Operating Systems Lab

Lab-2 Report

Pavan Kumar V Patil
200030041

January 15, 2023

1 Part 1

We have to print the string “Hello World” on screen. Each character must be printed by a different process. The process that prints the i th letter must have been spawned by the process that printed the (i-1) th letter.

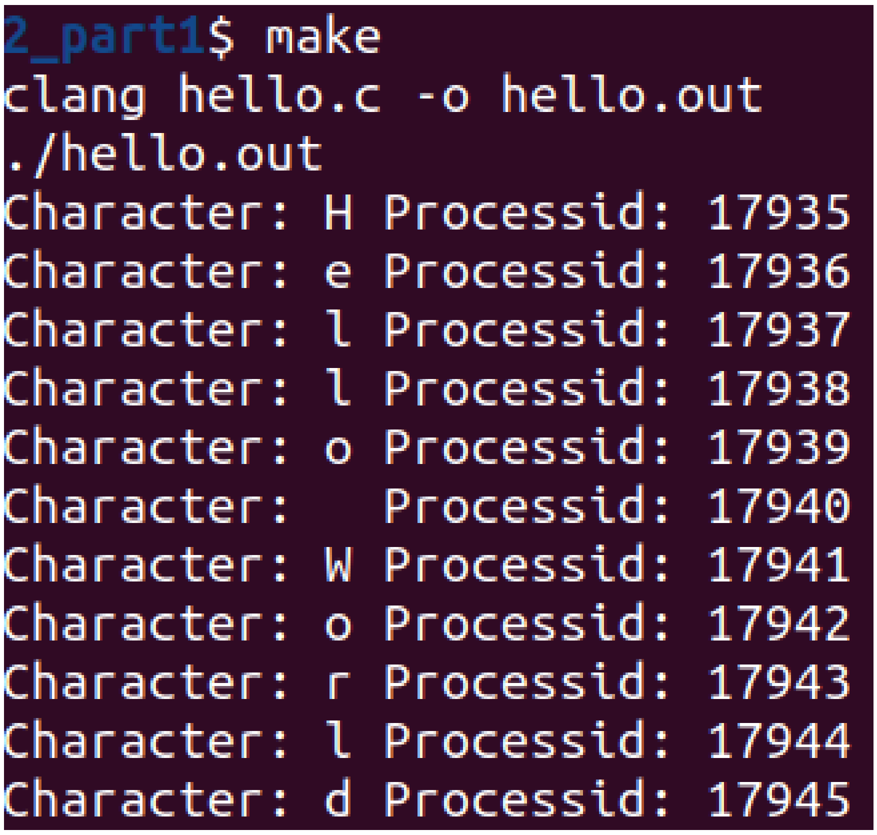
To do the above task we have code in following way:

- We have to save the string ”Hello World” in some variable, let say ’str’.
- After that we have run loop till we get character ” in str.
- Inside the loop we have to do system call ’fork’ which creates child process in that we have to print process id and character str[i] where i is the iterator of loop.
- sleep for random 3-4 seconds and for parent process break the loop or exit the program.

Code for above task:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
  char s[] = "Hello World";
  int pid, i;
  for (i = 0; s[i]!='\0'; i++)
  {
    pid = fork();
    if (pid == 0)
    {
      int process_id = getpid();
      printf("Character: %c Processid: %d\n", s[i], process_id);
    }
    else
    {
      sleep(rand() % 4 + 1);
      break;
    }
  }
  return 0;
}
```

We can do the above task in 25 number of lines of code.



```
2_part1$ make
clang hello.c -o hello.out
./hello.out
Character: H Processid: 17935
Character: e Processid: 17936
Character: l Processid: 17937
Character: l Processid: 17938
Character: o Processid: 17939
Character:   Processid: 17940
Character: W Processid: 17941
Character: o Processid: 17942
Character: r Processid: 17943
Character: l Processid: 17944
Character: d Processid: 17945
```

Figure 1: Output of part1

2 Part 2

We have write a collection of programs twice, half, square such that they execute sequentially with the same process-id, and each program should also print its PID. (process id) The user should be able to invoke any combination of these programs, to achieve the required functionality.

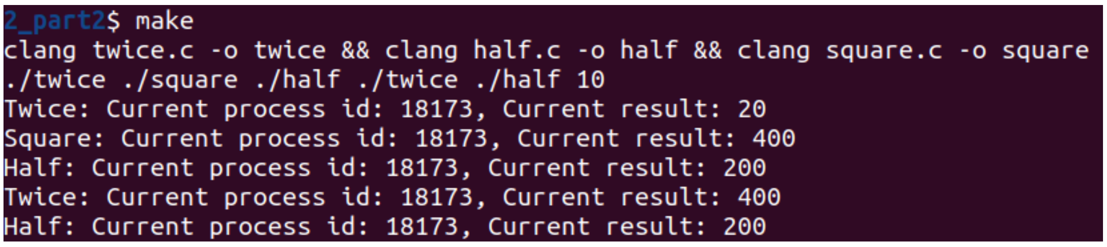
To do the above task we have code in following way:

- Firstly we have to make twice.c half.c and square.c source files.
- For each source file we have to pass arguments in sequence, means in which way task has to be done.
- For the current source file running, we have extract number from arguments say n and perform according to operation.
- Save the updated n value to argument and call execvp function and pass the parameters accordingly to execvp function.
- Benefit of execvp function is, it runs in current processes without creating another process.

Code for above task one source file (twice.c):

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int number = atoi(argv[argc - 1]);
    number = number * 2;
    printf("Twice: Current process id: %d, Current result: %d\n",
        getpid(), number);

    if (argc <= 2)
        return 0;

    char num[11];
    sprintf(num, "%d", number);
    strcpy(argv[argc - 1], num);

    argv = &argv[1];
    execvp(argv[0], argv);
    return 0;
}
```

Similarly for half.c and square.c.



```
2_part2$ make
clang twice.c -o twice && clang half.c -o half && clang square.c -o square
./twice ./square ./half ./twice ./half 10
Twice: Current process id: 18173, Current result: 20
Square: Current process id: 18173, Current result: 400
Half: Current process id: 18173, Current result: 200
Twice: Current process id: 18173, Current result: 400
Half: Current process id: 18173, Current result: 200
```

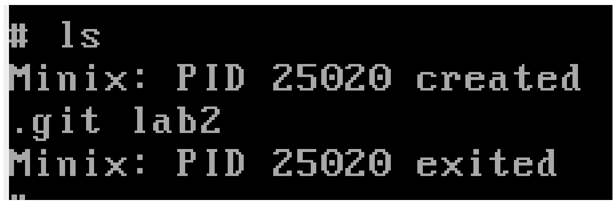
Figure 2: Output of part2

3 Part 3

We have modify Minix3 source code such that, when new process is created we have to print "Minix: PID <pid> created" and when process ends we have to print "Minix: PID <pid> exited".

To get the above ouput i have added **printf("Minix: PID %d created", new_pid);** when new pid is created in do_fork() function present in minix/servers/pm/forkexit.c file and added **printf("Minix: PID %d exited", mp− >mp_pid);** when exit_proc function called in do_exit function present in present in minix/servers/pm/forkexit.c file. After that we have to run make build command and reboot the system.

The child process creation depends on the creation of its parent and the parent process can only exit once all its children have exited, thus the tree structure of the processes is that the parent is created first and then its children are created, whereas while exiting the child exits first and then the parent process exits.



```
# ls
Minix: PID 25020 created
.git lab2
Minix: PID 25020 exited
```

Figure 3: Output of ls command process