Assignment-5
Pipes and shared memory

Subash Mylraj
(CED18I051)

25 October 2020

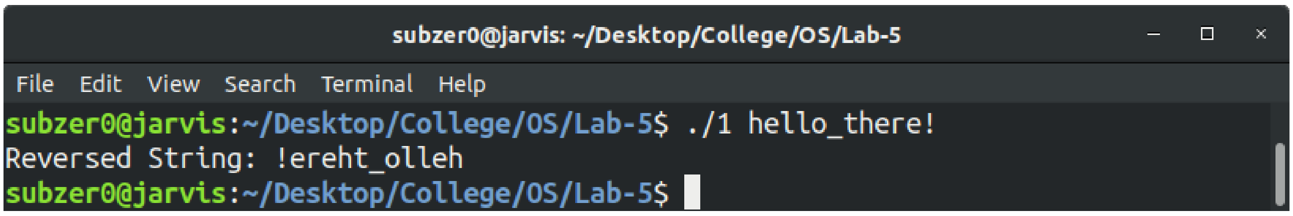
Question 1: Parent sets up a string which is read by child, reversed there and read back the parent

Code:

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

char* strrev (char*);

int main(int argc, char const *argv[])
{
   int fd1[2], fd2[2];
   pid_t pid;

   if(pipe(fd1) == -1){
      printf("Unable to create pipe\n");
   }
   if(pipe(fd2) == -1){
      printf("Unable to create pipe\n");
   }

   pid = fork();

   if(pid < 0){
      printf("fork failed\n");
   }
   else{
      if(pid == 0){  // Child block
         close(fd1[1]);

         char str[100];
         read(fd1[0], str, 100);

         close(fd1[0]);

         close(fd2[0]);

         write(fd2[1], strrev(str), strlen(str));

         close(fd2[1]);

      }
      else{        // Parent block
```

```
        close(fd1[0]);

        write(fd1[1], argv[1], strlen(argv[1]));
        close(fd1[1]);

        wait(NULL);

        close(fd2[1]);

        char rev_str[100];
        read(fd2[0], rev_str, 100);
        printf("Reversed String: %s\n", rev_str);

        close(fd2[0]);
     }
  }
  return 0;
}

char *strrev(char *str)
{
     char *itr1, *itr2;

     for (itr1=str, itr2=str+strlen(str)-1; itr2>itr1; ++itr1, --itr2)
     {
          *itr1 ^= *itr2;
          *itr2 ^= *itr1;
          *itr1 ^= *itr2;
     }
     return str;
}
```

Output:



subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./1 hello_there!
Reversed String: !ereht_olleh
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

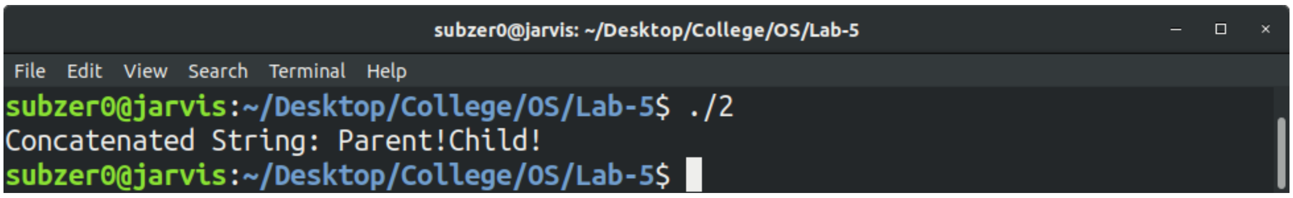
Question 2: Parent sets up string 1 and child sets up string 2. String 2 concatenated to string 1 at parent end and then read back at the child end.

Code:

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

int main(int argc, char const *argv[])
{
  int fd1[2], fd2[2];
```

```
   pid_t pid;

   if(pipe(fd1) == -1){
      printf("Unable to create pipe\n");
   }
   if(pipe(fd2) == -1){
      printf("Unable to create pipe\n");
   }

   pid = fork();

   if(pid < 0){
      printf("fork failed\n");
   }
   else{
      if(pid == 0){  // Child block
         close(fd1[1]);

         char str1[100], * str2 = "Child!";
         read(fd1[0], str1, 100);

         close(fd1[0]);

         close(fd2[0]);

         write(fd2[1], strcat(str1, str2), strlen(str1)+strlen(str2));

         close(fd2[1]);

      }
      else{          // Parent block
         close(fd1[0]);

         char* str = "Parent!";
         write(fd1[1], str, strlen(str));
         close(fd1[1]);

         wait(NULL);

         close(fd2[1]);

         char cat_str[100];
         read(fd2[0], cat_str, 100);
         printf("Concatenated String: %s\n", cat_str);

         close(fd2[0]);
      }
   }
   return 0;
}
```

Output:



```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./2
Concatenated String: Parent!Child!
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

Question 3: Substring generation at child end of a string setup at parent process end.

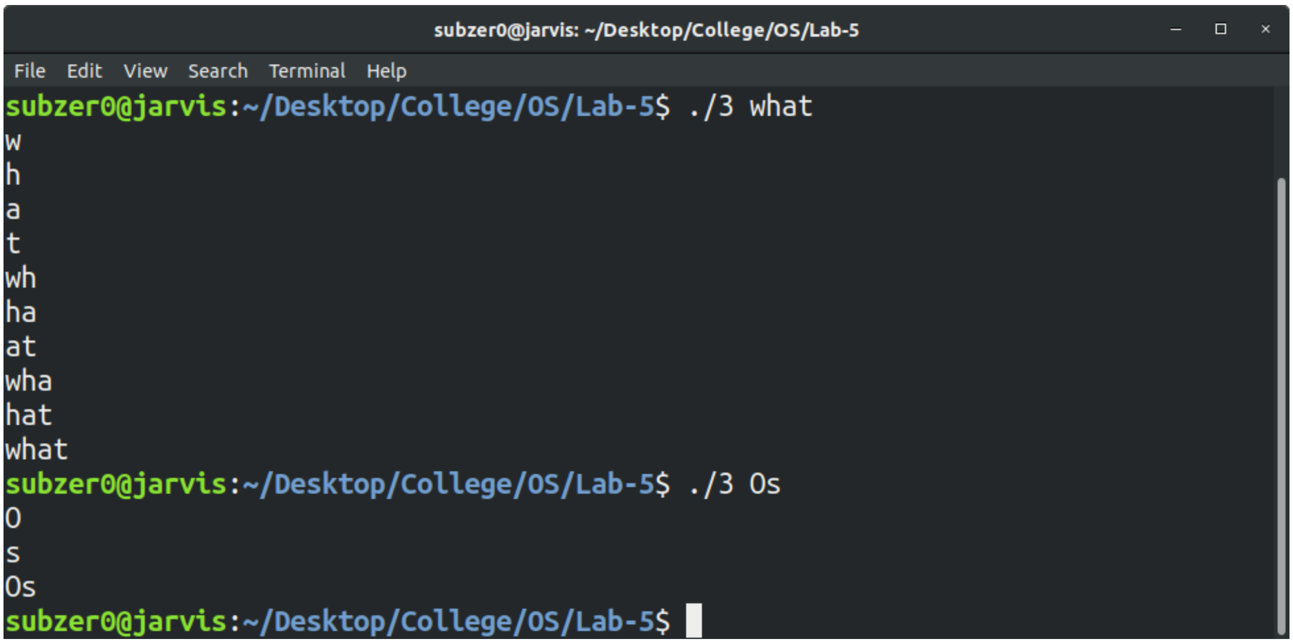
Code:

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

int main(int argc, char const *argv[])
{
   int fd1[2], fd2[2];
   pid_t pid;

   if(pipe(fd1) == -1){
      printf("Unable to create pipe\n");
   }

   pid = fork();
   if(pid < 0){
      printf("fork failed\n");
   }
   else{
      if(pid == 0){  // Child block
         close(fd1[1]);

         char str1[100];
         read(fd1[0], str1, 100);

            for(int len=1; len<=strlen(str1); len+=1){
               for(int ptr=0; ptr+len-1<strlen(str1); ptr+=1){
                  for(int i=ptr; i<ptr+len; i++){
                     printf("%c", str1[i]);
                  }
                  printf("\n");
               }
            }

         close(fd1[0]);
      }
      else{         // Parent block
         close(fd1[0]);

         write(fd1[1], argv[1], strlen(argv[1]));
         close(fd1[1]);

         wait(NULL);
      }
   }
   return 0;
}
```

Explanation:

Parent sends a string to child through a pipe. Child reads the string and finds all the possible substrings and prints it to terminal.

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./3 what
w
h
a
t
wh
ha
at
wha
hat
what
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./3 Os
O
s
Os
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ 
```

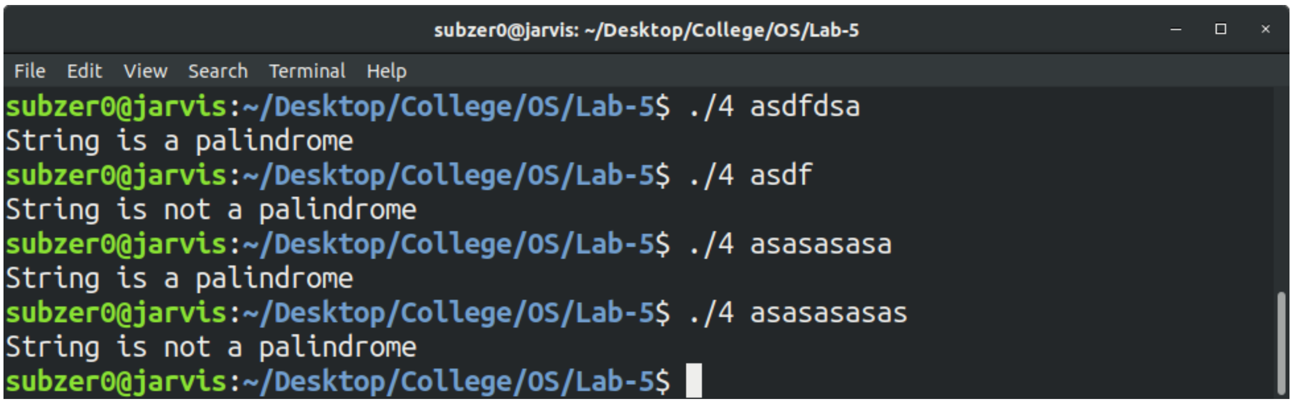
Question 4: String reversal and palindrome check using pipes / shared memory.

Code:

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

char* strrev (char*);

int main(int argc, char const *argv[])
{
   int fd1[2], fd2[2];
   pid_t pid;

   if(pipe(fd1) == -1){
      printf("Unable to create pipe\n");
   }
   if(pipe(fd2) == -1){
      printf("Unable to create pipe\n");
   }

   pid = fork();

   if(pid < 0){
      printf("fork failed\n");
   }
   else{
      if(pid == 0){  // Child block
         close(fd1[1]);
```

```
        char str[100] = {0};
        read(fd1[0], str, 100);

        close(fd1[0]);

        close(fd2[0]);
           char temp[100];
           strcat(temp, str);
           char* rev = strrev(temp);

        if(strcmp(str, rev) == 0){
              write(fd2[1], "1", 1);
           }
           else
              write(fd2[1], "0", 1);

        close(fd2[1]);

     }
     else{          // Parent block
        close(fd1[0]);

        write(fd1[1], argv[1], strlen(argv[1]));
        close(fd1[1]);

        wait(NULL);

        close(fd2[1]);

        char result[1];
        read(fd2[0], result, 1);

           if(result[0] == '0'){
              printf("String is not a palindrome\n");
           }
           else{
              printf("String is a palindrome\n");
           }

        close(fd2[0]);
     }
  }
  return 0;
}

char *strrev(char *str)
{
    char *itr1, *itr2;

    for (itr1=str, itr2=str+strlen(str)-1; itr2>itr1; ++itr1, --itr2){
        *itr1 ^= *itr2;
        *itr2 ^= *itr1;
        *itr1 ^= *itr2;
    }
    return str;
}
```

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asdfdsa
String is a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asdf
String is not a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asasasasa
String is a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asasasasas
String is not a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

Question 5: Armstrong number generation within a range. The digit extraction, cubing can be responsibility of child while the checking for sum == no can happen in child and the output list in the child.

Code:

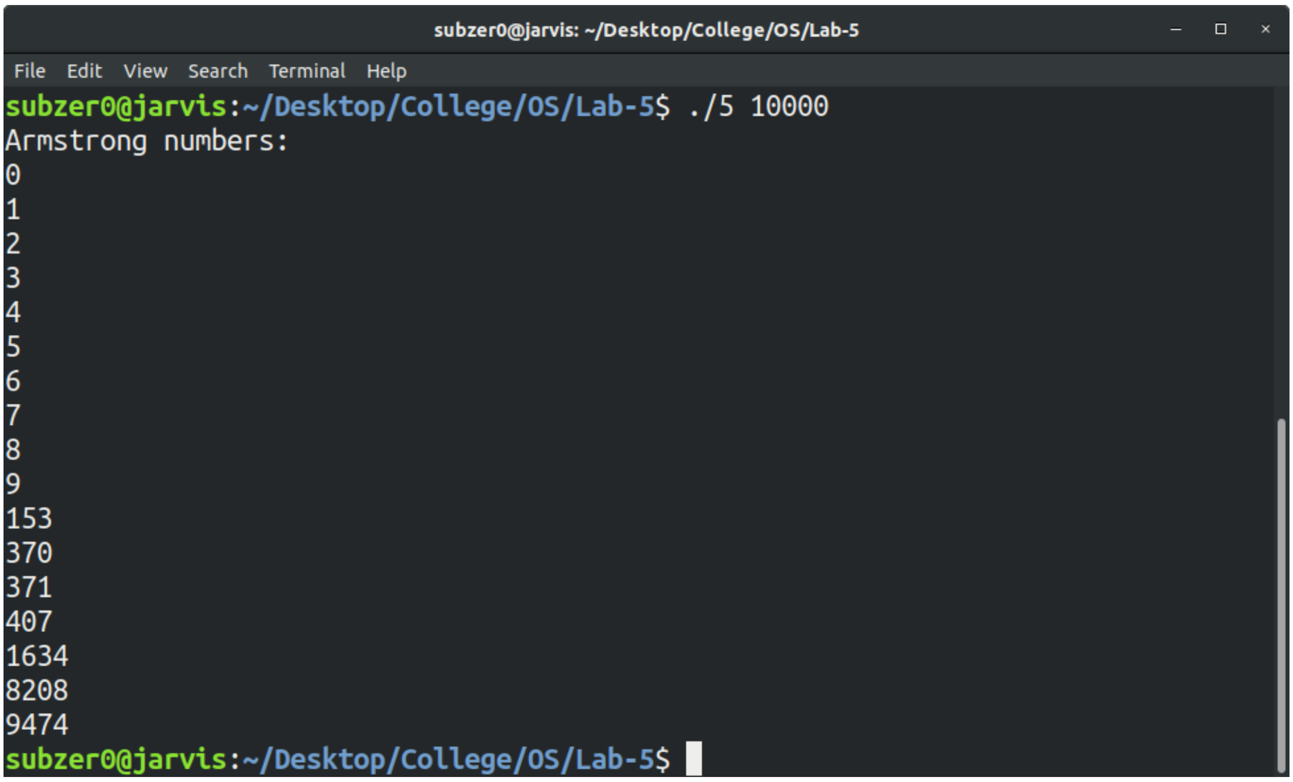
```c
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>
#include<math.h>
#include<sys/ipc.h>
#include<sys/shm.h>

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

   int n=atoi(argv[1]);

   int i=0, num=0, sum=0, pow_to=0, j=0;
   struct armstrong {
      int numbers[100];
      int n;
   };

   key_t key = ftok("shmfile", 65);
   int shmid = shmget(key, 5*sizeof(struct armstrong), 0666|IPC_CREAT);
   struct armstrong *armstg = shmat(shmid, 0, 0);

   for(i=0; i<n; i=i+1){

      sum = 0;
      num = i;
      pow_to = 0;
      for(; num>0; num/=10)
         pow_to+=1;
      num = i;

      pid_t pid = fork();
      if(pid == 0){
         while (num>0){
            int digit = num%10;
            sum = sum + pow(digit, pow_to);
            num = num/10;
         }
         if(sum == i){
            armstg->numbers[armstg->n] = i;
```

```
                armstg->n += 1;
            }
            exit(0);
        }

    }

    wait(NULL);

    printf("Armstrong numbers:\n");
    for(int i=0; i<armstg->n; i++)
        printf("%d\n", armstg->numbers[i]);

    shmdt(armstg);
    shmctl(shmid,IPC_RMID,NULL);

    return 0;
}
```

Explanation:

This program was done using shared memory. The parent forks for every number in the range. The child processes find whether the number they forked at is an armstrong number or not. If the number is an armstrong number, then it is appended to an array where all the armstrong numbers within that range are stored. This array is made available to all the processes using shared memory. A structure was used to store both the array the next iterator to that array.

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./5 10000
Armstrong numbers:
0
1
2
3
4
5
6
7
8
9
153
370
371
407
1634
8208
9474
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```