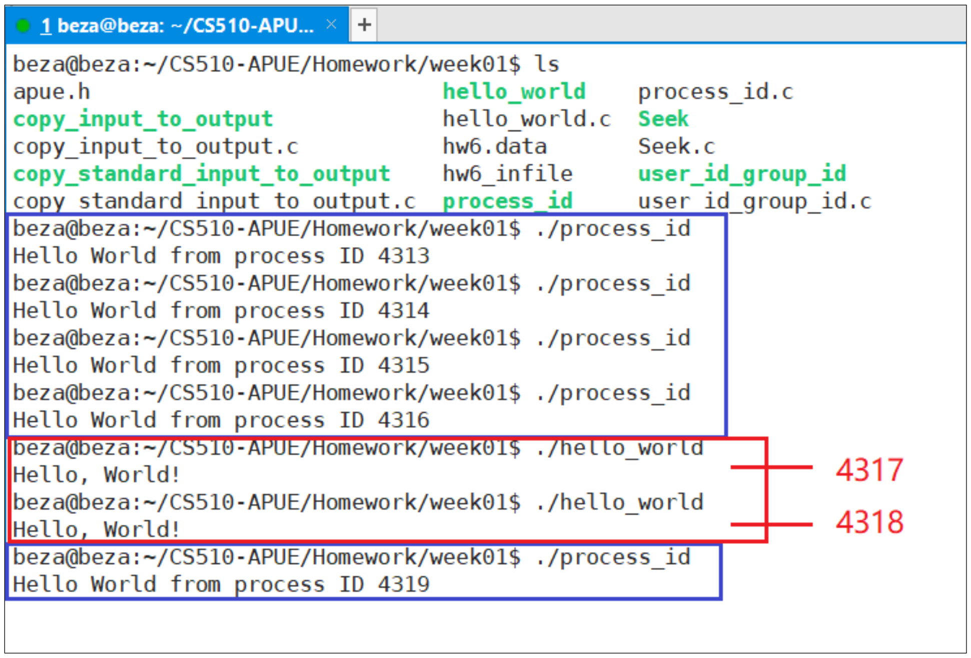
```
$   gcc ./hw6_copy_standard_input_to_output.c -o hw6
$   ./hw6
    Hello SFBU Linux 🡐 user input from keyboard
    Hello SFBU Linux
    Good evening 🡐 user input from keyboard
    Good evening
    ^C  🡐 signal to stop this process
```



```
root@beza:/home/beza/CS510-APUE/Homework/week01# gcc ./hw6_copy_standard_input_to_ou
tput.c -o hw6
root@beza:/home/beza/CS510-APUE/Homework/week01# ./hw6
Hello SFBU Linux
Hello SFBU Linux
Good evening
Good evening
^C
```

Q7. Exercise 1.2, Page 24

I have ran the ***process_id*** program for 4 consecutive times (pid:4313-4316) in Linux, and I ran ***hello_world*** program for 2 times. When I re-ran the ***process_id*** program, I got the pid of 4319, leaving 2 missing numbers from my previous running the same program. Therefore, I think the 4317 and 4318 were assigned to the ***hello_world*** programs, of which the process have ended already. So, we don't know what are the processes with IDs 852 and 853. The pid numbers may assigned to other programs, and they may be running or may be already ended.



```
beza@beza:~/CS510-APUE/Homework/week01$ ls
apue.h                              hello_world    process_id.c
copy_input_to_output                hello_world.c  Seek
copy_input_to_output.c              hw6.data       Seek.c
copy_standard_input_to_output       hw6_infile     user_id_group_id
copy_standard_input_to_output.c     process_id     user_id_group_id.c
beza@beza:~/CS510-APUE/Homework/week01$ ./process_id
Hello World from process ID 4313
beza@beza:~/CS510-APUE/Homework/week01$ ./process_id
Hello World from process ID 4314
beza@beza:~/CS510-APUE/Homework/week01$ ./process_id
Hello World from process ID 4315
beza@beza:~/CS510-APUE/Homework/week01$ ./process_id
Hello World from process ID 4316
beza@beza:~/CS510-APUE/Homework/week01$ ./hello_world
Hello, World!
beza@beza:~/CS510-APUE/Homework/week01$ ./hello_world
Hello, World!
beza@beza:~/CS510-APUE/Homework/week01$ ./process_id
Hello World from process ID 4319
```

Q8. Exercise 1.3, Page 24

void perror(const char *s);

The perror function takes a pointer to the first character in a string of characters. That pointer can point to a string literal. In this context, it's important that perror use the value pointed to by the given argument but not modify the value. Without the const attribute, the function could (try to) modify the value pointed to by the given pointer. The const enables the compiler to generate a compile-time error if the function did try to modify the value pointed to by the given pointer.

char * strerror(int errnum);

The strerror function doesn't take a pointer, it takes an int, which is passed by value so the function receives a copy of the value. There is no way for strerror to modify the client's copy of the value.

Q9. Exercise 1.4, Page 24

A signed 32-bit integer can store values [-2147483648, 2147483647]. By explicitly saying "signed integer", I assume that the intention is to use only the positive values [0, 2147483647].

Time starts at Januay 1, 1970 at 00:00:00 UTC; that will be time t=0. The clock advances 1 tick/second.

2147483647 ticks	* (1 second= 1 tick)
= 2147483647 seconds	* (1 minutes= 60 seconds)
= 35791394.1166 minutes	* (1 hour= 60 minutes)
= 596523.235 hours	* (1 day =24 hours)
= 24855.135 days	* (1 year= 365.25 days)
= 68 years	

So, the value would overflow in (1970 + 68) = 2038

To extend the overflow point, we can consider, an unsigned 32-bit value, that can store values [0, 4294967296].

4294967296 seconds	* (1 minute=60 seconds)
=71582788.26666 minutes	* (1 hour=60 minutes)
=1193046.47 hours	* (1 day=24 hours)
=49710.2696 days	* (1 year=365.25 days)
=136 years	

So, the value would overflow in 1970 + 136 = 2106

This helps, but just kicks the can down the road. To further extend the overlow point, we can consider a signed 64-bit value, that can store values [-9223372036854775808, 9223372036854775807].

9223372036854775807 seconds	* (1 minute=60 seconds)
=153722867280912928 minutes	* (1 hours=60 minutes)
=2562047788015215.5 hours	* (1 day=24 hours)
=106751991167300.64 days	* (1 year=365.25 days)
=292271023045.31 years	

So, the value would overflow in 1970 + 292271023045 = 292,271,025,015

Neither of these approaches are compatible with existing applications (at least not without recompiling). Using unsigned 32-bit values instead of signed 32-bit values can cause problems with existing applications that do arithmetic on these numbers and expect signed results. Extending it to 64-bit requires additional storage allocation by the applications. (If the applications are using the proper type abstractions, then they may just need to be recompiled).