COM301P- Dr.Sivaselvan

OS Lab Assignment 5

Animesh Kumar (CED18I065)

21st October, 2020

Output of codes given in class(theory)

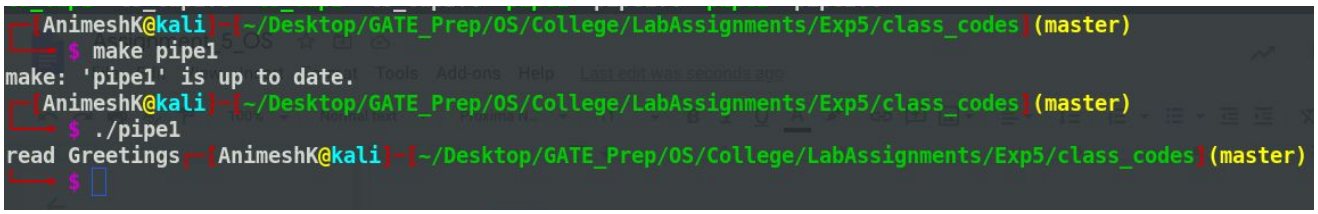
- Code 1:

```c
#include <sys/types.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>



#define BUFFER_SIZE 25

#define READ_END 0

#define WRITE_END 1



int main(void) {

 char write_msg[BUFFER_SIZE] = "Greetings";

 char read_msg[BUFFER_SIZE];

 int fd[2];

 pid_t pid;



 /* create the pipe */

 if (pipe (fd) == -1) {

   fprintf (stderr, "Pipe failed");

   return 1;

 }
```

```
/* fork a child process */
pid = fork();

if (pid < 0) { /* error occurred */
  fprintf (stderr, "Fork Failed");
  return 1;
}

if (pid > 0) { /* parent process */
  /* close the unused end of the pipe */
  close (fd[READ_END]);
  /* write to the pipe */
  write(fd[WRITE_END], write_msg, strlen(write_msg) +1);
  /* close the write end of the pipe */
  close(fd[WRITE_END]);
}

else { /* child process */
  /* close the unused end of the pipe */
  close (fd [WRITE_END]);
  /* read from the pipe */
  read(fd[READ_END], read_msg, BUFFER_SIZE);
```

```
   printf("read %s", read_msg);

   /* close the write end of the pipe */

   close(fd[READ_END]);

 }

 return 0;


}
```

- Output:



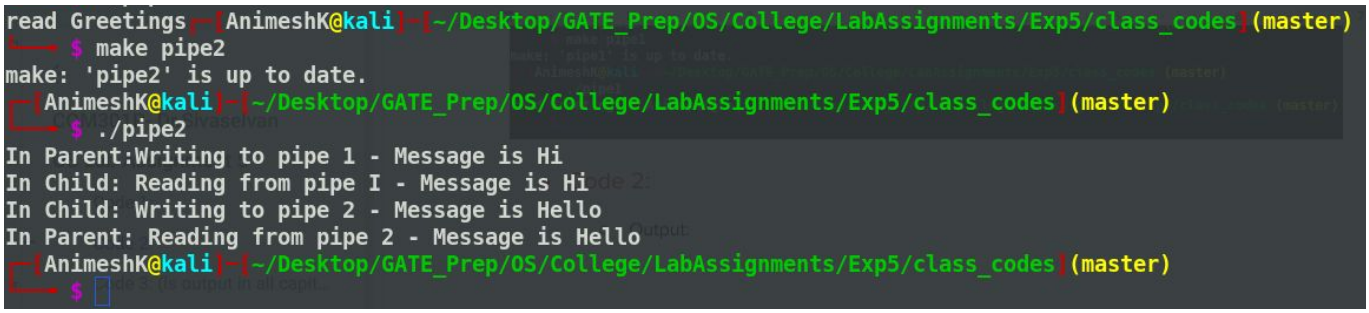
- Code 2:

```
#include<stdio.h>



#include<unistd.h>



int main() {

 int pipefds1[2], pipefds2[2];

 int returnstatus1 , returnstatus2;

 int pid;

 char pipe1writemessage[20] = "Hi";
```

```
 char pipe2writemessage[20] = "Hello";

 char readmessage[20];

 returnstatus1 = pipe(pipefds1);

 if (returnstatus1 == -1)  {

   printf("Unable to create pipe 1 \n");

   return 1;

 }

 returnstatus2 = pipe (pipefds2);


 if (returnstatus2 == -1) {

   printf("Unable to create pipe 2 \n");

   return 1;

 }

 pid = fork();

 if (pid > 0) {// Parent process

   close(pipefds1[0]); // Close the unwanted pipe1 read side

   close(pipefds2[1]); // Close the unwanted pipe2 write side

   printf("In Parent:Writing to pipe 1 - Message is %s\n",
pipe1writemessage);

   write(pipefds1[1], pipe1writemessage,
sizeof(pipe1writemessage)+1);

   read(pipefds2[0], readmessage, sizeof(readmessage));

   printf("In Parent: Reading from pipe 2 - Message is %s\n",
readmessage);
```

```
 }

 else  {

   close(pipefds1[1]); // Close the unwanted pipe1 write side

   close(pipefds2[0]); // Close the unwanted pipe2 read side

   read(pipefds1[0], readmessage, sizeof(readmessage));

   printf("In Child: Reading from pipe I - Message is %s\n",
readmessage);

   printf("In Child: Writing to pipe 2 - Message is %s\n",
pipe2writemessage);

   write(pipefds2[1], pipe2writemessage,
sizeof(pipe2writemessage)+1);

 }

 return 0;

}
```

- Output:



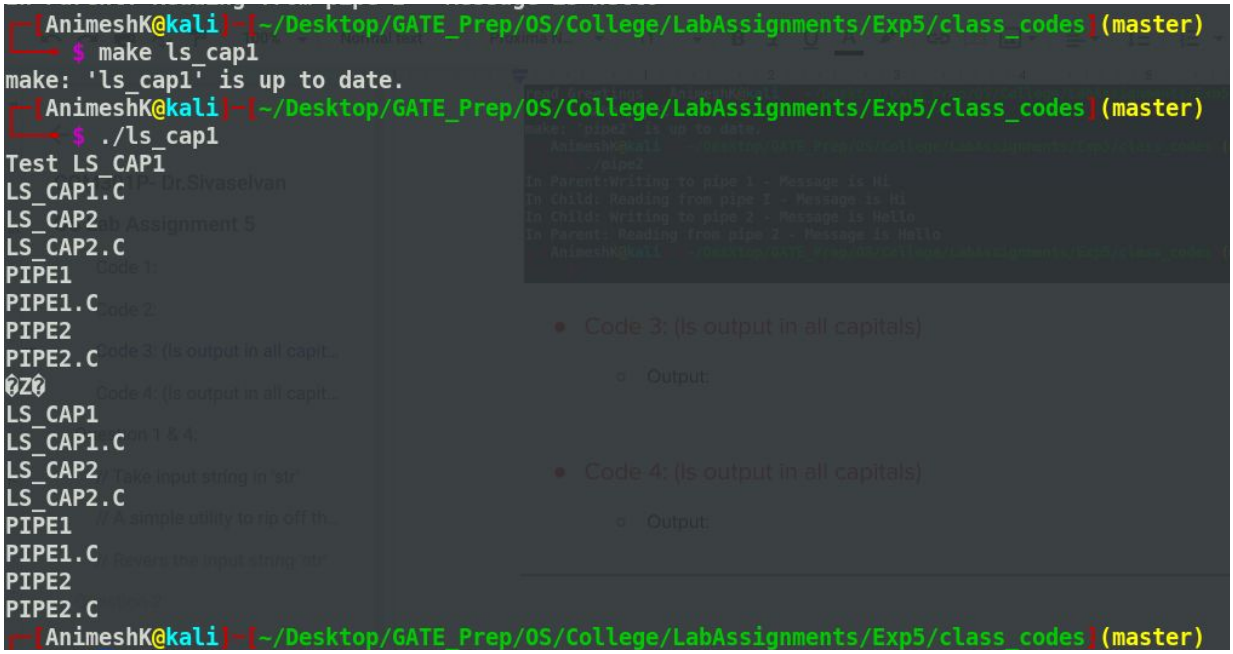
- Code 3: (ls output in all capitals)

```
#include<stdio.h>

#include<stdlib.h>
```

```c
#include<unistd.h>
#include<sys/types.h>
#include<string.h>
#include<sys/wait.h>
#include<pthread.h>
#include <ctype.h>


#define MAX 512


int main(int argc, char* argv[]) {
 int fd[2]; char buf[MAX]; int nb, i;
 if (pipe(fd) == -1) {
   perror("Creating pipe");
   exit(1);
 }
 switch(fork()) {
   case -1:
     perror("Creating a process ");
     exit(1);
   case 0:
     dup2(fd[1], 1);
     execvp("ls", argv); // output of ls command written on
fd[1] courtsey the dup2 call.
```

```
    perror("program ls");
    exit(1);
  default:
    close(fd[1]);
    while ((nb=read(fd[0], buf, MAX)) > 0) {
      for(i=0; i<nb; i++)
        buf[i] = toupper(buf[i]);
      printf("Test %s \n",buf);
      if (write(1, buf, nb) == -1) {
        perror ("Writting to stdout");
        exit(1);
      }
    } // end of while
    if (nb == -1) {
      perror("Reading from pipe");
      exit(1);
    }
} // end of switch
} // main end
```

- Output:



- Code 4: (ls output in all capitals)

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<pthread.h>

#include <ctype.h>



int main () {

 char *cmd1[] = { "/bin/ls", "-al", "/", 0 };
```

```
char *cmd2[] = { "/usr/bin/tr", "a-z", "A-Z", 0 };

int pid; int pfd[2];

pipe (pfd); // other validation code u can repeat from earlier discussion

switch (pid = fork()) {

  case 0: /* child */

    dup2(pfd[0], 0);

    close(pfd[1]); /* the child does not need this end of the pipe */

    execvp(cmd2[0], cmd2); // executes tr command on output of ls stored
at the respective pipe end

    perror(cmd2[0]);


  default: /* parent */

    dup2(pfd[1], 1);

    close(pfd[0]);

    /* the parent does not need this end of the pipe */

    execvp(cmd1[0], cmd1);

    perror(cmd1[0]);

    // ls executed and written to the pipe end of 1

  case -1:

    perror("fork");

    exit(1);

}

return 0;
```

```
}
```

- Output:

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5/class_codes](master)
└─$ ./ls_cap2
TOTAL 58172
DRWXR-XR-X  19 ROOT ROOT     4096 OCT 21 07:26 .
DRWXR-XR-X  19 ROOT ROOT     4096 OCT 21 07:26 ..
-RW-R--R--   1 ROOT ROOT        0 MAR 19  2020 0
LRWXRWXRWX   1 ROOT ROOT        7 JUN 28  2019 BIN -> USR/BIN
DRWXR-XR-X   4 ROOT ROOT     4096 AUG 15 11:39 BOOT
DRWX------   2 ROOT ROOT     4096 JUN 28  2019 .CACHE
DRWXR-XR-X  20 ROOT ROOT     3560 OCT 21 07:26 DEV
DRWXR-XR-X 200 ROOT ROOT    12288 OCT 21 21:24 ETC
-RW-R--R--   1 ROOT ROOT 59477810 AUG  6  2019 GOOGLE-CHROME-STABLE_CURRENT_AMD64.DEB
DRWXR-XR-X   4 ROOT ROOT     4096 JAN 16  2020 HOME
LRWXRWXRWX   1 ROOT ROOT       34 JUN 28  2019 INITRD.IMG -> BOOT/INITRD.IMG-4.19.0-KALI4-AMD64
LRWXRWXRWX   1 ROOT ROOT       34 JUN 28  2019 INITRD.IMG.OLD -> BOOT/INITRD.IMG-4.19.0-KALI4-AMD64
LRWXRWXRWX   1 ROOT ROOT        7 JUN 28  2019 LIB -> USR/LIB
LRWXRWXRWX   1 ROOT ROOT        9 JUN 28  2019 LIB32 -> USR/LIB32
LRWXRWXRWX   1 ROOT ROOT        9 JUN 28  2019 LIB64 -> USR/LIB64
LRWXRWXRWX   1 ROOT ROOT       10 JUN 28  2019 LIBX32 -> USR/LIBX32
DRWX------   2 ROOT ROOT    16384 JUN 28  2019 LOST+FOUND
DRWXR-XR-X   5 ROOT ROOT     4096 JAN 18  2020 MEDIA
DRWXR-XR-X   2 ROOT ROOT     4096 MAY  8  2019 MNT
DRWXR-XR-X   9 ROOT ROOT     4096 AUG  6 07:17 OPT
DR-XR-XR-X 323 ROOT ROOT        0 OCT 20 13:44 PROC
DRWXR-XR-X  32 ROOT ROOT     4096 AUG 30 09:13 ROOT
DRWXR-XR-X  40 ROOT ROOT     1180 OCT 21 22:12 RUN
LRWXRWXRWX   1 ROOT ROOT        8 JUN 28  2019 SBIN -> USR/SBIN
DRWXR-XR-X   3 ROOT ROOT     4096 JUN 28  2019 SRV
DR-XR-XR-X  13 ROOT ROOT        0 OCT 21 22:53 SYS
DRWXRWXRWT  22 ROOT ROOT    12288 OCT 21 23:04 TMP
```

Lab Assignment Solutions

- Question 1: Parent sets up a string which is read by child, reversed there and read back the parent
  &
- Question 4: String reversal and palindrome check using pipes / shared memory.

These problems were similar therefore solved into one code

- Algorithm: Given a string as input at the parent process.
  - The Parent forks and creates a child process and sends the string via pipe.
  - The child process reverses the string and sends the reversed string to the parent and terminates.
  - The Parent checks if the received reverse string is the same as the original, the string is palindrome. Else not a palindrome.
- Output:

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./Q1_Q4
Enter a String(all in lower case):
hell ogawe
The string 'hell ogawe' has the reverse string 'ewago lleh'. Therefore it is not a palindrome.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./Q1_Q4
Enter a String(all in lower case):
halua
The string 'halua' has the reverse string 'aulah'. Therefore it is not a palindrome.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./Q1_Q4
Enter a String(all in lower case):
lihhuhhil
The string 'lihhuhhil' has the reverse string 'lihhuhhil'. Therefore it is a palindrome.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./Q1_Q4
Enter a String(all in lower case):
lol
The string 'lol' has the reverse string 'lol'. Therefore it is a palindrome.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$
```

- C Program:

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<time.h>



#define SIZE 100



void get_input(char * str);

void RipOffNextLine(char * str);

void Reverse(char * str);
```

```c
int main() {
 char str[SIZE];
 int fd1[2];  // Parent writing and Child Reading
 int fd2[2];  // Child writing and Parent Reading
 pipe(fd1);
 pipe(fd2);
 memset(str, 0, SIZE);
 pid_t pid = fork();
 if (pid == 0) {  // Child Process
   // Take string from parent
   close(fd1[1]);
   read(fd1[0], str, SIZE);
   close(fd1[0]);

   Reverse(str);

   // Give back the reversed string to parent
   close(fd2[0]);
   write(fd2[1], str, SIZE);
   close(fd2[1]);
   exit(0);
 }
```

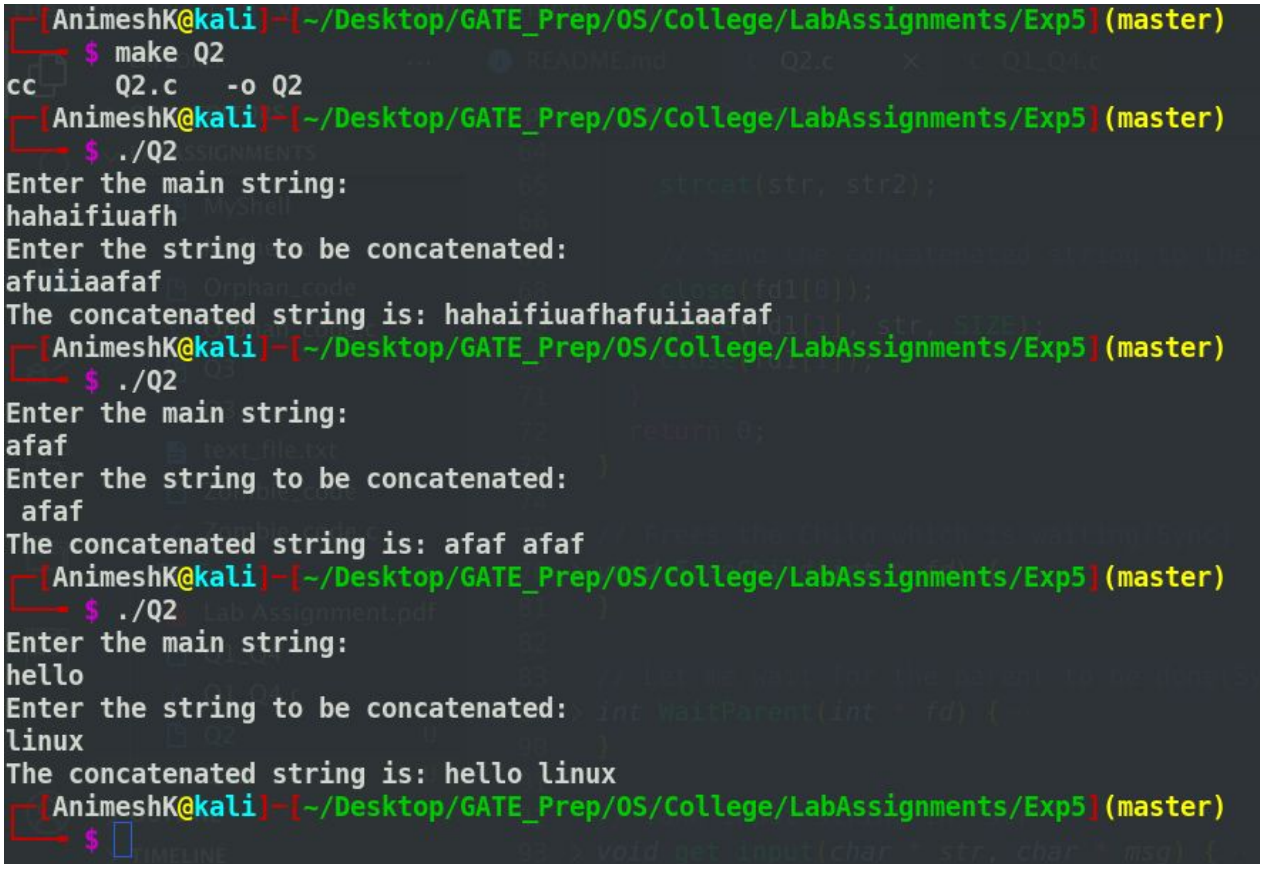
```
else {
  char rev_str[SIZE];
  memset(rev_str, 0, SIZE);
  get_input(str);

  // Send the read string to the child
  close(fd1[0]);
  write(fd1[1], str, SIZE);
  close(fd1[1]);

  // Get the reversed string from the child
  close(fd2[1]);
  read(fd2[0], rev_str, SIZE);
  close(fd2[0]);

  printf("The string '%s' has the reverse string '%s'. ", str, rev_str);
  if (!strcmp(str, rev_str))
    printf("Therefore it is a palindrome.\n");
  else
    printf("Therefore it is not a palindrome.\n");
}

return 0;
```

```
}


// Take input string in 'str'
void get_input(char * str) {
 printf("Enter a String(all in lower case): \n");
 fgets(str, SIZE, stdin);
 RipOffNextLine(str);
}


// A simple utility to rip off the trailing '\n' from 'str'
void RipOffNextLine(char * str) {
 int i = 0;
 while(str[i] != '\n')
   i++;
 str[i] = '\0';
}


// Revers the input string 'str'
void Reverse(char * str) {
 char rev_str[SIZE];
 memset(rev_str, 0, SIZE);
 int i = 0;
 int j = 0;
```

```
while(str[i] != '\0')
  i++;
i--;
while(i >= 0) {
  rev_str[j] = str[i];
  i--;
  j++;
}
strcpy(str, rev_str);
// fputs(rev_str, stdout);
}
```

- 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.
    - Algorithm: Given 2 strings as input at the parent process and child process respectively. **This process was synchronised using a locking mechanism, where the child is made to wait using a while loop until the parent has completed taking the input.**
        - The child process sends the string2 to the parent via pipe
        - The Parent receives the string2 from child via pipe, concatenates it after string1 and sends the concatenated string to child via pipe.
        - The pipe receives the concatenated string and prints it to stdout.
        - Child terminates and the parent terminates

- Output:



- C Program:

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<time.h>
```

```c
#define SIZE 100



void get_input(char * str, char * msg);

void RipOffNextLine(char * str);

void FreeChild(int * fd);

int WaitParent(int * fd);


int main() {

 char str[SIZE];

 memset(str, 0, SIZE);

 int fd1[2];

 int fd2[2];

 int fd3[2];  // For Synchronizing the input from stdin by the two processes.

 pipe(fd1);

 pipe(fd2);

 pipe(fd3);

 pid_t pid = fork();

 if (pid == 0) {  // Child Process

   // Keep waiting until parent signals

   while(WaitParent(fd3)) { }
```

```
  char concat_str[SIZE];
  memset(concat_str, 0, SIZE);
  get_input(str, "the string to be concatenated");

  // Send the string to parent
  close(fd2[0]);
  write(fd2[1], str, SIZE);
  close(fd2[1]);

  // Take concatenated string from parent
  close(fd1[1]);
  read(fd1[0], concat_str, SIZE);
  close(fd1[0]);

  printf("The concatenated string is: ");
  fflush(stdout);
  fputs(concat_str, stdout);
  printf("\n");
  exit(0);
}
else {
  char str2[SIZE];
  memset(str2, 0, SIZE);
```

```
   get_input(str, "the main string");
   FreeChild(fd3);

   // Get the string2 from the child
   close(fd2[1]);
   read(fd2[0], str2, SIZE);
   close(fd2[0]);

   strcat(str, str2);

   // Send the concatenated string to the child
   close(fd1[0]);
   write(fd1[1], str, SIZE);
   close(fd1[1]);
 }
 return 0;
}

// Frees the Child which is waiting(Sync)
void FreeChild(int * fd) {
 int let_child_wait = 0;
 close(fd[0]);
 write(fd[1], &let_child_wait, 4);
```

```
 close(fd[1]);
}

// Let me wait for the parent to be done(Sync)
int WaitParent(int * fd) {
 int wait_for_parent = 1;
 close(fd[1]);
 read(fd[0], &wait_for_parent, 4);
 close(fd[0]);
 return wait_for_parent;
}

// Take input string in 'str'
void get_input(char * str, char * msg) {
 printf("Enter %s: \n", msg);
 fgets(str, SIZE, stdin);
 RipOffNextLine(str);
}

// A simple utility to rip off the trailing '\n' from 'str'
void RipOffNextLine(char * str) {
 int i = 0;
 while(str[i] != '\n')
```

```
   i++;
 str[i] = '\0';
}
```

- Question 3: Substring generation at child end of a string setup at parent process end.
    - Algorithm: Given a string and two indices as the two ends of the required substring as input at the parent process.
        - The parent sends the string and indices to the child via pipe and the child process receives it.
        - The child then calculates the required substring and sends it back to the parent process. The child terminates.
        - The parent process receives the substring from the child and prints it on stdout.
        - The parent terminates
    - Output:

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ make Q3
make: 'Q3' is up to date.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./Q3
Enter the string:
hello world better place
Enter the first and second index respectively:
3 18
The requested substring is: lo world better
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./Q3
Enter the string:
unconsiousness kills
Enter the first and second index respectively:
2 10
The requested substring is: consiousn
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./Q3
Enter the string:
Establish the right basis before action
Enter the first and second index respectively:
6 24
The requested substring is: ish the right basis
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$
```

- C Program:

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<time.h>



#define SIZE 100
```

```c
void get_substring(char * str, char * substring, int * indices);

void get_input(char * str, int * indices);

void RipOffNextLine(char * str);


int main() {

 char str[SIZE];

 int indices[2];

 memset(str, 0, SIZE);

 int fd1[2];

 int fd2[2];

 int fd3[2];  // For Synchronizing the input from stdin by the two processes.

 pipe(fd1);

 pipe(fd2);

 pipe(fd3);

 pid_t pid = fork();

 if (pid == 0) {  // Child Process

   // Take string from parent

   close(fd1[1]);

   read(fd1[0], str, SIZE);

   close(fd1[0]);


   // Take indices from parent
```

```
    close(fd2[1]);
    read(fd2[0], indices, 8);
    close(fd2[0]);

    char substring[indices[1] - indices[0] + 2];
    get_substring(str, substring, indices);

    // Send the generated substring
    close(fd3[0]);
    write(fd3[1], substring, indices[1] - indices[0] + 2);
    close(fd3[1]);
    exit(0);
}
else {              // Parent Process
    get_input(str, indices);
    char substring[indices[1] - indices[0] + 2];
    // Send the string to the child
    close(fd1[0]);
    write(fd1[1], str, SIZE);
    close(fd1[1]);

    // Send the indices to the child
    close(fd2[0]);
```

```
    write(fd2[1], indices, 8);
    close(fd2[1]);

    // Get the substring from the child
    close(fd3[1]);
    read(fd3[0], substring, indices[1] - indices[0] + 2);
    close(fd3[0]);

    printf("The requested substring is: ");
    fputs(substring, stdout);
    printf("\n");
  }
  return 0;
}

// Take input string in 'str' and get indices of the substring
void get_input(char * str, int * indices) {
  printf("Enter the string: \n");
  fgets(str, SIZE, stdin);
  RipOffNextLine(str);
  printf("Enter the first and second index respectively: \n");
  scanf("%d %d", &indices[0], &indices[1]);
}
```

```
// A utility which returns required substring in 'substring'
void get_substring(char * str, char * substring, int * indices) {
 strncpy(substring, str + indices[0], indices[1] - indices[0] + 1);
 substring[indices[1] - indices[0] + 1] = '\0';
}



// A simple utility to rip off the trailing '\n' from 'str'
void RipOffNextLine(char * str) {
 int i = 0;
 while(str[i] != '\n')
   i++;
 str[i] = '\0';
}
```

- Question 5: Armstrong number generation within a range. The digit extraction, cubing can be the responsibility of the child while the checking for sum == no can happen in the child and the output list in the child.(multithreading)
    - Algorithm: Input is a and b. The program will print all the armstrong numbers between a and b, both inclusive. We package the data (the number and its powered sum) into a structure called `struct block` .
        - For every number in the closed interval [a, b] we create a runner thread.
        - The runner thread calculates powered sum using `get_powered_sum(int n)` and compares it with the number itself.
        - The `get_powered_sum(int n)` extracts the digits and using a power function it finds the sum, then this sum value is returned back to the respective runner thread
        - The runner checks whether the sum value is equal to the number, if yes then outputs that number.
    - Output:

```
┌──[AnimeshK@kali]-[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ gcc Q5_t.c -lpthread
┌──[AnimeshK@kali]-[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./a.out
Enter a and b for the range [a, b] both inclusive: 1 10000
The Armstrong numbers in the range [a, b]:
1
2
3
4
5
6
7
8
9
153
370
371
407
1634
8208
9474
┌──[AnimeshK@kali]-[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$
```

- C Program:

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<pthread.h>



// A block to hold the data to passed among threads

struct block {

 int num;       // the number
```

```
 int pow_sum;  // the powered sum of the number NUM
};



int get_powered_sum(int n);
void *runner(void * params);
int Pow(int n, int x);


int main() {
 int a, b;
 printf("Enter a and b for the range [a, b] both inclusive: ");
 scanf("%d %d", &a, &b);


 pthread_t tid[b - a + 1];       // set of all TIDs
 struct block args[b - a + 1];  // set of all argument BLOCKSs
 pthread_attr_t attr;
 pthread_attr_init(&attr);
  printf("The Armstrong numbers in the range [a, b]: \n");


 // Creating threads for all numbers
 for (int i = a; i <= b; i++) {
   args[i - a].num = i;
   pthread_create(&tid[i - a], &attr, runner, &args[i - a]);
```

```
 }


 // Waiting for all threads to complete(pthread_join())
 for (int i = a; i <= b; i++) {
   pthread_join(tid[i - a], NULL);
 }
 return 0;
}


// The thread routine to be run
void *runner(void * params) {
 struct block * args = params;
 args->pow_sum = get_powered_sum(args->num);
 if (args->num == args->pow_sum)
   printf("%d\n", args->num);
 pthread_exit(NULL);
}


// powered sum of digits of N
int get_powered_sum(int n) {
 int len = 0;
 int sum = 0;
 int i = n;
```

```
 while(i != 0) {
   i /= 10;
   len++;  // length calculation
 }
 i = n;
 while(i != 0) {
   sum += Pow((i % 10), len);  // pow_sum calculation
   i /= 10;
 }
 return sum;
}


// Power function (n^x)
int Pow(int n, int x) {
 if (x == 0)    return 1;
 int prod = n;
 while(x > 1) {
   prod *= n;
   x--;
 }
 return prod;
}
```

- Question 6: Ascending Order sort within Parent and Descending order sort (or vice versa) within the child process of an input array. (u can view as two different outputs –first entire array is ascending order sorted in op and then the second part descending order output).(multithreading)

    - Algorithm: The input is n, and then follows n integers.
        - The parent process sorts the array in ascending order and prints the sorted array via pipes.
        - Concurrently the child process sorts the array in descending order and sends the sorted array to parent via pipes. The child terminates.
        - The parent receives the descending sorted array. Then it prints both the ascending and descending sorted arrays then terminates.
    - Output:

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ gcc Q6_t../a.out
Enter the number of integers: 12
12 -34 31 60 -84 32 90 10 16 72 56 100
Sorted Array in Ascending Order: -84 -34 10 12 16 31 32 56 60 72 90 100
Sorted Array in Descending Order: 100 90 72 60 56 32 31 16 12 10 -34 -84
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ gcc Q6_t.c -lpthread
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./a.out
Enter the number of integers: 12
12 -34 31 60 -84 32 90 10 16 72 56 100
Sorted Array in Ascending Order: -84 -34 10 12 16 31 32 56 60 72 90 100
Sorted Array in Descending Order: 100 90 72 60 56 32 31 16 12 10 -34 -84
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./a.out
Enter the number of integers: 7
12 32 45 -6 0 -83 120
Sorted Array in Ascending Order: -83 -6 0 12 32 45 120
Sorted Array in Descending Order: 120 45 32 12 0 -6 -83
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$
```

- C Program:

```c
#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdlib.h>

#include<time.h>

#include<pthread.h>



#define ASC 1      // for ascending order

#define DESC 0     // for descending order



// A structure to hold the data to be passed to the threads
```

```c
struct block {
 int * arr;    // the original array
 int arr_size; // the arr_size
 int * mod_arr; // the sorted array
 int order;    // ASC or DESC
};


void arrncpy(int * dest, int * src, int size);
void* runner(void* params);
void desc_sort(int * arr, int size);
void asc_sort(int * arr, int size);
void print(int * arr, int size);
void my_swap(int* x, int* y);


int main() {
 int n;
 pthread_t tid1, tid2;
 pthread_attr_t attr;
 struct block first, second;
 printf("Enter the number of integers: ");
 scanf("%d", &n);
 int arr[n], asc_sorted[n], desc_sorted[n];
 for(int i = 0; i < n; i++)
```

```
  scanf("%d", &arr[i]);


// Setting up the parameters to be passed to the threads
first.arr = arr;
first.arr_size = n;
first.order = ASC;
first.mod_arr = asc_sorted;


second.arr = arr;
second.arr_size = n;
second.order = DESC;
second.mod_arr = desc_sorted;


pthread_attr_init(&attr);


// Creating the two threads
pthread_create(&tid1, &attr, runner, &first);  // for ascending sort
pthread_create(&tid2, &attr, runner, &second); // for descending sort
pthread_join(tid1, NULL);
pthread_join(tid2, NULL);


printf("Sorted Array in Ascending Order: ");
print(first.mod_arr, n);
```

```
 printf("Sorted Array in Descending Order: ");
 print(second.mod_arr, n);
 return 0;
}


// The thread routine which triggers asc_sort/desc_sort based on 'params'
void* runner(void* params) {
 struct block * args = params;
 arrncpy(args->mod_arr, args->arr, args->arr_size);
 if (args->order == ASC)
   asc_sort(args->mod_arr, args->arr_size);
 else
   desc_sort(args->mod_arr, args->arr_size);
 pthread_exit(NULL);
}


// Copy SRC to DEST array of SIZE
void arrncpy(int * dest, int * src, int size) {
 for (int i = 0; i < size; i++)
   dest[i] = src[i];
}


// Sorting in ascending order
```

```
void asc_sort(int * arr, int size) {
 for (int i = 0; i < size - 1; i++) {
   for (int j = 0; j < size - i - 1; j++) {
     if (arr[j] > arr[j + 1])
       my_swap(&arr[j], &arr[j + 1]);
   }
 }
}


// Sorting in descending order
void desc_sort(int * arr, int size) {
 for (int i = 0; i < size - 1; i++) {
   for (int j = 0; j < size - i - 1; j++) {
     if (arr[j] < arr[j + 1])
       my_swap(&arr[j], &arr[j + 1]);
   }
 }
}


// swap two integers
void my_swap(int* x, int* y) {
 int temp = *x;
 *x = *y;
```

```
 *y = temp;
}


// print the array arr
void print(int * arr, int size) {
 for(int i = 0; i < size; i++)
   printf("%d ", arr[i]);
 printf("\n");
}
```

- Question 7:Implement a multiprocessing version of binary search where the parent searches for the key in the first half and subsequent splits while the child searches in the other half of the array. By default u can implement a search for the first occurrence and later extend to support multiple occurrences (duplicated elements search as well). (using multithreading)
    - Algorithm: Given the input unsorted array and key to be searched. Two threads are created. We use a `struct block` to hold the data to be passed as a parameter while various thread creations.
        - The first thread sorts the first half of the array in ascending order and searches the key to be searched using BinarySearch() in this part.

- The second thread sorts the other half of the array in ascending order and searches the key to be searched using BinarySearch() in this part.

  void binary_search(int arr[],int key, int l, int h, int hot[]);

  This binary_search version is a multithreading version using recursive strategy.

1. if (l > h) return; // Base case
2. If arr[mid] == key
   - Two new threads with tid1 and tid2 are created.
   - The right thread(tid2) finds all the occurrences of the key in the second half of the arr[] recursively..
   - The left thread(tid1) finds all the occurrences of the key in the first half of the arr[] recursively.
   - Later the main() thread prints the appropriate indices in both parts of the sorted array using show() method.
3. else if arr[mid] < key (thread creation not needed)
   - recursively search in the arr[] in the range(mid + 1, h)
4. else (thread creation not needed)
   - recursively search in the arr[] in the range(l, mid - 1)

- The main() thread terminates.

- Output:

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ gcc Q7_t.c -lpthread
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./a.out
Enter the number of integers: 13
Enter the array:12 47 54 47 45 12 -3 0 12 12 47 45 48
Enter the key to be searched: 47

Sorted second half Array in Ascending Order: -3 0 12 12 45 47 48

Sorted first half Array in Ascending Order: 12 12 45 47 47 54
The key is present at these locations in the second half of sorted array:
11
The key is present at these locations in the first half of sorted array:
3 4
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./a.out
Enter the number of integers: 13
Enter the array:12 47 54 47 45 12 -3 0 12 12 47 45 48
Enter the key to be searched: 12

Sorted second half Array in Ascending Order: -3 0 12 12 45 47 48

Sorted first half Array in Ascending Order: 12 12 45 47 47 54
The key is present at these locations in the first half of sorted array:
0 1
The key is present at these locations in the second half of sorted array:
8 9
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
```

- C Program:

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<pthread.h>
```

```
// A structure to pass data among threads
struct block {
 int * array;     // the array input
 int * hot;       // the hot array
 int key;         // the key to be searched
 int start;       // searching start index
 int end;          // searching end index
 char msg[64];      // "first" or "second"
};


void* runner_main(void * params);
void* runner(void * params);
void asc_sort(int * arr, int start, int end, char * msg);
void print(int * arr, int start, int end);
void my_swap(int* x, int* y);
void binary_search(int arr[],int key, int l, int h, int hot[]);
void show(int hot[], int l, int h);


int n; //  number of integers


int main() {
 int key;
 pthread_t tid1, tid2;
```

```
pthread_attr_t attr;

printf("Enter the number of integers: ");

scanf("%d", &n);

int arr[n];

printf("Enter the array:");

for(int i = 0; i < n; i++)

  scanf("%d", &arr[i]);

printf("Enter the key to be searched: ");

scanf("%d", &key);

int hot[n];

for (int i = 0; i < n; i++) hot[i] = 0;


struct block main, left_thread, right_thread;

main.array = arr;

main.hot = hot;

main.key = key;

 left_thread = main;

left_thread.start = 0;

left_thread.end = n / 2 - 1;

strcpy(left_thread.msg, "first");


right_thread = main;

right_thread.start = n / 2;
```

```
 right_thread.end = n - 1;
 strcpy(right_thread.msg, "second");


 pthread_attr_init(&attr);
 pthread_create(&tid1, &attr, runner_main, &left_thread);
 pthread_create(&tid2, &attr, runner_main, &right_thread);
 pthread_join(tid1, NULL);
 pthread_join(tid2, NULL);
 return 0;
}


// Multithreading binary search version for all occurrences of key
void binary_search(int arr[],int key, int l, int h, int hot[]) {
 if (l > h)      // Base case
   return;
 int mid = (l + h) / 2;
 if (arr[mid] == key) {
   hot[mid] = 1;
   pthread_t tid1, tid2;
   pthread_attr_t attr;
   struct block left, right, main;
   main.array = arr;
   main.key = key;
```

```
  main.hot = hot;


  // setting up arguments for the left thread
  left = main;
  left.start = l;
  left.end = mid - 1;


  // setting up arguments for the right thread
  right = main;
  right.start = mid + 1;
  right.end = h;


  pthread_attr_init(&attr);


  // Calculate fib(n-1) and fib(n-2) by initiating thread1 and thread2
  pthread_create(&tid1, &attr, runner, &left); // search 'key' in left
half
  pthread_create(&tid2, &attr, runner, &right); // search 'key' in right
half
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
  pthread_exit(NULL);
  // return;
 }
```

```
 else if (arr[mid] < key)
   return binary_search(arr, key, mid + 1, h, hot);
 else
   return binary_search(arr, key, l, mid - 1, hot);
}

/* Thread routine which sorts and binary searches in
one(left/right) part of the input array */
void* runner_main(void * params) {
 struct block * args = params;
 asc_sort(args->array, args->start, args->end, args->msg);
 binary_search(args->array, args->key, args->start, args->end, args->hot);
 printf("The key is present at these locations in the %s half of sorted
array: \n", args->msg);
 show(args->hot, args->start, args->end);
 pthread_exit(NULL);
}

// Thread routine which binary searches in one(left/right) part of the
array
void* runner(void * params) {
 struct block * args = params;
 binary_search(args->array, args->key, args->start, args->end, args->hot);
 pthread_exit(NULL);
```

```
}


// print the hot[] location with 1 entries (Printing the Search data)
void show(int hot[], int l, int h) {
 int flag = 0;
 for (int i = l; i <= h; i++) {
   if (hot[i] == 1) {
     flag = 1;
     printf("%d ", i);
   }
 }
 if (flag == 0)
   printf("\nNo such location in this part of the array.");
 printf("\n");
}


// Ascending order sorting and printing
void asc_sort(int * arr, int start, int end, char * msg) {
 for (int i = start; i < end; i++) {
   for (int j = start; j < end + start - i; j++) {
     // printf("hello 22 %d\n", arr[j]);
     if (arr[j] > arr[j + 1]) {
       // printf("hello 2 %d\n", i);
```

```
        my_swap(&arr[j], &arr[j + 1]);
      }
    }
  }
  printf("\nSorted %s half Array in Ascending Order: ", msg);
  print(arr, start, end);
}

// swap two integer variables
void my_swap(int* x, int* y) {
  int temp = *x;
  *x = *y;
  *y = temp;
}

// print the array in range [start, end]
void print(int * arr, int start, int end) {
  for(int i = start; i <= end; i++)
    printf("%d ", arr[i]);
  printf("\n");
}
```

- Question 8: Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a parent child relationship to contribute a faster version of fib series generation. (using multithreading)
    - Algorithm: The input is n. The Program outputs the fibonacci series upto nth fibonacci number.

        int Fib(int n);

        - The Fib(n) function runs two threads
        - The two threads make recursive calls to Fib(n - 1) and Fib(n - 2) respectively. These values are stored in the respective `struct block` of (n - 1) and (n - 2).
        - Inside the main() thread, the Fib(n) sums up the above two values and returns it to the main() function.
        - The parent receives first and second and returns the sum of first and second.
        - After successful execution of Fib function , the value is printed
        - The parent process terminates.
    - Output:

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ gcc Q8_t.c -lpthread
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$ ./a.out
Enter the number n: 15
Fibonacci series upto nth fibonacci number:
Fib(0): 0
Fib(1): 1
Fib(2): 1
Fib(3): 2
Fib(4): 3
Fib(5): 5
Fib(6): 8
Fib(7): 13
Fib(8): 21
Fib(9): 34
Fib(10): 55
Fib(11): 89
Fib(12): 144
Fib(13): 233
Fib(14): 377
Fib(15): 610
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5](master)
└─$
```

- C Program:

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<pthread.h>




// A block helds the number with its fibonacci number

struct block {
```

```
 int num; // the number
 int fib_num; // fibonacci of 'num'
};


int Fib(int n);
void* runner(void *params);


int main() {
 int n;
 printf("Enter the number n: ");
 scanf("%d", &n);
 printf("Fibonacci series upto nth fibonacci number: \n");
 for(int i = 0; i <= n; i++)
   printf("Fib(%d): %d\n", i, Fib(i));
 return 0;
}


// Generates nth fibonacci number
int Fib(int n) {
 if (n < 2)
   return n;
 pthread_t tid1, tid2;
 pthread_attr_t attr;
```

```
 struct block number1, number2;
 number1.num = n - 1;
 number2.num = n - 2;
 pthread_attr_init(&attr);

 // Calculate fib(n-1) and fib(n-2)
 pthread_create(&tid1, &attr, runner, &number1);
 pthread_create(&tid2, &attr, runner, &number2);
 pthread_join(tid1, NULL); // Wait for tid1 thread to complete
 pthread_join(tid2, NULL); // Wait for tid2 thread to complete
 return (number1.fib_num + number2.fib_num); // return the sum
}

// The Thread routine executed by Fib(n)
void* runner(void *params) {
 struct block * b = params;
 b->fib_num = Fib(b->num);
 pthread_exit(NULL);
}
```

- ## (Question 9) EXTRA CREDIT Qn: Longest Common Subsequence(multithreading)

    - Algorithm:

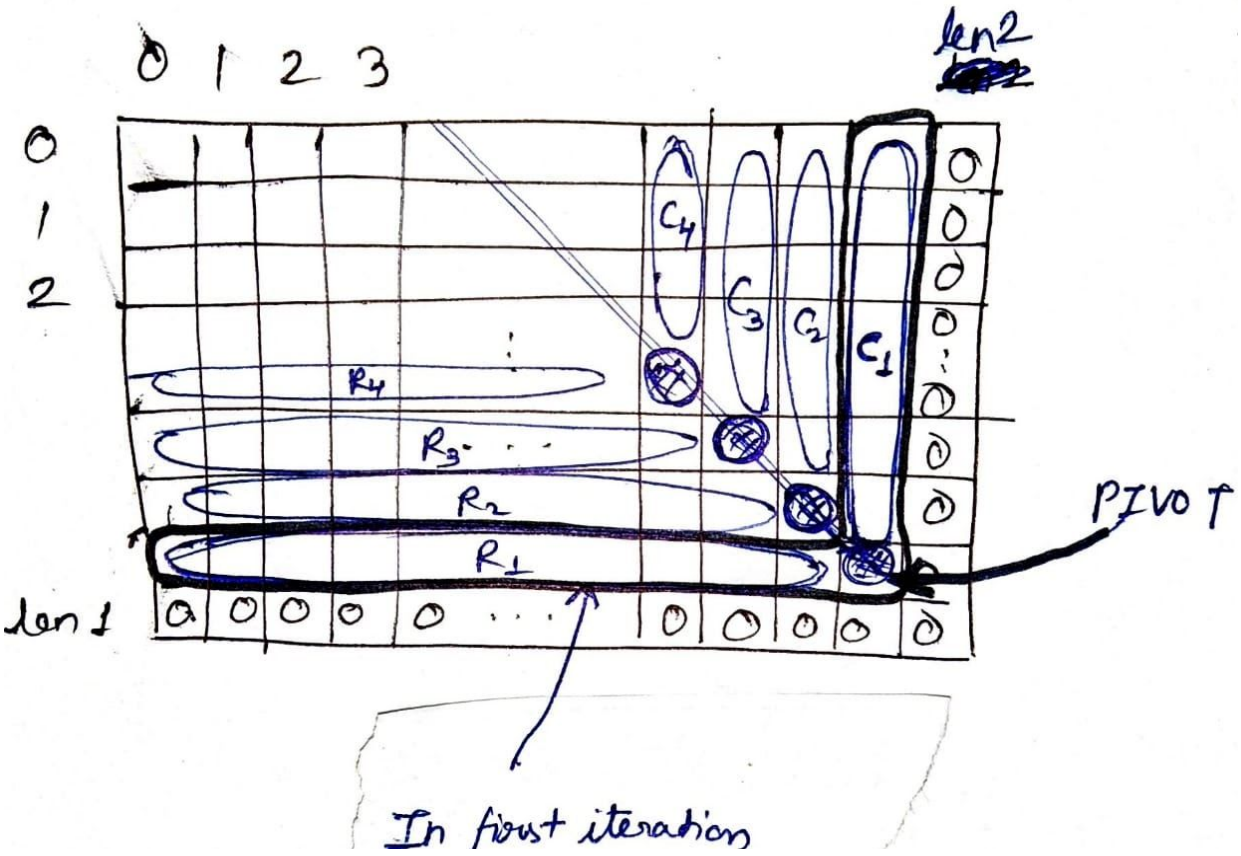
      ### With Dynamic Programming(DP):

      Given two strings str1 and str2 as input. Let idx1 and idx2 be two pointers scanning through the input strings. Let LCS(str1, str2) denote the length of the longest common subsequence.

      - Base Case: If at least one of the pointers has reached the end of their respective string. The LCS will be zero.
      - Inductive Step:
        - if (str1[idx1] is the same as str2[idx2]) then return (1 + LCS(idx1 + 1, idx2 + 1)).
        - else find return max(LCS(idx1 + 1, idx2), LCS(idx1, idx2 + 1)
      - For DP, we'll have to create a table Cache[len1+1][len2+1], where Cache[i][j] represents the LCS of the str1 considered from idx1 and str2 considered from idx2. Our goal is to find Cache[0][0].

      ### Fusing Multithreading using POSIX threads:

      I don't want to explain the traditional approach to fill the matrix in bottom-up fashion. But we shall do it in a parallel approach. It must be self explanatory from the below illustration. Follow 0 based indexing for the below Cache[len1+1][len2+1] table.

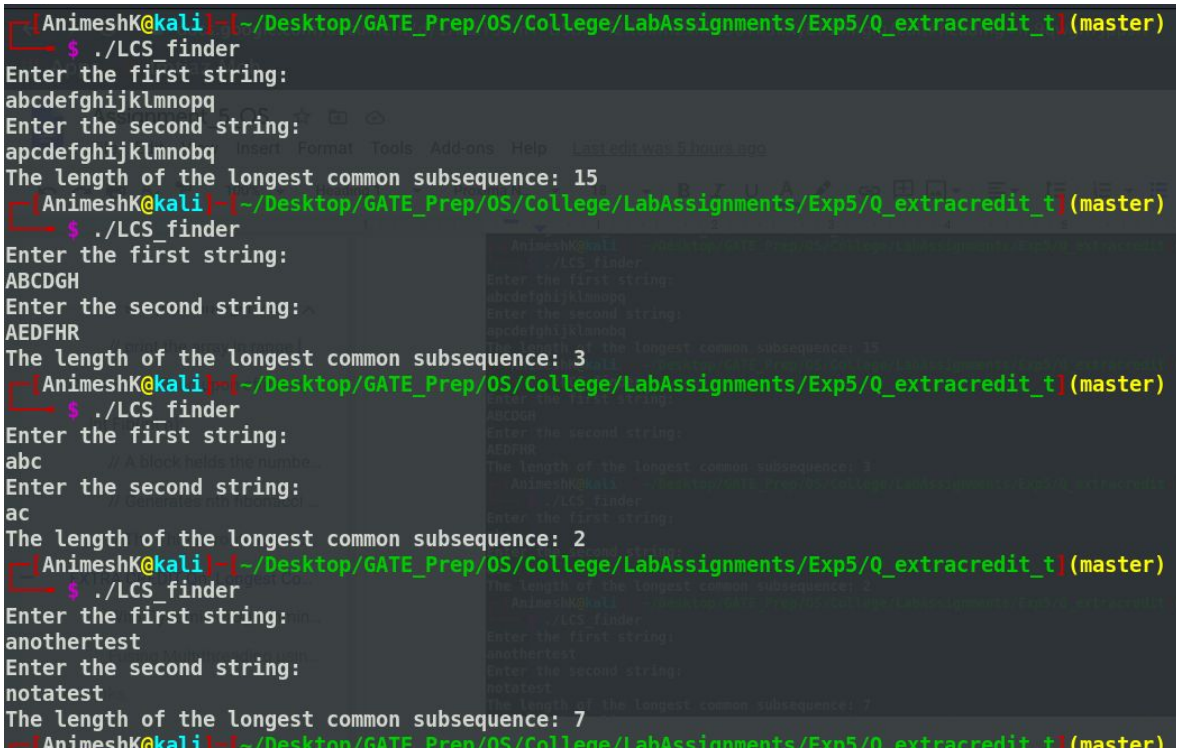


(The inverted 'L' (using black sketch) is performed using one process, the Vertical part of L(column) is by one thread and the horizontal part (row) is by the other thread. This L is identified by the Pivot which moves along y = -x straight line after every iteration)

- The base cases are right-most column and bottom-most row(initialised with 0).
- The Cell (len1 - 1, len2 - 1) becomes the first pivot and is getting filled using the FillCell() method(which implicitly takes care of the DP logic).
- The R1 associated with P1 is filled by the first thread and similarly the C1 associated with P1 is filled by the second thread.
- The next pivot P2 will be the cell (len1 - 2, len2 - 2), and so on the pivots will be assigned along the y = - x straight line.
- Corresponding to every pivot Pi there will be 2 threads calculating Ri and Ci. **This process continues until we are exhausted either by the width of the matrix or the height of it.**

- At the end we return Cache[0][0]. The main process terminates.
- Assuming perfect parallelism, this algorithm takes O(min(len1, len2)) time. We iterate through the while loop creating 1 pivot each time and 2 threads which fill the matrix.

- Output:



```
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5/Q_extracredit](master)
└─► $ ./LCS_finder
Enter the first string:
132535365
Enter the second string:
123456789
The length of the longest common subsequence: 5
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5/Q_extracredit](master)
└─► $ ./LCS_finder
Enter the first string:
ABCDGH
Enter the second string:
AEDFHR
The length of the longest common subsequence: 3
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5/Q_extracredit](master)
└─► $ ./LCS_finder
Enter the first string:
albert
Enter the second string:
berta
The length of the longest common subsequence: 4
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5/Q_extracredit](master)
└─► $ ./LCS_finder
Enter the first string:
12345
Enter the second string:
45321
The length of the longest common subsequence: 2
┌─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp5/Q_extracredit](master)
└─► $
```

- C Program: The size of the program was huge, therefore the code for this question has been attached with the file ***Q_extracredit.zip***.

Thanks,

Animesh Kumar

CED18I065