// assignment-1-1.c

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>


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


   int i;

   for(i = 0; i < argc; i++)

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

   exit(0);


}
```

//assignment-1-2.c

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<stdlib.h>

#include<sys/errno.h>


int main(){


   pid_t pid;

   char *args[] = {"elephant", "dog", "cat", "deer", NULL};
```

```
   printf("Before calling fork() system call\n");


   pid = fork();


   if(pid == 0){


      printf("Entered in child process. pid : %d ppid : %d\n", getpid(), getppid());

      execv("./assignment-1-1", args);

      perror("Failed to replace process image\n"); // this line gets executed means execv failed. no need to check seperately.

      _exit(errno); //inside child process, it's better to use _exit() coz it doesn't flush out i/o buffers whereas exit() does.


   }

   else if(pid == -1){


      perror("Failed to create child process\n");

      exit(errno);


   }

   else

   {

      sleep(1);

      printf("Entered in parent process. pid : %d ppid : %d child-process id : %d\n", getpid(), getppid(), pid);
```

```
    }

}
```

//assignment-1-3-child-1.c : file creation and taking user i/p s.

```
#include<stdio.h> // perror
#include <unistd.h> // read, write, close
#include <stdlib.h> // exit()
#include <fcntl.h> // mode consts
#include <signal.h> // SIGTSTP
#include <sys/errno.h> //errno

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

        int fd, msg;
        char read_byte;
        fd = open(argv[argc - 1], O_WRONLY | O_CREAT | O_EXCL, 0777);
        if (fd != -1)
        {

    write(STDOUT_FILENO, "Enter the content of the file:\n", 31);
```

```
                while (read(STDIN_FILENO, &read_byte, sizeof(read_byte)) > 0)
                {

                        if (read_byte == 'q')
                                break;

                        write(fd, &read_byte, sizeof(read_byte));

                }
        write(STDOUT_FILENO, "File Creation Done.\n", 20);
                close(fd);

        }
        else
        {

                write(STDOUT_FILENO, &errno, sizeof(errno));
                perror("File Open Error");
                exit(errno);

        }
        exit(1);
}

/*
```

to quit and save the content, press 'q' and then enter

*/

// assignment-1-3-child-2.c : file displaying

```
#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

#include<unistd.h>


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


        int fd;

        char read_byte;

        fd = open(argv[argc - 1], O_RDONLY);

        if(fd!= -1){


    write(STDOUT_FILENO, "Data reading starts...\n", 23);


                while(read(fd, &read_byte, sizeof(read_byte)) > 0){


                        write(STDOUT_FILENO, &read_byte, sizeof(read_byte));


                }
```

```
        write(STDOUT_FILENO, "\nData reading done.\n", 20);


            }
            else if(fd == -1)
                        fd = open(argv[1], O_CREAT, 0777);
            close(fd);
            exit(1);


}
```

// assignment-1-3.c : parent process goes here

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


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


   pid_t pid_1, pid_2;
   char *args[] = {argv[1], NULL};
   int status;
```

```
   pid_1 = fork();

   if(pid_1 == 0) /* child process 1 which creates file*/{

      printf("Entered in child process 1. pid : %d ppid : %d\n", getpid(), getppid());
      execv("./assignment-1-3-child-1", args);
      perror("Failed to replace child-1 process image\n");
      _exit(errno);

   }
   else if(pid_1 > 0) /* within parent process, creates child process 2*/
   {

      wait(&status);
      printf("Returned in parent process. pid : %d ppid : %d child-process 1 id : %d\n", getpid(), getppid(),
pid_1);

      pid_2 = fork();

      if(pid_2 == 0) /* child process 2 which displays file*/{

         printf("Entered in child process 2. pid : %d ppid : %d\n", getpid(), getppid());
         execv("./assignment-1-3-child-2", args);
         perror("Failed to replace child-2 process image\n");
         _exit(errno);
```

```
        }
        else if(pid_2 > 0){

            wait(&status);
            printf("Returned in parent process. pid : %d ppid : %d child-process id : %d\n", getpid(), getppid(),
pid_2);

        }
        else
        {
            perror("Fork Error in child 2 creation");
            exit(errno);
        }

    }
    else
    {
        perror("Fork Error in child 1 creation");
        exit(errno);
    }

}
```