AUP : Assignment - 4 [Process Control]

Aditi Rajendra Medhane 111803177

20th September 2021

Q2

Write a program to print all existing environment variables with their values. Later input a new variable and its value and add to the environment list. Also change the value of PATH to "/usr/bin". Once again call to print the environment list.

Then in the command line, print the environment list. What is your observation?

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <pwd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <dirent.h>

void print_env(char **envp){
    while(*envp){
        printf("%s\n", *envp);
        envp++;
    }
}

int main(){
    extern char **environ;

    print_env(environ);
    printf("\n");

    if(putenv("PATH=/usr/bin") == -1){
        perror("putenv");
        return errno;
    }

    printf("Changed PATH\n");

    print_env(environ);
    return 0;
}

```

Explanation

- At the start of execution, **PATH** environment variable has some value
- After a successful call to *putenv* changes the value of **PATH** to */usr/bin*
- After the program exits, the value of the path when *env* command is run is still the initial value.
- This is because any program inherits the environment variables of it's parent. But when a child changes the value of it's environment, the value of *environ* for it's parent does not change.

Output

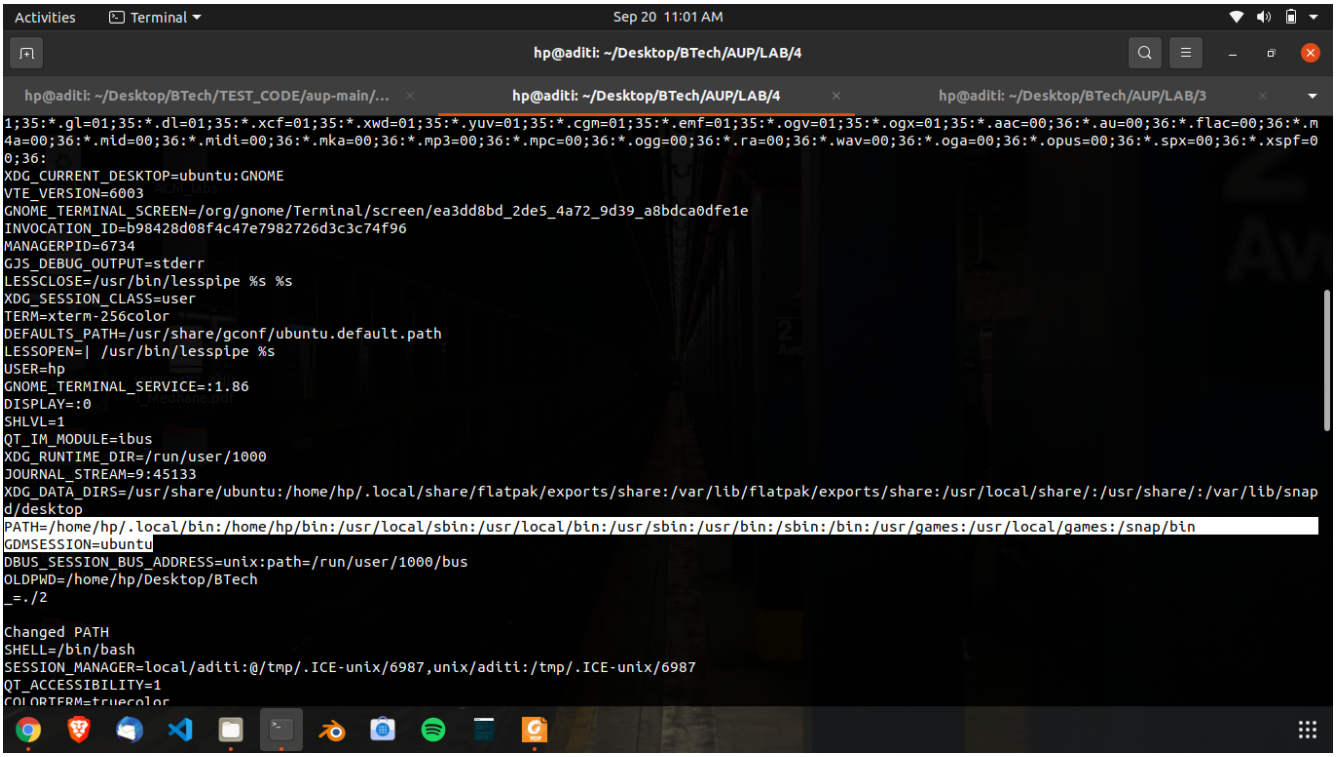

Figure 1: Value of PATH variable printed in program before it is changed

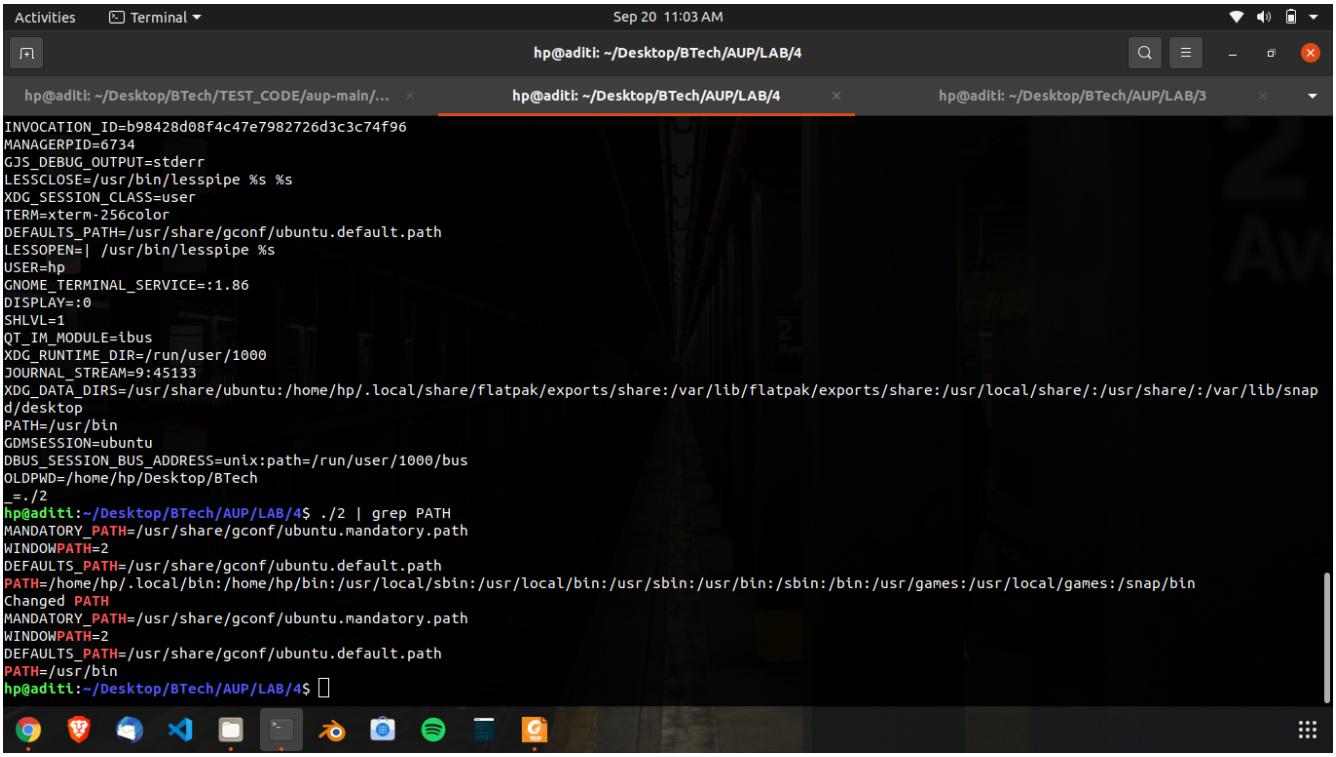

Figure 2: Value of PATH variable after it is changed using putenv + GREP "PATH"

Q3

Write a program to include different types of variables to demonstrate the behavior of setjmp/lonjmp.

- Include a public variable
- Include a jmp_buf automatic variable, a static variable and automatic in main()
- Invoke a function a() with the argument jmp_buf variable.
- If return values of a() is nonzero, exit.
- Update values of public, static and automatic variables
- Then invoke b() with the argument jmp_buf variable.
- Print values of public, static and automatic variables
- In a(),
  - Include a static variable and automatic variable
  - setjmp() invocation with argument as received jmp_buf variable.
  - Update values of static variable and automatic variable
  - Return value of return value of setjmp
- In b(), just invoke longjmp() with received jmp_buf variable and a non zero value.

Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pwd.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  #include <unistd.h>
7  #include <fcntl.h>
8  #include <errno.h>
9  #include <dirent.h>
10
11 #include <setjmp.h>
12
13 int public_var;
14 jmp_buf buf;
15 static int static_var;
16
17 int a(jmp_buf buf) {
18         static int a_static_var;
```

```
19          auto int a_auto_var;
20          int ret;
21
22          a_static_var = 4;
23          a_auto_var = 5;
24
25          printf("a_static_var = %d\na_auto_var = %d\npublic_var = %d\nstatic_var = %d\n\n",
26                  a_static_var, a_auto_var, public_var, static_var);
27
28          ret = setjmp(buf);
29
30          printf("setjump(buf) = %d\n", ret);
31          printf("a_static_var = %d\na_auto_var = %d\npublic_var = %d\nstatic_var = %d\n\n",
32                  a_static_var, a_auto_var, public_var, static_var);
33
34          a_static_var = 104;
35          a_auto_var = 105;
36
37          return ret;
38  }
39
40  int b(jmp_buf buf) {
41          longjmp(buf, 42);
42  }
43
44
45  int main(void) {
46          auto int auto_var;
47
48          public_var = 1;
49          static_var = 2;
50          auto_var = 3;
51
52          printf("Before a(buf)\n");
53          printf("public_var = %d\nstatic_var = %d\nauto_var = %d\n\n",
54                  public_var, static_var, auto_var);
55
56          if (a(buf)) {
57                  exit(0);
58          }
59
60          printf("After a(buf), before b(buf)\n");
61          printf("public_var = %d\nstatic_var = %d\nauto_var = %d\n\n",
62                  public_var, static_var, auto_var);
63
64          public_var = 101;
65          static_var = 102;
66          auto_var = 103;
67
68          b(buf);
69
70          printf("After b(buf)\n");
71          printf("public_var = %d\nstatic_var = %d\nauto_var = %d\n\n",
72                  public_var, static_var, auto_var);
73
74          return 0;
75  }
```

Explanation

In case of the program above, the values of the static and global variables will *not* be rolled back to initial state.

As for the *automatic* variables..

To quote the manual page for *longjmp*-

the values of automatic variables are unspecified after a call to longjmp() if they meet all the following criteria:

- they are local to the function that made the corresponding setjmp() call;
- their values are changed between the calls to setjmp() and longjmp(); and
- they are not declared as volatile.

It is observed that for an umoptimized code, the value of the automatic variable *a_auto_var* is *not* rolled back. This is because it is not stored in a register.

For the optimized code, the automatic variable value gets rolled back- this is because *a_auto_var is* stored in the register.

Note that the output itself is different- this is because *clang* and *gcc* might eliminate sections of code which they deem useless.

Output

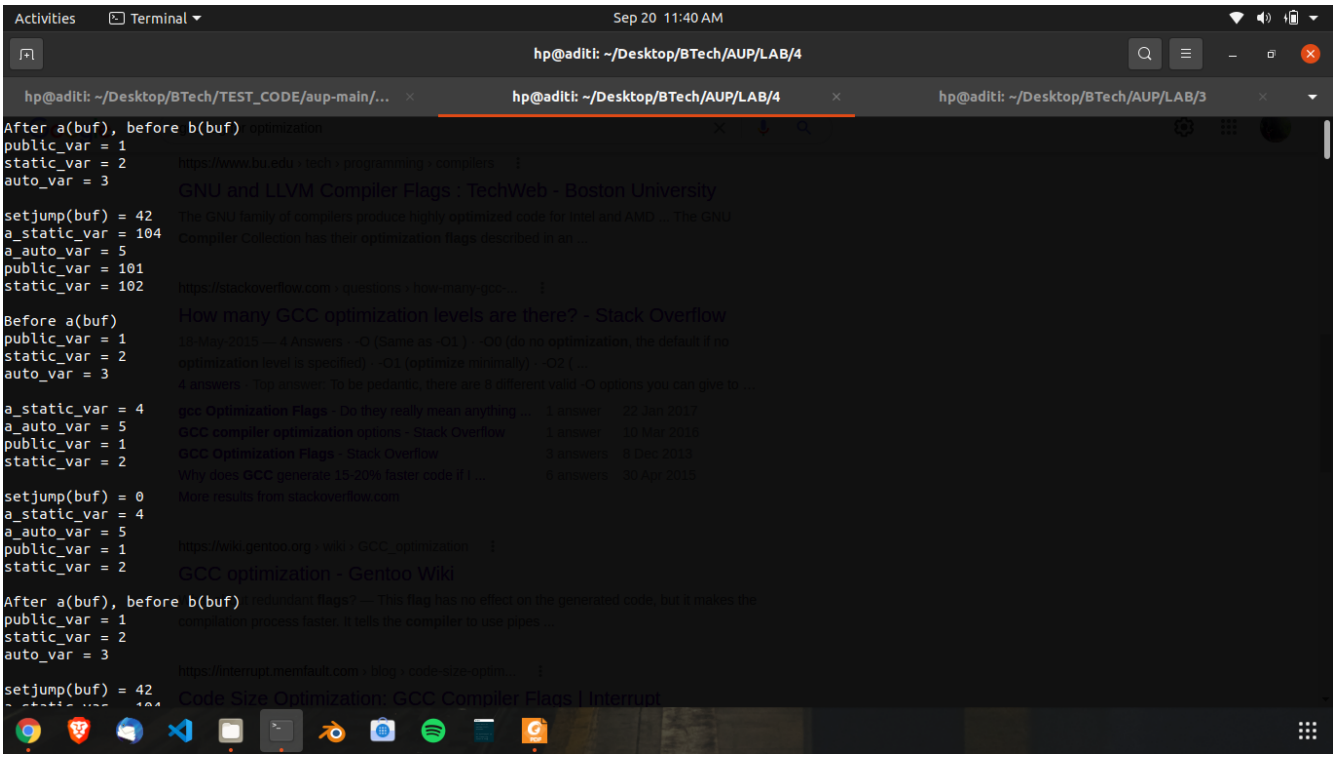

Figure 3: Output for unoptimized code

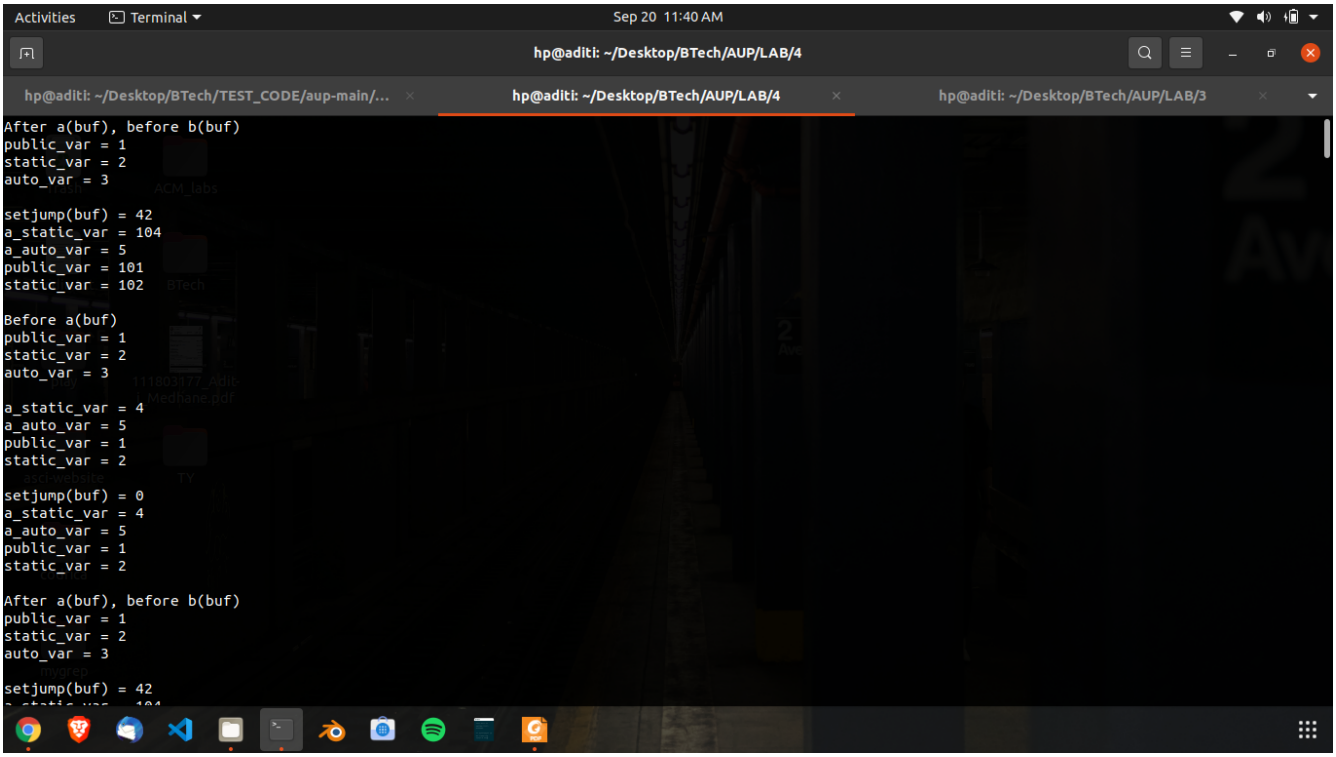

Figure 4: Output for optimized code

Q4

Creates a total of three different processes. each processes compute the factorial of integers between 1 and 10 by recursion and print the results to the screen and then terminate. Make sure to print an identifying string for the output of each process as in:

PROCESS1:fact(1)=1

PROCESS2:fact(2)=1

PROCESS2:fact(2)=2

PROCESS1:fact(2)=2

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <pwd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <dirent.h>


#define FACT_NUM 10
#define NO_OF_CHILDREN 2

long int fact(int n){
    long int f;
    if( n == 1){
        f = 1;
    }
    else{
        f = n * fact(n - 1);
    }
    return f;
```

```
24
25  }
26
27  void work_done_by_child(int child_num){
28      int i;
29      //PRINT FACTORIAL VALUES FROM n = 1 to FACT_NUM
30      for(i = 1; i <= FACT_NUM; i++){
31          printf("Process%d:fact(%d)=%ld\n", child_num, i, fact(i));
32      }
33
34  }
35
36  int main(){
37      int child, i;
38      //STEP 1: CREATE CHILDREN
39      //STEP 2: DISPATCH EACH CHILD TP CALCULATE & PRINT FACT (DONE USING work_done_by_child)
40      for(i = 0; i < NO_OF_CHILDREN; i++){
41          if((child = fork()) == -1){
42              //FORK FAILED
43              perror("Creation of child process failed");
44          }
45          else if(child == 0){
46              work_done_by_child(i + 1);
47              return 0;
48          }
49      }
50
51      return 0;
52  }
53
54
```

Output

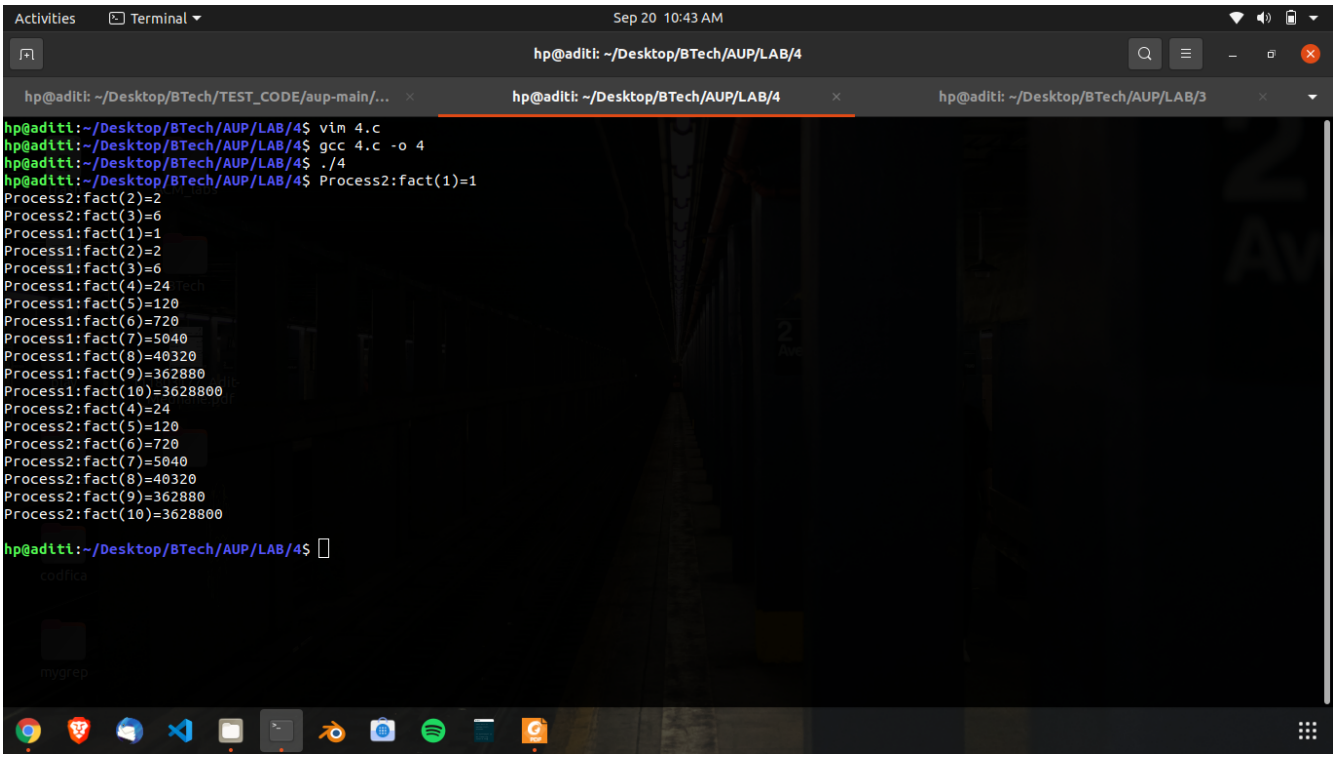

Figure 5: PROCESS 1 & 2 fact calc & print