SET – A

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

```
#include <stdio.h>
#include <sys/stat.h>
int main(int argc, char *argv[]) {
   struct stat st;
   for (int i = 1; i < argc; i++) {
      if (stat(argv[i], &st) == 0) {
         printf("File: %s\n", argv[i]);
         printf("  Inode: %ld\n", st.st_ino);
         if (S_ISREG(st.st_mode))   printf("  Type: Regular file\n");
         else if (S_ISDIR(st.st_mode))  printf("  Type: Directory\n");
         else if (S_ISCHR(st.st_mode))  printf("  Type: Char device\n");
         else if (S_ISBLK(st.st_mode))  printf("  Type: Block device\n");
         else if (S_ISFIFO(st.st_mode)) printf("  Type: FIFO/pipe\n");
         else if (S_ISLNK(st.st_mode))  printf("  Type: Symbolic link\n");
         else if (S_ISSOCK(st.st_mode)) printf("  Type: Socket\n");
      } else perror("stat");
   }
   return 0;
}
```

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)

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
void handler(int sig) {
   printf("Alarm fired! (SIGALRM received)\n");
}
int main() {
   signal(SIGALRM, handler);
   pid_t pid = fork();
   if (pid == 0) {  // child
      sleep(2);
      kill(getppid(), SIGALRM);
```

```
    } else {        // parent
        pause(); // wait for signal
    }
    return 0;
}
```

SET - B

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[]) {
    if (argc < 2) { printf("Usage: %s <filename>\n", argv[0]); return 1; }
    struct stat st;
    if (stat(argv[1], &st) == 0) {
        printf("Inode: %ld\n", st.st_ino);
        printf("Hard Links: %ld\n", st.st_nlink);
        printf("Permissions: %o\n", st.st_mode & 0777);
        printf("File Size: %ld bytes\n", st.st_size);
        printf("Last Access: %s", ctime(&st.st_atime));
        printf("Last Modify: %s", ctime(&st.st_mtime));
    } else perror("stat");
    return 0;
}
```

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.

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
int count = 0;
void handler(int sig) {
    count++;
    if (count == 1) printf("Ctrl-C pressed once! Press again to exit.\n");
    else exit(0);
}
int main() {
    signal(SIGINT, handler);
```

```
    while (1); // infinite loop
    return 0;
}
```

SET – C

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

(SAME AS SET-A Q1)

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 itsexecution in 5 second then parent process kills child process.

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
pid_t child;
void handler(int sig) {
    printf("Child took too long! Killing it...\n");
    kill(child, SIGKILL);
}
int main() {
    child = fork();
    if (child == 0) {
        // Child executes "sleep 10"
        execlp("sleep", "sleep", "10", NULL);
    } else {
        signal(SIGALRM, handler);
        alarm(5);
        wait(NULL);
        printf("Parent: child finished or killed.\n");
    }
    return 0;
}
```

SET – D

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

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
```

```
    for (int i = 1; i < argc; i++) {
        if (access(argv[i], F_OK) == 0)
            printf("%s exists\n", argv[i]);
        else
            printf("%s not found\n", argv[i]);
    }
    return 0;
}
```

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!!!".

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
void sighup()  { printf("Child: got SIGHUP\n"); }
void sigint()  { printf("Child: got SIGINT\n"); }
void sigquit() { printf("My Papa has Killed me!!!\n"); _exit(0); }
int main() {
    pid_t pid = fork();
    if (pid == 0) {
        signal(SIGHUP, sighup);
        signal(SIGINT, sigint);
        signal(SIGQUIT, sigquit);
        while (1) pause();
    } else {
        sleep(3); kill(pid, SIGHUP);
        sleep(3); kill(pid, SIGINT);
        sleep(3); kill(pid, SIGHUP);
        sleep(3); kill(pid, SIGINT);
        sleep(3); kill(pid, SIGQUIT);
    }
    return 0;
}
```

SET – E

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

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

```
#include <dirent.h>
int main() {
    struct dirent *de;
    DIR *dr = opendir(".");
    if (!dr) return 1;
    int count = 0;
    while ((de = readdir(dr)) != NULL) {
        printf("%s\n", de->d_name);
        count++;
    }
    closedir(dr);
    printf("Total files: %d\n", 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”

```
#include <stdio.h>
#include <unistd.h>
int main() {
    int fd[2]; pipe(fd);
    if (fork() == 0) { // child
        close(fd[0]);
        write(fd[1], "Hello World\n", 12);
        write(fd[1], "Hello SPPU\n", 11);
        write(fd[1], "Linux is Funny\n", 15);
        close(fd[1]);
    } else { // parent
        char buf[100];
        close(fd[1]);
        int n = read(fd[0], buf, sizeof(buf)-1);
        buf[n] = '\0';
        printf("%s", buf);
        close(fd[0]);
```

```
    }
    return 0;
}
```

SET – F

Q.1) Display all the files from current directory which are created in particular Month

```
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>
#include <string.h>
int main() {
    DIR *d = opendir(".");
    struct dirent *de;
    struct stat st;
    int month;
    printf("Enter month (1-12): ");
    scanf("%d", &month);
    while ((de = readdir(d)) != NULL) {
        stat(de->d_name, &st);
        struct tm *t = localtime(&st.st_mtime);
        if (t->tm_mon + 1 == month)
            printf("%s\n", de->d_name);
    }
    closedir(d);
    return 0;
}
```

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

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/resource.h>
int main() {
    int n; printf("Enter n: "); scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        if (fork() == 0) { sleep(1); return 0; }
```

```
    }
    struct rusage usage;
    for (int i = 0; i < n; i++) wait(NULL);
    getrusage(RUSAGE_CHILDREN, &usage);
    printf("User time: %ld.%06ld sec\n", usage.ru_utime.tv_sec, usage.ru_utime.tv_usec);
    printf("System time: %ld.%06ld sec\n", usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);
    return 0;
}
```

SET – G

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

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main() {
    int fd = open("out.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
    dup2(fd, 1); // redirect stdout to file
    close(fd);
    printf("This goes into out.txt\n");
    return 0;
}
```

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

```
#include <stdio.h>
#include <unistd.h>
int main() {
    int fd[2]; pipe(fd);
    if (fork() == 0) { // child 1 -> ls -l
        dup2(fd[1], 1);
        close(fd[0]); close(fd[1]);
        execlp("ls", "ls", "-l", NULL);
    }
    if (fork() == 0) { // child 2 -> wc -l
        dup2(fd[0], 0);
        close(fd[1]); close(fd[0]);
        execlp("wc", "wc", "-l", NULL);
    }
    close(fd[0]); close(fd[1]);
```

```
    wait(NULL); wait(NULL);
    return 0;
}
```

SET – H

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

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

(SAME AS SET G Q1,Q2)BOTH

SET – I

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

```
#include <stdio.h>
#include <unistd.h>
int main() {
    int fd[2]; pipe(fd);
    if (fork() == 0) { // child
        close(fd[1]);
        char buf[50];
        int n = read(fd[0], buf, sizeof(buf));
        buf[n] = '\0';
        printf("Child got: %s\n", buf);
    } else { // parent
        close(fd[0]);
        write(fd[1], "Hello from Parent", 17);
        close(fd[1]);
    }
    return 0;
}
```

Q.2 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.

SAME AS SET-A Q1

SET – J

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

```
#include <stdio.h>
#include <unistd.h>
int main() {
    int fd[2]; pipe(fd);
    if (fork() == 0) { // child1 -> ls
```

```
        dup2(fd[1], 1);
        close(fd[0]); close(fd[1]);
        execlp("ls", "ls", NULL);
    }
    if (fork() == 0) { // child2 -> wc -c
        dup2(fd[0], 0);
        close(fd[1]); close(fd[0]);
        execlp("wc", "wc", "-c", NULL);
    }
    close(fd[0]); close(fd[1]);
    wait(NULL); wait(NULL);
    return 0;
}
```

Q.2) Generate parent process to write unnamed pipe and will write into it. Also generate child process which will read from pipe (SAME AS SET – I Q1)