YCCE

Vision

"To become the most preferred institution providing innovative, research and value based, professional education for the society at large".

Mission

YCCE is committed to

- Attract best talent and create learning ambience
- Practice Innovative teaching-learning & research
- Integrate Industry-Institute Collaborations
- Nurture students towards holistic development and choicest career

Department

Vision of the Department

To be a well-known center for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary arena by developing problem-solving skills through emerging technologies.

Nagar Yuwak Shikshan Sanstha's
Yeshwantrao Chavan College of Engineering
(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)
Hingna Road, Wanadongri, Nagpur - 441 110
Ph.: 07104-237919, 234623, 329249, 329250 Fax: 07104-232376, Website: www.ycce.edu
Department of Computer Science and Engineering (IOT)

23CT1402	Lab: Operating Systems
Name of the Student: Puneet K Raut	Semester/ Section: 5 A
Roll No: 58	Enrollment Number: 23070647

Sr. No.	COs	POs											PSOs	
		Course Outcomes	1	2	3	4	5	6	7	8	9	10	11	PSO 1	PSO2
1	CO1	Demonstrate the ability to execute Linux process management, memory management, and shell commands to manage system resources efficiently.	3	3	3	-		-	-	-	-	-	3	3	
2	CO2	Develop programs utilizing system calls, thread programming, and page replacement algorithms to simulate and analyze operating system functionalities.	3	3	3	-	-	-	-	-	-	-	3	3	
3	CO3	Design and implement process scheduling, memory allocation, and deadlock detection algorithms to address real-world operating system challenges.	3	3	3	-		-	-	-	-	-	3	3	
		Avg		3	3	-		-	-	-	-	-	3	3	

Practical No. 2

Aim: : Execute linux system calls process management ,memory management , file managment system calls

Theory:

Process managment:

This system calls perform the task of process creation, process termination, etc. The Linux System calls under this are fork() , exit() , exec().

- **fork():**
  - A new process is created by the fork() system call.
  - A new process may be created with fork() without a new program being run-the new sub-process simply continues to execute exactly the same program that the first (parent) process was running.
  - It is one of the most widely used system calls under process management.

- **exit():**
  - The exit() system call is used by a program to terminate its execution.
  - The operating system reclaims resources that were used by the process after the exit() system call.

- **exec():**
  - A new program will start executing after a call to exec()
  - Running a new program does not require that a new process be created first: any
  - process may call exec() at any time. The currently running program is immediately terminated, and the new program starts executing in the context of the existing process.

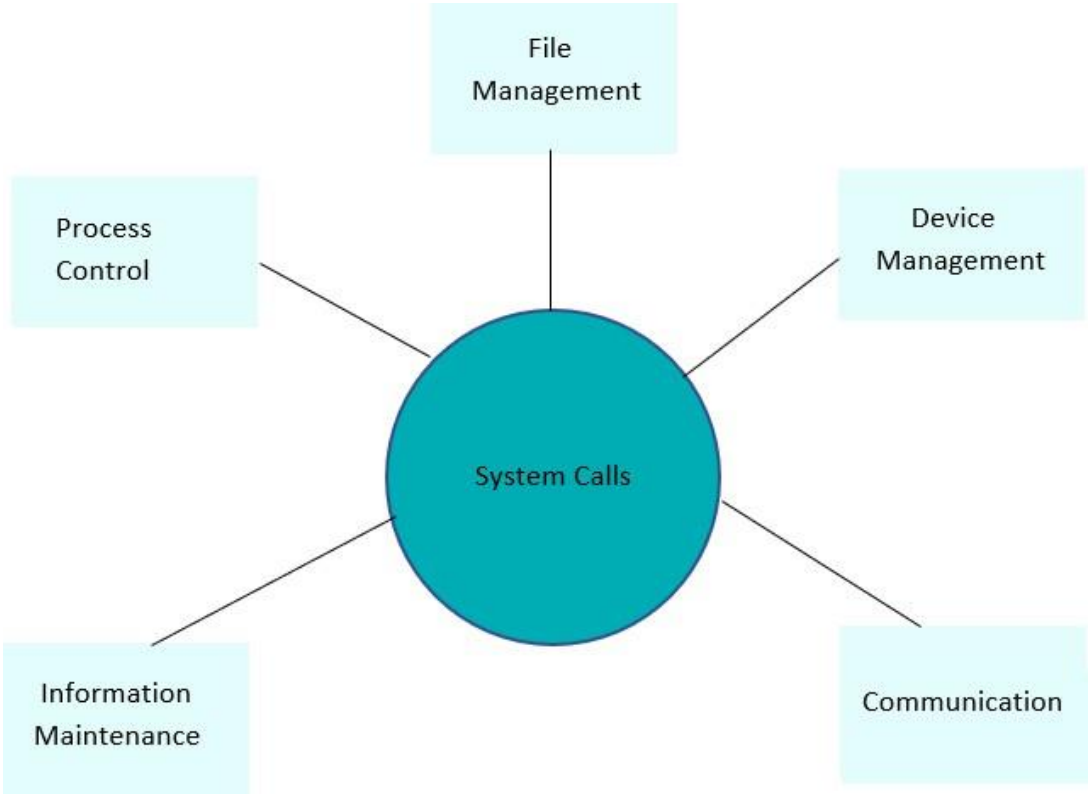
- **File Management :**
  File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc. The Linux System calls under this are open(), read(),write(), close()

  • **open():**
    - It is the system call to open a file.
    - This system call just opens the file, to perform operations such as read and write, we need to execute different system call to perform the operations.

  • **read():**
    - This system call opens the file in reading mode
    - We can not edit the files with this system call.
    - Multiple processes can execute the read() system call on the same file simultaneously.

- **write():**
  - This system call opens the file in writing mode
  - We can edit the files with this system call.
  - Multiple processes can not execute the write() system call on the same file simultaneously.

- **close():**
  - This system call closes the opened file.



Source Code:
File Name:

Sample code 1:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   fork();
   fork();
   fork();
   printf("hello\n");
   return 0;
}
```

Sample code 2:

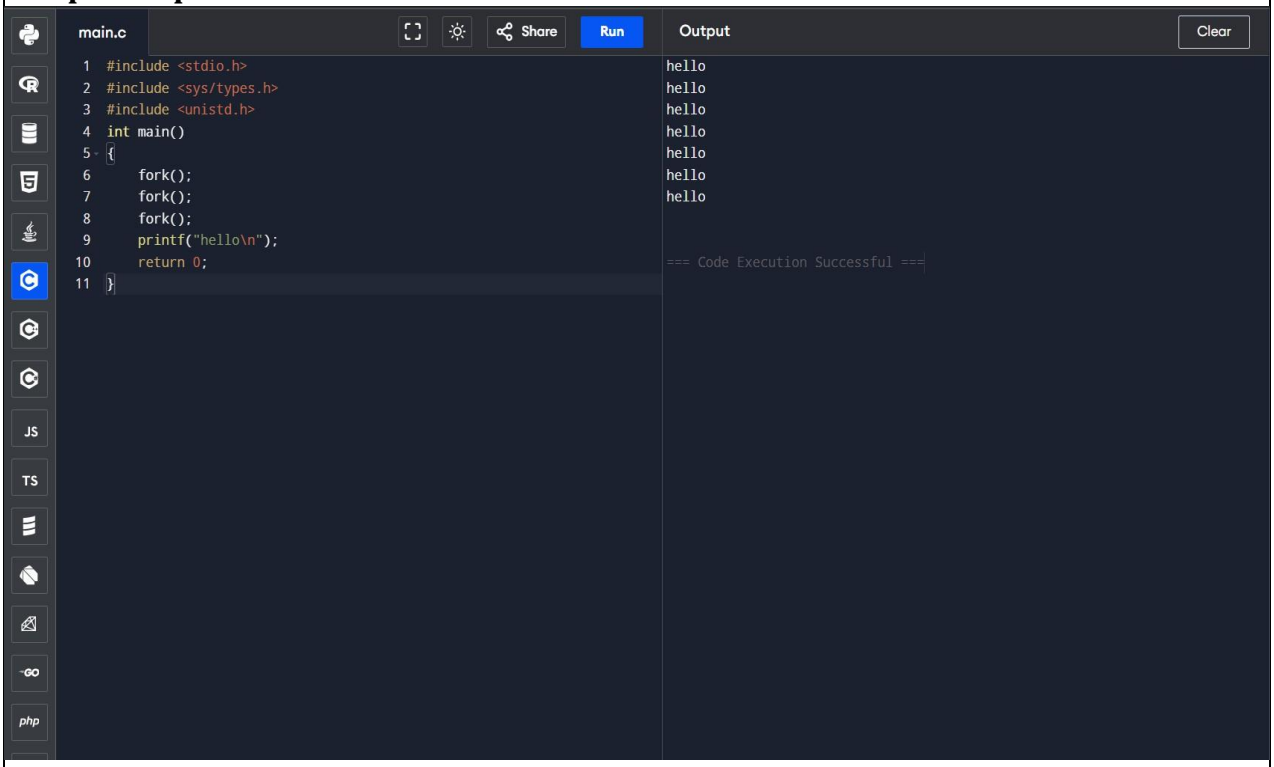
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
   pid_t p = fork();

   if (p == -1) {
      perror("fork failed");
      return 1;
   } else if (p == 0) {
      printf("Hello from child! My id is %d\n", getpid());
   } else {
      printf("Hello from parent! My id is %d\n", getpid());
   }

   return 0;
}
```

File Management 3:

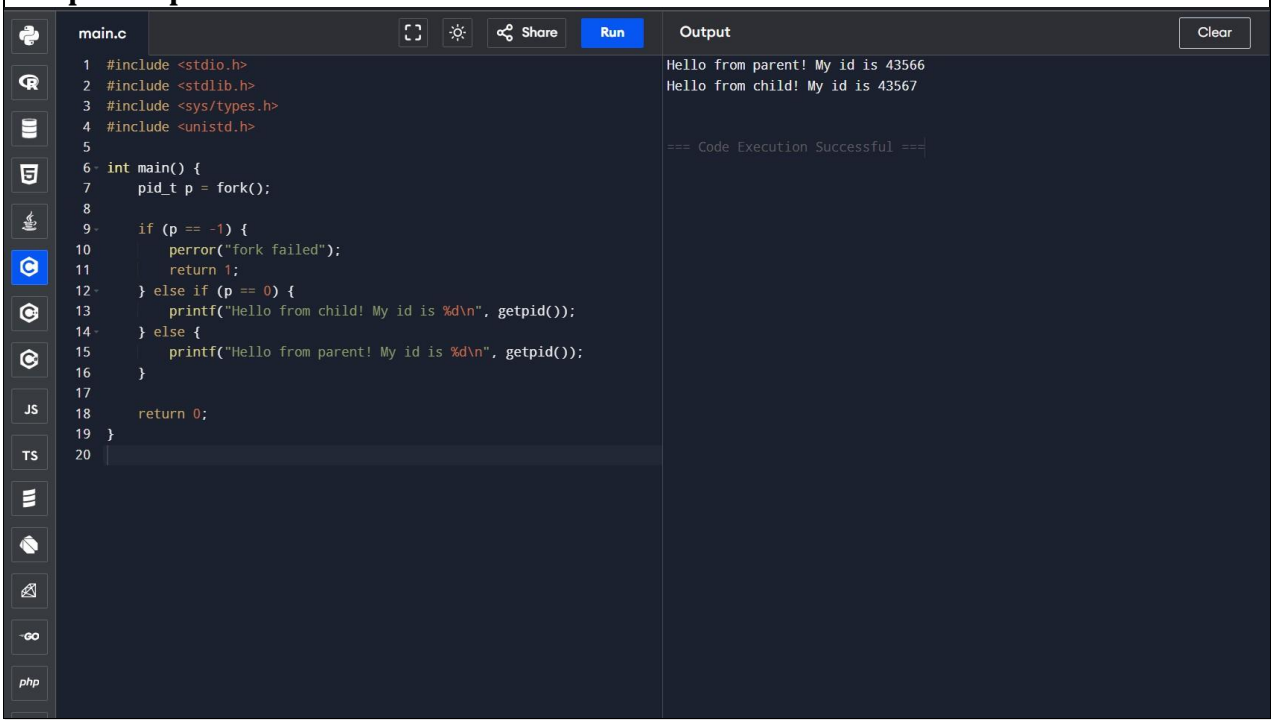
```
#include <fcntl.h>     // For open()
#include <unistd.h>    // For write() and close()
#include <string.h>    // For strlen()

int main() {
   int fd = open("example.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
   if (fd < 0) {
      // Error handling
      return 1;
   }

   write(fd, "Hello, file system!\n", strlen("Hello, file system!\n"));
   close(fd);
   return 0;
}
```

OUTPUT (SCREEN SHOT) IF ANY:

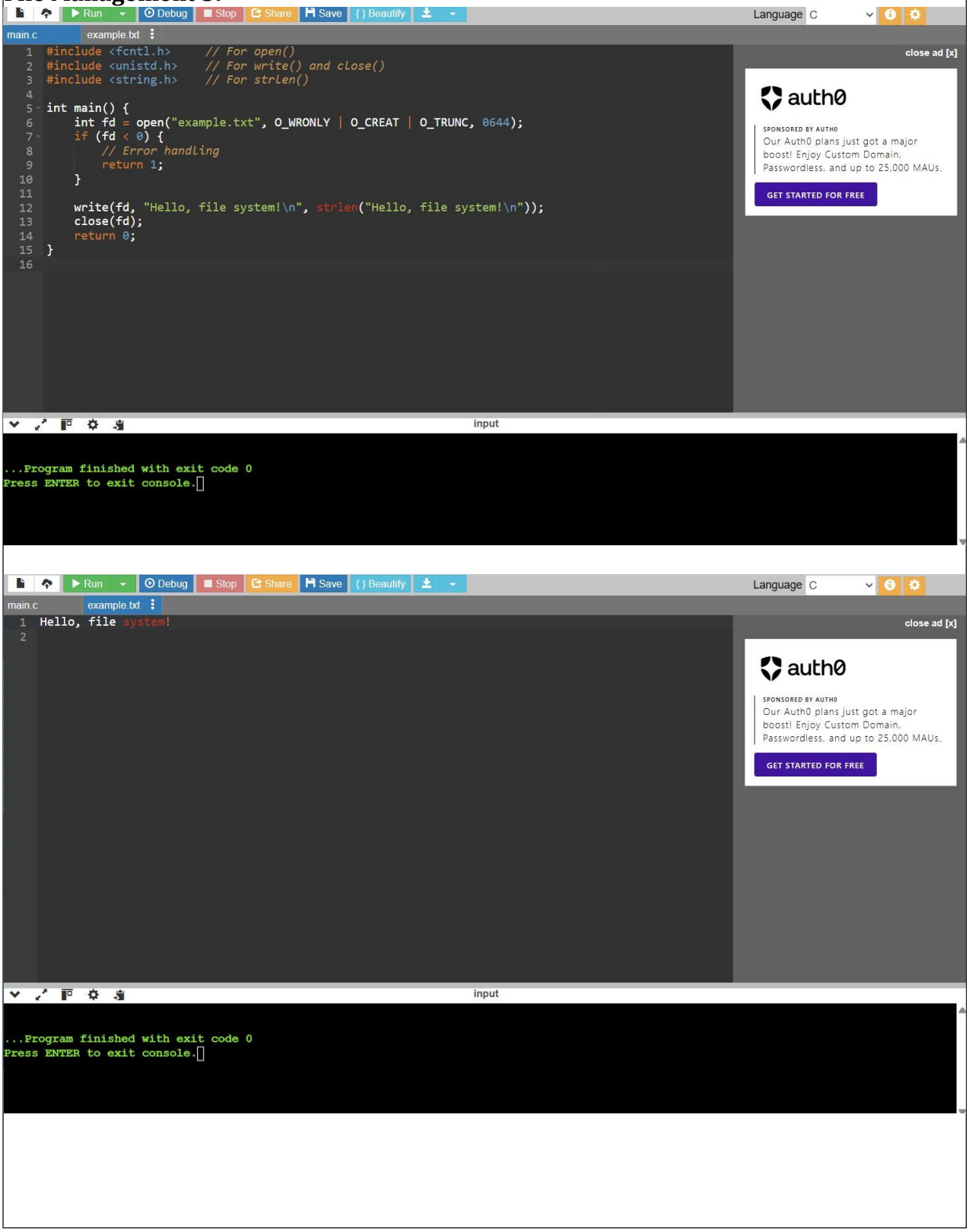
Sample Output 1:



Sample output 2:



File Management 3:



Conclusion:
By running these programs, we learned how the Linux kernel provides low-level control over processes, memory, and file operations through system calls**.**

Github Link- https://github.com/Puneet4382/Operating-System-Practicals