Agricultural Development Trust's

Shardabai Pawar Art,Commerce & Science College Shardanagar,

Malegaon (Bk), Tal-Baramati, Dist- Pune 413 115.

Department Of Computer Science

CERTIFICATE

Seat No:-

Date:-

This is to certify that, the work entered in this journal is the work of Mr. / Miss_____. Who has worked for the I / II / III / IV semester of the MSc (Comp.Sci) Year 2025- 2026 in the College. He / She has completed work as prescribed by Savitribai Phule Pune University, Pune.

Practical in-Charge **Head of Department**

Internal Examiner **External Examiner**

INDEX

Subject:-Lab course on CS-501-MJ(Advance Operating System) **Seat No:-**

Sr.No	Particular	Date	Sign & Remark
1	Assignment No 1		
2	Assignment No 2		
3	Assignment No 3		
4	Assignment No 4		
5	Assignment No 5		
6	Assignment No 6		
7	Assignment No 7		
8	Assignment No 8		
9	Assignment No 9		
10	Assignment No 10		

---------- Slip 1 ----------

Q.1) Take multiple files as Command Line Arguments and print their inode numbers and file types [10 Marks]

```
#include<stdio.h>

#include <sys/stat.h>

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

struct stat fileStat;

for (int i = 1; i < argc; i++) {

    if (stat(argv[i], &fileStat) == -1) {
perror("stat");

 continue;

       }

  printf("File: %s\n", argv[i]);

   printf("Inode: %lu\n", fileStat.st_ino);

  printf("Type: ");

  if (S_ISREG(fileStat.st_mode)) printf("Regular File\n");

  else if (S_ISDIR(fileStat.st_mode)) printf("Directory\n");

  else if (S_ISCHR(fileStat.st_mode)) printf("Character Device\n");
else if (S_ISBLK(fileStat.st_mode)) printf("Block Device\n");

 else if (S_ISFIFO(fileStat.st_mode)) printf("FIFO/Pipe\n");

 else if (S_ISLNK(fileStat.st_mode)) printf("Symbolic Link\n");
else if (S_ISSOCK(fileStat.st_mode)) printf("Socket\n");

   }return 0;

}
```

Command to run: - cc s1q1.c

./a.out file1.txt /etc /dev/null

Q.2) Write a C program to send SIGALRM signal by child process to parent process and parent process make a provision to catch the signal and display alarm is fired.(Use Kill, fork, signal and sleep system call) [20 Marks]

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
 void alarm_handler(int sig) {
            printf("Alarm is fired!\n");
}
 int main() {
     pid_t pid;
    signal(SIGALRM, alarm_handler);
    pid = fork();
 if (pid == 0) {         // Child
sleep(2);
 kill(getppid(), SIGALRM);
} else {         //Parent
pause();
     wait(NULL);
    }
    return 0;
}
```

Command to run: - cc s1q2.c

./a.out

---------- Slip 2 ----------

Q.1) Write a C program to find file properties such as inode number, number of hard link, File permissions, File size, File access and modification time and so on of a given file using stat() system call.

```
#include <stdio.h>

#include <sys/stat.h>

#include <time.h>


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

    struct stat fileStat;

if(argc != 2) {

   printf("Usage: %s <file_name>\n", argv[0]);

        return 1;

    }

    if(stat(argv[1], &fileStat) < 0)
{

  perror("stat");

     return 1;

    }

 printf("Information for %s\n", argv[1]);

 printf("---------------------------\n");

 printf("File inode number: %ld\n", (long)fileStat.st_ino);
printf("Number of hard links: %ld\n", (long)fileStat.st_nlink);

printf("File Permissions: ");

printf( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
printf( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
printf( (fileStat.st_mode & S_IWUSR) ? "w" : "-");
printf( (fileStat.st_mode & S_IXUSR) ? "x" : "-");
printf( (fileStat.st_mode & S_IRGRP) ? "r" : "-");
printf( (fileStat.st_mode & S_IWGRP) ? "w" : "-");
```

```
printf( (fileStat.st_mode & S_IXGRP) ? "x" : "-");
printf( (fileStat.st_mode & S_IROTH) ? "r" : "-");
printf( (fileStat.st_mode & S_IWOTH) ? "w" : "-");
printf( (fileStat.st_mode & S_IXOTH) ? "x\n" : "-
\n”)

 printf("File size: %ld bytes\n", (long)fileStat.st_size);

printf("Last access time: %s", ctime(&fileStat.st_atime));
printf("Last modification time: %s", ctime(&fileStat.st_mtime));
printf("Last status change time: %s", ctime(&fileStat.st_ctime));

return 0;

}
```

Command to run: - cc s2q1.c

./s2q1 file1.txt

Q.2) Write a C program that catches the ctrl-c (SIGINT) signal for the first time and display the appropriate message and exits on pressing ctrl-c again. [20 Marks]

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
volatile sig_atomic_t sigint_count = 0;
void handle_sigint(int sig) {
  sigint_count++;
   if(sigint_count == 1) {
    printf("\nCaught Ctrl-C for the first time! Press again to exit.\n");
    } else {
      printf("\nCaught Ctrl-C again! Exiting now.\n");
        exit(0);
    }
}
int main() {
    signal(SIGINT, handle_sigint);
printf("Press Ctrl-C to trigger the signal handler.\n");
 // Infinite loop to keep the program running and catch signals
    while(1) {
        // Do nothing, just wait for signal interrupts
    }
return 0;
}
```

Command to run: - cc s2q2.c

./a.out and press clt+C

---------- Slip 3 ----------

Q.1) Print the type of file and inode number where file name accepted through Command Line [10 Marks]

```
#include<stdio.h>

#include <sys/stat.h>

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

struct stat fileStat;

for (int i = 1; i < argc; i++) {

    if (stat(argv[i], &fileStat) == -1) {
perror("stat");

 continue;

       }

       printf("File: %s\n", argv[i]);

   printf("Inode: %lu\n", fileStat.st_ino);

  printf("Type: ");

  if (S_ISREG(fileStat.st_mode)) printf("Regular File\n");

  else if (S_ISDIR(fileStat.st_mode)) printf("Directory\n");

 else if (S_ISCHR(fileStat.st_mode)) printf("Character Device\n");
else if (S_ISBLK(fileStat.st_mode)) printf("Block Device\n");

 else if (S_ISFIFO(fileStat.st_mode)) printf("FIFO/Pipe\n");

 else if (S_ISLNK(fileStat.st_mode)) printf("Symbolic Link\n");
else if (S_ISSOCK(fileStat.st_mode)) printf("Socket\n");

   }

   return 0;

}
```

Command to run: - cc s1q1.c

./a.out file1.txt /etc /dev/null

Q.2) Write a C program which creates a child process to run linux/ unix command or any user defined program. The parent process set the signal handler for death of child signal and Alarm signal. If a child process does not complete its execution in 5 second then parent process kills child process. [20 Marks]

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

pid_t child_pid = -1;

void sigchld_handler(int sig) {

   int status;

   pid_t pid = waitpid(child_pid, &status, WNOHANG);

   if (pid == child_pid) {

      printf("Child process %d terminated\n", pid);
          child_pid = -1;  // child no longer running
          alarm(0);        // cancel alarm

   }

}

void sigalrm_handler(int sig) {

   if (child_pid > 0) {

   printf("Time limit exceeded. Killing child process %d\n", child_pid);

      kill(child_pid, SIGKILL);

   }
}

int main() {

// Register signal handlers

  signal(SIGCHLD, sigchld_handler);
```

```
   signal(SIGALRM, sigalrm_handler);
 child_pid = fork();
if (child_pid < 0) {
    perror("fork failed");
        exit(1);
    }
    if (child_pid == 0) {
        // Child process: Run a user-defined command here (e.g., "sleep 10")
            execlp("sleep", "sleep", "10", (char *) NULL);
        // If execlp fails
            perror("execlp failed");
        exit(1);
    } else {
        // Parent process: set an alarm for 5 seconds
        alarm(5);
        printf("Parent is waiting for child to finish or timeout\n");
        // Wait for signals (child termination or alarm)
            while (child_pid > 0) {
               pause(); // wait for signal
        }
        printf("Parent ends\n");
    }
    return 0;
}
```

---------- Slip 4 ----------

Q.1) Write a C program to find whether a given files passed through command line arguments are present in current directory or not. [10 Marks]

```
#include <stdio.h>
#include <sys/stat.h>
int main(int argc, char *argv[]) {
 struct stat fileStat;
 if (argc < 2) {
     printf("Usage: %s <file1> <file2> ...\n", argv[0]);
        return 1;
    }
    for (int i = 1; i < argc; i++) {
     if (stat(argv[i], &fileStat) == 0) {
       printf("File \"%s\" is present in the current directory.\n", argv[i]);
        } else {
            printf("File \"%s\" is NOT present in the current directory.\n", argv[i]);
        }
    }
    return 0;
}
```

Command to run: - cc s4q1.c

./s4q1 file1.txt file2.txt

Q.2) Write a C program which creates a child process and child process catches a signal SIGHUP, SIGINT and SIGQUIT. The Parent process send a SIGHUP or SIGINT signal after every 3 seconds, at the end of 15 second parent send SIGQUIT signal to child and child terminates by displaying message "My Papa has Killed me!!!". [20 Marks]

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

void handle_signal(int sig) {

if (sig == SIGHUP) {

     printf("Child received SIGHUP\n");

   } else if (sig == SIGINT) {
printf("Child received SIGINT\n");

   } else if (sig == SIGQUIT) {
printf("My Papa has Killed me!!!\n");

        exit(0);

   }

}

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

 if (pid < 0) {

    perror("fork failed");

        exit(1);

   }

 if (pid == 0) {     // Child process: set signal handlers

  signal(SIGHUP, handle_signal);

  signal(SIGINT, handle_signal);

   signal(SIGQUIT, handle_signal);
```

```
  // Infinite loop: waiting for signals
 while (1) {
  pause(); // Wait for signals
        }
    } else {
       // Parent process: send signals to child
int count = 0;
 while (count < 5) {
   sleep(3);
   if (count % 2 == 0) {
         kill(pid, SIGHUP);
printf("Parent sent SIGHUP\n");
} else {
kill(pid,SIGINT);
printf("Parent sent
SIGINT\n");
}
count++; // After 15
seconds, send SIGQUIT
to child
  kill(pid, SIGQUIT);
printf("Parent sent SIGQUIT\n");
 wait(NULL); // Wait for child to terminate
  printf("Parent exiting\n");
    }   return 0;
}
```

----------Slip 5 ----------

Q.1) Read the current directory and display the name of the files, no of files in current directory [10 Marks]

```
#include <stdio.h>

#include <dirent.h>

#include <string.h>

int main() {

DIR *d;

  struct dirent *dir;
int file_count = 0;

d = opendir(".");

    if (!d) {

        perror("opendir");

        return 1;

    }

printf("Files in current directory:\n");     while
((dir = readdir(d)) != NULL) {

// Ignore '.' and '..' entries

if (dir->d_type == DT_REG || dir->d_type == DT_DIR || dir->d_type == DT_LNK) {
if (strcmp(dir->d_name, ".") && strcmp(dir->d_name, "..")) {

       printf("%s\n", dir->d_name);

              file_count++;

           }

        }

    }

    closedir(d);

 printf("Total number of files: %d\n", file_count);

    return 0;

}
```

Q.2) Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it. Message1 = “Hello World” Message2 = “Hello SPPU” Message3 = “Linux is Funny” [20 Marks]

```
#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <stdint.h>   // for uint32_t

int main() {

  int fd[2];

   pid_t pid;

   char buffer[100];

char *msg1 = "Hello World";
char *msg2 = "Hello SPPU";
char *msg3 = "Linux is Funny";

if (pipe(fd) == -1) {
perror("pipe");

    return 1;

    }

pid = fork();     if
(pid < 0) {
perror("fork");
return 1;

    }
if (pid == 0) {

  // Child   process (writer)

  close(fd[0]); // Close read end

 char *messages[] = {msg1, msg2, msg3};

        for (int i = 0; i < 3; i++) {

uint32_t len = strlen(messages[i]) + 1; // include '\0'

    // Write length first
```

```
write(fd[1], &len, sizeof(len));
// Then write message
write(fd[1], messages[i], len);

        }

 close(fd[1]);

    } else {

        // Parent process (reader)
close(fd[1]); // Close write end

 for (int i = 0; i < 3; i++) {

      uint32_t len;             // First read the length

  read(fd[0], &len, sizeof(len));

       // Then read exactly 'len' bytes

    read(fd[0], buffer, len);

   printf("Received Message: %s\n", buffer);

        }

 close(fd[0]);

    }

return 0;

}
```

Command to run: - cc s5q2.c

./a.out

----------Slip 6----------

Q.1) Display all the files from current directory which are created in particular month [10 Marks]

```
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>
#include <string.h>

int main() {
   DIR *d;
   struct dirent *de;
   struct stat st; char
   month[10];
   printf("Enter month abbreviation (e.g., Jan, Feb, Mar): ");
   scanf("%s", month);
   d = opendir(".");
   if (d == NULL)
   {
      perror("opendir");
      return 1;
   }
   while ((de = readdir(d)) != NULL)
      { if (stat(de->d_name, &st) ==0)
      {
         struct tm *t = localtime(&st.st_mtime); // modification time
         char mon[10];
         strftime(mon, sizeof(mon), "%b", t); // format month (e.g., Jan)
         if (strcmp(mon, month) == 0) {
            printf("%s\n", de->d_name);
         }
```

```
        }
    }

    closedir(d);
    return 0;
}
```

Command to run: - cc s6q1.c
./a.out

Q.2) Write a C program to create n child processes. When all n child processes terminates, Display total cumulative time children spent in user and kernel mode [20 Marks]

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

int main() {
   int n, i;
   pid_t pid;
   struct rusage usage;
   struct timeval total_utime = {0, 0};  // total user time struct
   timeval total_stime = {0, 0};  // total system time

   printf("Enter number of child processes: ");
   scanf("%d", &n);

   for (i = 0; i < n; i++)
      { pid = fork();
      if (pid == 0) {
         // Child process: do some dummy work
         for (long j = 0; j < 10000000; j++); // consume CPU
         exit(0);
      } else if (pid < 0)
         {
         perror("fork");
         exit(1);
      }
   }
```

```
    // Parent waits for all children
    for (i = 0; i < n; i++) {
        wait4(-1, NULL, 0, &usage);

        // Accumulate user time
        total_utime.tv_sec  += usage.ru_utime.tv_sec;
        total_utime.tv_usec += usage.ru_utime.tv_usec;

        // Normalize microseconds
        if (total_utime.tv_usec >= 1000000) {
            total_utime.tv_sec++;
            total_utime.tv_usec -= 1000000;
        }

        // Accumulate system time  total_stime.tv_sec
        += usage.ru_stime.tv_sec;
        total_stime.tv_usec += usage.ru_stime.tv_usec;

        if (total_stime.tv_usec >= 1000000) {
            total_stime.tv_sec++;
            total_stime.tv_usec -= 1000000;
        }
    }

    printf("\nTotal cumulative user time   = %ld.%06ld seconds",
           (long) total_utime.tv_sec, (long) total_utime.tv_usec);
    printf("\nTotal cumulative system time = %ld.%06ld seconds\n",
           (long) total_stime.tv_sec, (long) total_stime.tv_usec);

    return 0;
}
```

----------Slip 7----------

Q.1) Write a C Program that demonstrates redirection of standard output to a file

[10 Marks]

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main() {
   int fd;
   // Open file for writing (create if not exists, truncate if exists)
   fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC,
   0644);

   if (fd < 0) {
      perror("open failed");
      return 1;
   }
   // Redirect stdout to file
   dup2(fd,
   STDOUT_FILENO);
   // Now printf will go to file instead of screen
   printf("This text goes into output.txt\n");
   printf("Redirection using dup2() successful!\n");
   close(fd);
   return 0;
}
```

Command to run: cc s7q1.c
./a.out
cat output.txt

Q.2) Implement the following unix/linux command (use fork, pipe and exec system call) ls –l | wc –l [20 Marks]

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

int main() {
   int pipefd[2];
   pid_t pid1, pid2;

   if (pipe(pipefd) == -1) {
      perror("pipe");
      exit(EXIT_FAILURE);
   }

   pid1 = fork();
   if (pid1 < 0) {
      perror("fork");
      exit(EXIT_FAILURE);
   }

   if (pid1 == 0) {
      // First child: execute `ls -l`
      close(pipefd[0]);         // Close read end, write output to pipe
      dup2(pipefd[1], STDOUT_FILENO);
      close(pipefd[1]);

      execlp("ls", "ls", "-l", (char
      *)NULL); perror("execlp ls");
      exit(EXIT_FAILURE);
```

```
    }
    pid2 = fork();
    if (pid2 < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    if (pid2 == 0) {
        // Second child: execute `wc -l`
        close(pipefd[1]);          // Close write end, read input from pipe
        dup2(pipefd[0], STDIN_FILENO);
        close(pipefd[0]);

        execlp("wc", "wc", "-l", (char
        *)NULL); perror("execlp wc");
        exit(EXIT_FAILURE);
    }

    // Parent process closes both ends and waits for children
    close(pipefd[0]);
    close(pipefd[1]);

    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return 0;
}
```

Command to run: cc s7q2.c

./a.out

----------Slip 8----------

Q.1)Write a C program that redirects standard output to a file output.txt. (use of dup and open system call). [10 Marks]

```
#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>


int main() {
   int fd;
   // Open file for writing (create if not exists, truncate if exists)
   fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC,
   0644);

    if (fd < 0) {
      perror("open failed");
      return 1;
   }
   // Redirect stdout to file
   dup2(fd,
   STDOUT_FILENO);
   // Now printf will go to file instead of screen
   printf("This text goes into output.txt\n");
   printf("Redirection using dup2() successful!\n");
   close(fd);
   return 0;
}
```

Command to run: cc s7q1.c
./a.out
cat output.txt

Q.2)Implement the following unix/linux command (use fork, pipe and exec system call) ls –l | wc –l. [20 Marks]

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

int main() {
int pipefd[2]; pid_t
pid1, pid2;

        if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);

}
pid1 = fork(); if
(pid1 < 0) {
perror("fork");
exit(EXIT_FAILURE);
}

        if (pid1 == 0) {
// First child: execute `ls -l`
close(pipefd[0]);  // Close read end, write output to pipe
dup2(pipefd[1], STDOUT_FILENO);
close(pipefd[1]);

execlp("ls", "ls", "-l", (char *)NULL);
perror("execlp ls");
exit(EXIT_FAILURE);
```

```
}

pid2 = fork(); if
(pid2 < 0) {
perror("fork");
exit(EXIT_FAILURE);
}
if (pid2 == 0) {
// Second child: execute `wc -l`
close(pipefd[1]);  // Close write end, read input from pipe
dup2(pipefd[0], STDIN_FILENO);
close(pipefd[0]);

execlp("wc", "wc", "-l", (char *)NULL);
perror("execlp wc");
exit(EXIT_FAILURE);

}

// Parent process closes both ends and waits for children
close(pipefd[0]);
close(pipefd[1]);

waitpid(pid1, NULL, 0);
waitpid(pid2, NULL, 0);

return 0;
}
```

Command to run: cc s7q2.c

./a.out

----------Slip 9----------

Q.1) Generate parent process to write unnamed pipe and will read from it [10 Marks]

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
   int fd[2];
   char write_msg[] = "Hello from parent";
   char read_msg[100];
   if (pipe(fd) == -1) {
      perror("pipe");
      return 1;
   }
   // Parent writes to pipe
   write(fd[1], write_msg, strlen(write_msg) + 1);
   // Now read from pipe
   read(fd[0], read_msg, sizeof(read_msg));
   printf("Parent read from pipe: %s\n", read_msg);
   //close both ends
   Close(fd[0]);
   Close(fd[1]);
   Return 0;
   }
```

Command to run:cc s9q1.c

./a.out

Q2.Write a C program to Identify the type (Directory, character device, Block device, Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call. [20 Marks]

```
#include <stdio.h>
#include <sys/stat.h>

int main(int argc, char *argv[]) {
   if (argc != 2) {
      printf("Usage: %s <file_name>\n", argv[0]);
      return 1;
   }

   struct stat fileStat;
   if (stat(argv[1], &fileStat) < 0) {
      perror("stat");
      return 1;
   }

   printf("File type of \"%s\": ", argv[1]);
   if (S_ISREG(fileStat.st_mode))
      printf("Regular File\n");
   else if (S_ISDIR(fileStat.st_mode))
```

```
        printf("Directory\n");
    else if (S_ISCHR(fileStat.st_mode))
        printf("Character Device\n");
    else if (S_ISBLK(fileStat.st_mode))
        printf("Block Device\n");
    else if (S_ISFIFO(fileStat.st_mode))
        printf("FIFO/Pipe\n");
    else if (S_ISLNK(fileStat.st_mode))
        printf("Symbolic Link\n");
    else if (S_ISSOCK(fileStat.st_mode))
        printf("Socket\n");
    else
        printf("Unknown Type\n");

    return 0;
}
```

Command to run : cc s9q2.c

./a.out output.txt

----------Slip 10----------

Q.1)Write a program that illustrates how to execute two commands concurrently with a pipe. [10 Marks]

```
#include   <stdio.h>
#include  <stdlib.h>
#include <unistd.h>

int main() {
   int fd[2];   // Pipe file descriptors
   pid_t pid;

   if (pipe(fd) == -1) {
      perror("pipe");
      exit(1);
   }

   pid = fork();
   if (pid < 0) {
      perror("fork");
      exit(1);
   }

   if (pid == 0) {
      // Child process → executes first command (ls -l)
      close(fd[0]);            // Close unused read end dup2(fd[1],
      STDOUT_FILENO); // Redirect stdout to pipe
      close(fd[1]);

      execlp("ls", "ls", "-l", NULL);
      perror("execlp ls");
      exit(1);
```

```
    } else {
        // Parent process → executes second command (wc -l)
        close(fd[1]);            // Close unused write end
        dup2(fd[0], STDIN_FILENO); // Redirect stdin from
        pipe close(fd[0]);

        execlp("wc", "wc", "-l", NULL);
        perror("execlp wc");
        exit(1);
    }

    return 0;
}
```

Command to run : cc s10q1.c

./a.out

Q2)Generate parent process to write unnamed pipe and will write into it. Also generate child process which will read from pipe [20 Marks]

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
   int  fd[2];
   pid_t pid;
   char msg[] = "Message from Parent to Child via Pipe";
   char buffer[100];
   if (pipe(fd) == -1) {
      perror("pipe");
      return 1;
   }
   pid = fork();
   if (pid < 0) {
      perror("fork");
      return 1;
   }
   if (pid > 0) {
      // Parent: write to pipe close(fd[0]);
      // Close reading end
      write(fd[1], msg, strlen(msg) + 1); // write message (+1 for '\0')
      close(fd[1]);
   } else {
      // Child: read from pipe
      close(fd[1]); // Close writing end
      read(fd[0], buffer, sizeof(buffer));
      printf("Child read from pipe: %s\n", buffer);
      close(fd[0]);
   }
   return 0;
}
```

Command to run : cc s10q2.c

./a.out