11) Write a C program
a. to read first 20 characters from a file
b. seek to 10th byte from the beginning and display 20 characters from there
c. seek 10 bytes ahead from the current file offset and display 20 characters
d. display the file size

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<stdlib.h>
int main ()
{
  int file = 0, n;
  char buffer[25];
  if ((file = open ("testfil.txt", O_RDONLY)) < 0)
    {
      printf ("file open error\n");
      exit (0);
    }
  if (read (file, buffer, 20) != 20)
    printf ("file read operation failed\n");
  else
    write (STDOUT_FILENO, buffer, 20);
  printf ("\n");
  if (lseek (file, 10, SEEK_SET) < 0)
    printf ("lseek operation to beginning of file failed\n");
  if (read (file, buffer, 20) != 20)
    printf ("file read operation failed\n");
  else
    write (STDOUT_FILENO, buffer, 20);
  printf ("\n");
  if (lseek (file, 10, SEEK_CUR) < 0)
    printf ("lseek operation to beginning of file failed\n");
  if (read (file, buffer, 20) != 20)
    printf ("file read operation failed\n");
  else
    write (STDOUT_FILENO, buffer, 20);
  printf ("\n");
  if ((n = lseek (file, 0, SEEK_END)) < 0)
    printf ("lseek operation to end of file failed\n");
  printf ("size of file is %d bytes\n", n);
  close (file);
  return 0;
}
```

7. Write a C program to illustrate the effect of setjmp and longjmp functions on register and volatile variables.

```
#include <setjmp.h>
#include<stdio.h>
#include<stdlib.h>
static void f1 (int, int, int, int);
static void f2 (void);
static jmp_buf jmpbuffer;
static int globval;
int main (void)
{
  int autoval;
  register int regival;
  volatile int volaval;
  static int statval;
  globval = 1;
  autoval = 2;
  regival = 3;
  volaval = 4;
  statval = 5;
  if (setjmp (jmpbuffer) != 0)
    {
      printf ("after longjmp:\n");
      printf
             ("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n",
              globval, autoval, regival, volaval, statval);
      exit (0);
    }                                       /*
                                             * Change variables after setjmp, but before longjmp.
                                             */
  globval = 95;
  autoval = 96;
  regival = 97;
  volaval = 98;
  statval = 99;
  f1 (autoval, regival, volaval, statval);      /* never returns */
  exit (0);
}

static void f1 (int i, int j, int k, int l)
{
  printf ("in f1():\n");
  printf
    ("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n",
     globval, i, j, k, l);
  globval = 10000;
  j = 10000;
  f2 ();
}

static void f2 (void)
{
  longjmp (jmpbuffer, 1);
}
```

2. Write a C program which takes file descriptor as an argument and prints the description of selected file flags for that descriptor.

```
#include "apue.h"
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>
#include <string.h>
int main (int argc, char *argv[])
{
  int val;
  if (argc != 2)
    {
      printf ("usage: a.out <descriptor#>");
      exit (1);
    }
  if ((val = fcntl (atoi (argv[1]), F_GETFL, 0)) < 0)
    {
      printf ("fcntl error for fd %d", atoi (argv[1]));
      exit (1);
    }
  switch (val & O_ACCMODE)
    {
    case O_RDONLY:
      printf ("read only");
      break;
    case O_WRONLY:
      printf ("write only");
      break;
    case O_RDWR:
      printf ("read write");
      break;
    default:
      exit (1);
    }
  if (val & O_APPEND)
    printf (", append");
  if (val & O_NONBLOCK)
    printf (", nonblocking");
  if (val & O_SYNC)
    printf (", synchronous writes");
//putchar("\n");
  exit (0);
}
```

3. Write a C program to simulate system function.

```
#include<errno.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
int system1(const char *cmdstring)
{
	pid_t pid;
	int status;
	if(cmdstring == NULL)
		return(1);
	pid=fork();
	if(pid< 0)
		status = -1;
	else if(pid == 0)
	{
		execl("/bin/sh","sh","-c",cmdstring,(char *)0);
		_exit(127);
	}
	else
	{
		while(waitpid(pid,&status,0) < 0)
		{
			if(errno != EINTR)
			{
				status = -1;
				break;
			}
		}
	}
	return(status);
}
int main()
{
	int status;
	status = system1("ls -ls");
	printf("Command executed with status %d",status);
	status = system1("date");
	printf("Command executed with status %d",status);
	status = system1("sdfsd");
	printf("Command executed with status %d",status);
}
```

4. Write a C program to
a. To create a child process.
b. Child should execute an interpreter file by passing few arguments and some environment variables.
c. Parent should execute an interpreter file by passing few arguments
d. Create an interpreter file which has the path of echoall.c file
e. Create echoall.c file which prints the arguments and environment variables received through parent and child process

```
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
#include <sys/wait.h>
int main()
{
        pid_t pid;
        if((pid=fork())<0)
                printf("error");
        else if(pid==0)
if(execl("textinterpreter","test","myarg1","myarg2", "myarg4", (char *)0)<0)
        printf("error1");
if(waitpid(pid,NULL,0)<0)
        printf("wait error");
return 0;
}
```

<u>text interpreter</u> (textinterpreter)

```
#! /home/guest1/echoarg   my2
```

<u>echoarg.c file</u>

```
int main(int argc,char *argv[])
{     int i;
      for(i=0;i<argc;i++)
                printf("argv[%d]=%s",i,argv[i]);
      return(0);
}
```

```
cc echoarg.c -o echoarg
```

```
chmod 777 textinterpreter
```

```
gcc inter.c –o inter
./inter
```

1. Write a C program to create a child process and show how parent and child processes will share the text file and justify that both parent and child shares the same file offset.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<fcntl.h>
#include<sys/wait.h>
int main(void)
{
        pid_t pid;
        off_t offset;
        char buffer[32];
        int fd,status;
        if((fd = open("test2.txt",O_CREAT | O_RDWR)) == -1)
        {
                printf("Read error\n");
                exit(1);
        }
        write(fd,"Hi abc from msrit\n",18);
        pid = fork();
        if(pid == -1)
        {
                printf("Fork error\n");
                exit(1);
        }
        else if(pid == 0)
        {
                offset = lseek(fd,0,SEEK_CUR);
                printf("Child current offset \n%ld",offset);
                lseek(fd,0,SEEK_SET);
                read(fd,buffer,14);
                lseek(fd,5,SEEK_SET);
                printf("Child's current offset is \n%ld",lseek(fd,0,SEEK_CUR));
        }
        else
        {
                wait(&status);
                offset = lseek(fd,0,SEEK_CUR);
                printf("Parent current offset \n%ld",offset);
                lseek(fd,0,SEEK_SET);
                read(fd,buffer,14);
                lseek(fd,10,SEEK_SET);
                printf("Parent's current offset \n%ld",lseek(fd,0,SEEK_CUR));
        }
        return 0;
}
```

9. Write a C program to avoid zombie status of a process.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main (void)
{
  pid_t pid;
  if ((pid = fork ()) < 0)
    {
      printf ("forkerror");
    }
  else if (pid == 0)
    {                                   /* first child */
      if ((pid = fork ()) < 0)
          printf ("fork error");
      else if (pid > 0)
          exit (0);
      sleep (2);
      printf ("second child, parent pid = %ld\n", (long) getppid ());
      exit (0);
    }
  if (waitpid (pid, NULL, 0) != pid)
    printf ("waitpid error");
  exit (0);
}
```

5. Write a program to Copy access and modification time of a file to another file using utime function.

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

```
#include <fcntl.h>
#include <utime.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include<unistd.h>
int main (int argc, char *argv[])
{
  int i, fd;
  struct stat statbuf;
  struct utimbuf timebuf;
  for (i = 1; i < argc; i++)
    {
      if (stat (argv[i], &statbuf) < 0)
          {                         /* fetch current times */
            printf ("%s: stat error", argv[i]);
            continue;
          }
      if ((fd = open (argv[i], O_RDWR | O_TRUNC)) < 0)
          {                         /* truncate */
            printf ("%s: open error", argv[i]);
            continue;
          }
      close (fd);
      timebuf.actime = statbuf.st_atime;
      timebuf.modtime = statbuf.st_mtime;
      if (utime (argv[i], &timebuf) < 0)
          {                         /* reset times */
            printf ("%s: utime error", argv[i]);
            continue;
          }
    }
  exit (0);
}
```

8. Write a C program to perform the following operations
a. To create a child process
b. Child process should execute a program to show the use of the access function
c. Parent process should wait for the child process to exit
d. Also print the necessary process IDs

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

int main() {
    pid_t pid;

    // create a child process
    pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // child process executes a program to show the use of the access function
        if (access("test.txt", F_OK) == 0) {
            printf("test.txt exists and is readable\n");
        } else {
            printf("test.txt does not exist or is not readable\n");
        }
        exit(EXIT_SUCCESS);
    } else {
        // parent process waits for the child process to exit
        int status;
        waitpid(pid, &status, 0);
        printf("Parent process: Child process exited with status %d\n", status);
        printf("Parent process ID: %d\n", getpid());
        printf("Child process ID: %d\n", pid);
    }

    return 0;
}
```

10. Write a C program such that it initializes itself as a daemon Process

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

int main() {

    pid_t pid, sid; // process and session IDs
    // Fork off the parent process
    pid = fork();
    if (pid < 0) {
        exit(EXIT_FAILURE); // fork error
    }
    if (pid > 0) {
        exit(EXIT_SUCCESS); // parent process exit
    }

    // Change the file mode mask
    umask(0);

    // Open any logs here

    // Create a new SID for the child process
    sid = setsid();
    if (sid < 0) {
        exit(EXIT_FAILURE); // sid error
    }

    // Change the current working directory
    if ((chdir("/")) < 0) {
        exit(EXIT_FAILURE); // change dir error
    }

    // Close all open file descriptors
    int fd;
    for (fd = sysconf(_SC_OPEN_MAX); fd >= 0; fd--) {
        close (fd);
    }

    // Open stdin, stdout, stderr to /dev/null
    stdin = fopen("/dev/null", "r");
    stdout = fopen("/dev/null", "w");
    stderr = fopen("/dev/null", "w");

    // Now the child daemon process is ready to run
    while (1) {
        // Your daemon code here
        sleep(1); // to prevent the loop from consuming too much CPU time
    }

    exit(EXIT_SUCCESS);
}
```

12. Programs to demonstrate usage of umask and chmod functions

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

int main() {
  mode_t new_mask = 002; // sets the default permission to rw-rw-r--
  mode_t old_mask = umask(new_mask);

  // Create a new file with default permissions
  FILE *fp = fopen("test.txt", "w");
  if (fp == NULL) {
    perror("Error creating file");
    exit(EXIT_FAILURE);
  }
  fputs("This is a test", fp);
  fclose(fp);

  // Change the file permissions using chmod()
  if (chmod("test.txt", 0644) == -1) {
    perror("Error setting file permissions");
    exit(EXIT_FAILURE);
  }

  // Read the file with the changed permissions
  fp = fopen("test.txt", "r");
  if (fp == NULL) {
    perror("Error opening file");
    exit(EXIT_FAILURE);
  }
  char content[256];
  fgets(content, 256, fp);
  printf("%s", content);
  fclose(fp);

  return 0;
}
```

6. Consider the last 100 bytes as a region. Write a C program to check whether the region is locked or not. If the region is locked, print pid of the process which has locked. If the region is not locked, lock the region with an exclusive lock, read the last 50 bytes and unlock the region.

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
```

```
{
    int fd;
    char buffer[50];
    struct flock fl;

    fd = open("test.txt", O_RDWR);
    if (fd == -1) {
        perror("open");
        return 1;
    }

    // check if the last 100 bytes are already locked
    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_END;
    fl.l_start = -100;
    fl.l_len = 100;

    if (fcntl(fd, F_GETLK, &fl) == -1) {
        perror("fcntl");
        return 1;
    }

    if (fl.l_type == F_UNLCK) {
        // if not locked, lock the region with an exclusive lock
        fl.l_type = F_WRLCK;
        if (fcntl(fd, F_SETLK, &fl) == -1) {
            perror("fcntl");
            return 1;
        }

        // read the last 50 bytes
        lseek(fd, -50, SEEK_END);
        read(fd, buffer, 50);
        printf("%.*s\n", 50, buffer);

        // unlock the region
        fl.l_type = F_UNLCK;
        if (fcntl(fd, F_SETLK, &fl) == -1) {
            perror("fcntl");
            return 1;
        }
    } else {
        // if locked, print the PID of the process that's holding the lock
        printf("Region is locked by PID %d\n", fl.l_pid);
    }

    close(fd);
    return 0;
}
```