Slip no .1

Q1) Take multiple files as Command Line Arguments and print their inode numbers and file types

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

const char* get_file_type(mode_t mode)
 {
   if (S_ISREG(mode))  return "Regular File";
   if (S_ISDIR(mode))  return "Directory";
   if (S_ISLNK(mode))  return "Symbolic Link";
   if (S_ISCHR(mode))  return "Character Device";
   if (S_ISBLK(mode))  return "Block Device";
   if (S_ISFIFO(mode)) return "FIFO (Named Pipe)";
   if (S_ISSOCK(mode)) return "Socket";
   return "Unknown";
}

int main(int argc, char *argv[])
{
   if (argc < 2) {
       printf("Usage: %s <file1> <file2> ...\n", argv[0]);
       return 1;
 }

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


 if (lstat(argv[i], &fileStat) == -1)
 {
          perror(argv[i]);
          continue;
  }
 printf("%s -> Inode: %lu, Type: %s\n",
             argv[i],
             (unsigned long)fileStat.st_ino,
             get_file_type(fileStat.st_mode));
    }

    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 <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>

void alarm_handler(int signum)
 {
    printf("Alarm is fired! Received signal %d\n", signum);
}

int main() {
    pid_t pid;
    signal(SIGALRM, alarm_handler);

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid == 0) {
        sleep(2)
        pid_t ppid = getppid();
        printf("Child sending SIGALRM to parent (PID: %d)\n", ppid);
        kill(ppid, SIGALRM);
        exit(0);
    } else {
       printf("Parent waiting for alarm...\n");
        pause();  // Wait for signal
        printf("Parent process exiting.\n");
    }

    return 0;
}
```

Slip no .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 <stdlib.h>
#include <sys/stat.h>
#include <time.h>
#include <unistd.h>
#include <stdio.h>
void print_permissions(mode_t mode)
 {
    printf("Permissions: ");
    printf( (S_ISDIR(mode)) ? "d" : "-");
    printf( (mode & S_IRUSR) ? "r" : "-");
    printf( (mode & S_IWUSR) ? "w" : "-");
    printf( (mode & S_IXUSR) ? "x" : "-");
    printf( (mode & S_IRGRP) ? "r" : "-");
    printf( (mode & S_IWGRP) ? "w" : "-");
    printf( (mode & S_IXGRP) ? "x" : "-");
    printf( (mode & S_IROTH) ? "r" : "-");
    printf( (mode & S_IWOTH) ? "w" : "-");
    printf( (mode & S_IXOTH) ? "x" : "-");
    printf("\n");
}
```

```
void print_file_type(mode_t mode)
 {
    printf("File Type: ");
    if (S_ISREG(mode)) printf("Regular File\n");
    else if (S_ISDIR(mode)) printf("Directory\n");
    else if (S_ISLNK(mode)) printf("Symbolic Link\n");
    else if (S_ISCHR(mode)) printf("Character Device\n");
    else if (S_ISBLK(mode)) printf("Block Device\n");
    else if (S_ISFIFO(mode)) printf("FIFO (Named Pipe)\n");
    else if (S_ISSOCK(mode)) printf("Socket\n");
    else printf("Unknown\n");
}

int main(int argc, char *argv[]) {
 if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }
    const char *filename = argv[1];
    struct stat fileStat;
    if (stat(filename, &fileStat) < 0)
 {
        perror("stat error");
        return 1;
    }
 printf("File: %s\n", filename);
    print_file_type(fileStat.st_mode);
    printf("Inode Number      : %lu\n", (unsigned long)fileStat.st_ino);
    printf("Hard Links        : %lu\n", (unsigned long)fileStat.st_nlink);
    print_permissions(fileStat.st_mode);
    printf("File Size         : %lld bytes\n", (long long)fileStat.st_size);
    printf("Last Access Time  : %s", ctime(&fileStat.st_atime));
    printf("Last Modify Time  : %s", ctime(&fileStat.st_mtime));
    printf("Last Status Change: %s", ctime(&fileStat.st_ctime));

    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 <stdlib.h>
#include <signal.h>
#include <unistd.h>

int sigint_count = 0;

void handle_sigint(int signum)
 {
   sigint_count++;
   if (sigint_count == 1) {
      printf("\nCaught SIGINT (Ctrl+C). Press again to exit.\n");
   } else {
      printf("\nCaught SIGINT again. Exiting now.\n");
      exit(0);
   }
}

int main() {
   signal(SIGINT, handle_sigint);

   printf("Program is running. Press Ctrl+C to trigger SIGINT.\n");
 while (1)
 {
      pause();  // Wait for signal
   }

   return 0;
}
```

Slip no .3

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

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

const char* get_file_type(mode_t mode)
 {
    if (S_ISREG(mode))  return "Regular File";
    if (S_ISDIR(mode))  return "Directory";
    if (S_ISLNK(mode))  return "Symbolic Link";
    if (S_ISCHR(mode))  return "Character Device";
    if (S_ISBLK(mode))  return "Block Device";
    if (S_ISFIFO(mode)) return "FIFO (Named Pipe)";
    if (S_ISSOCK(mode)) return "Socket";
    return "Unknown";
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <file1> <file2> ...\n", argv[0]);
        return 1;
    }

    for (int i = 1; i < argc; i++) {
        struct stat fileStat;
 if (lstat(argv[i], &fileStat) == -1) {
            perror(argv[i]);
            continue;
 }


        printf("%s -> Inode: %lu, Type: %s\n",
               argv[i],
               (unsigned long)fileStat.st_ino,
               get_file_type(fileStat.st_mode));
    }

    return 0;
}
```

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.

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

pid_t child_pid = -1;

void handle_sigchld(int signum) {
   int status;
   waitpid(child_pid, &status, 0);
   printf("Child process (PID: %d) terminated.\n", child_pid);
   exit(0);
}

void handle_sigalrm(int signum) {
   printf("Timeout! Child process (PID: %d) took too long. Killing it.\n",
child_pid);
   kill(child_pid, SIGKILL);
}

int main() {
   signal(SIGCHLD, handle_sigchld);
   signal(SIGALRM, handle_sigalrm);

   child_pid = fork();

   if (child_pid < 0) {
      perror("fork failed");
      exit(1);
   }
   if (child_pid == 0)
{
      execlp("sleep", "sleep", "10", NULL);
      perror("execlp failed");
      exit(1);
   }
else
{
      printf("Parent: Started child with PID %d\n", child_pid);
      alarm(5);
while (1)
     {
         pause();
      }
}

   return 0;
}
```

Slip no.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.

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

int main(int argc, char *argv[])
 {
   if (argc < 2) {
      printf("Usage: %s <file1> <file2> ...\n", argv[0]);
      return 1;
   }

   for (int i = 1; i < argc; i++) {
      if (access(argv[i], F_OK) == 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;
}
```

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 <stdlib.h>
#include <unistd.h>
#include <signal.h>

void handle_sighup(int sig)
 {
   printf("Child: Received SIGHUP signal\n");
}
void handle_sigint(int sig)
 {
   printf("Child: Received SIGINT signal\n");
}

void handle_sigquit(int sig)
 {
   printf("Child: Received SIGQUIT signal\n");
   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)
 {
       signal(SIGHUP, handle_sighup);
        signal(SIGINT, handle_sigint);
        signal(SIGQUIT, handle_sigquit);
         while (1)
 {
            pause();  // Wait for signals
  }
    }
 else {

        sleep(1);

        for (int i = 1; i <= 5; i++) {
            if (i % 2 == 1) {
                printf("Parent: Sending SIGHUP to child\n");
                kill(pid, SIGHUP);
            } else {
                printf("Parent: Sending SIGINT to child\n");
                kill(pid, SIGINT);
            }
            sleep(3);
        }
       printf("Parent: Sending SIGQUIT to child\n");
        kill(pid, SIGQUIT);
        wait(NULL);
    }

    return 0;
}
```

Slip no. 5

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

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

int main()
{
   struct dirent *entry;
   DIR *dir;
   int file_count = 0;
   dir = opendir(".");

   if (dir == NULL) {
      perror("Unable to open current directory");
      return 1;
   }

   printf("Files and directories in the current directory:\n");

   while ((entry = readdir(dir)) != NULL) {
      if (entry->d_name[0] == '.' &&
        (entry->d_name[1] == '\0' || (entry->d_name[1] == '.' && entry-
>d_name[2] == '\0')))
         continue;

      printf("%s\n", entry->d_name);
      file_count++;
   }

   closedir(dir);

   printf("\nTotal number of files/directories: %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”

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

int main() {
   int pipefd[2];
   pid_t pid;

   char *messages[] =
 {
      "Message1 = Hello World\n",
      "Message2 = Hello SPPU\n",
      "Message3 = Linux is Funny\n"
   };

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

   pid = fork();

   if (pid < 0) {
      perror("Fork failed");
      exit(1);
   }

   if (pid == 0) {

      close(pipefd[0]);
```

```
    for (int i = 0; i < 3; i++) {
      write(pipefd[1], messages[i], strlen(messages[i]));
    }

    close(pipefd[1]);
  }
else
 {
    close(pipefd[1]);
    char buffer[100];
    int n;

    printf("Parent: Reading messages from the pipe...\n");
   while ((n = read(pipefd[0], buffer, sizeof(buffer)-1)) > 0)
{
      buffer[n] = '\0';
      printf("%s", buffer);
}

    close(pipefd[0]);
  }

  return 0;
}
```

Slip no.6

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

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

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

```
{
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <month_number (1-12)>\n", argv[0]);
        return 1;
    }

    int month = atoi(argv[1]);
    if (month < 1 || month > 12) {
        fprintf(stderr, "Invalid month number. Please enter 1 to 12.\n");
        return 1;
    }

    DIR *dir = opendir(".");
    if (!dir) {
        perror("Failed to open current directory");
        return 1;
    }

    struct dirent *entry;
    struct stat fileStat;
    char timeBuf[80];

    printf("Files modified in month %d:\n", month);

    while ((entry = readdir(dir)) != NULL) {
        // Skip '.' and '..'
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..")
== 0)
            continue;

        if (stat(entry->d_name, &fileStat) == -1)
{
            perror("stat failed");
            continue;
        }
struct tm *mod_time = localtime(&fileStat.st_mtime);
        if (mod_time->tm_mon + 1 == month) { // tm_mon is 0-11, so +1 for
1-12
            printf("%s\n", entry->d_name);
        }
    }

    closedir(dir);
    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 <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <time.h>

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

    int n = atoi(argv[1]);
    if (n <= 0) {
        fprintf(stderr, "Number of children must be positive.\n");
        return 1;
    }

    pid_t pid;
    int i;

    for (i = 0; i < n; i++) {
        pid = fork();
        if (pid < 0) {
            perror("fork failed");
            exit(1);
        }
        if (pid == 0) {

            srand(getpid());
            int sleep_time = 1 + rand() % 3;
            sleep(sleep_time);
            exit(0);
        }
    }

    for (i = 0; i < n; i++) {
        wait(NULL);
    }
    struct rusage usage;
    if (getrusage(RUSAGE_CHILDREN, &usage) == -1) {
        perror("getrusage failed");
        return 1;
    }

    printf("Total user CPU time of all children: %ld.%06ld seconds\n",
           usage.ru_utime.tv_sec, usage.ru_utime.tv_usec);

    printf("Total system CPU time of all children: %ld.%06ld seconds\n",
           usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);

    return 0;
}
```

Slip no.7

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

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

```
int main() {

    int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd < 0)
 {
        perror("Failed to open file");
        exit(1);
    }
 if (dup2(fd, STDOUT_FILENO) < 0) {
        perror("dup2 failed");
        close(fd);
        exit(1);
    }

    close(fd);
  printf("This output is redirected to the file 'output.txt'.\n");
    printf("Hello, this is a demonstration of stdout redirection!\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 <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

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

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

    pid1 = fork();
    if (pid1 < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }
    if (pid1 == 0) {
        close(pipefd[0]);
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[1]);

        execlp("ls", "ls", "-l", (char *)NULL);
        perror("execlp ls failed");
        exit(EXIT_FAILURE);
    }
    pid2 = fork();
    if (pid2 < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }
    if (pid2 == 0)
 {
        close(pipefd[1]);
        dup2(pipefd[0], STDIN_FILENO);
        close(pipefd[0]);

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

    close(pipefd[0]);
    close(pipefd[1]);
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return 0;
}
```

Slip no.8

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

```
 #include <stdio.h>
 #include <fcntl.h>
 #include <unistd.h>
int main(){
    int fd;
    fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd < 0) {
       perror("open");
       return 1;
    }
    if (dup2(fd, STDOUT_FILENO) < 0) {
       perror("dup2");
       close(fd);
       return 1;
    }
  close(fd);
    printf("This output will be written to output.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 <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) {
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[0]);
        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) {
        dup2(pipefd[0], STDIN_FILENO);
        close(pipefd[1]);
        close(pipefd[0]);

        execlp("wc", "wc", "-l", (char *)NULL);
        perror("execlp wc");
        exit(EXIT_FAILURE);
    }
    close(pipefd[0]);
    close(pipefd[1]);
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return 0;
}
```

Slip no. 9

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

```
#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;
   }

   write(fd[1], write_msg, strlen(write_msg));

   close(fd[1]); // Close write end so we can read cleanl
   int bytes = read(fd[0], read_msg, sizeof(read_msg) - 1);
   if (bytes >= 0) {
      read_msg[bytes] = '\0';
      printf("Parent read: %s\n", read_msg);
   } else {
      perror("read");
   }

   close(fd[0]);
   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

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

int main(int argc, char *argv[]) {
    struct stat fileStat;
  if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    if (stat(argv[1], &fileStat) == -1) {
        perror("stat");
        exit(EXIT_FAILURE);
    }
    if (S_ISREG(fileStat.st_mode))
        printf("File type: Regular file\n");
    else if (S_ISDIR(fileStat.st_mode))
        printf("File type: Directory\n");
    else if (S_ISCHR(fileStat.st_mode))
        printf("File type: Character device\n");
    else if (S_ISBLK(fileStat.st_mode))
        printf("File type: Block device\n");
    else if (S_ISFIFO(fileStat.st_mode))
        printf("File type: FIFO (named pipe)\n");
    else if (S_ISLNK(fileStat.st_mode))
        printf("File type: Symbolic link\n");
    else if (S_ISSOCK(fileStat.st_mode))
        printf("File type: Socket\n");
    else
        printf("File type: Unknown\n");

    return 0;
}
```

Slip no. 10

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

```
#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 == -1) {
      perror("fork");
      exit(EXIT_FAILURE);
   }

   if (pid1 == 0) {
      dup2(pipefd[1], STDOUT_FILENO);
      close(pipefd[0]);
      close(pipefd[1]);
      execlp("ls", "ls", "-l", (char *)NULL);
      perror("execlp command1");
      exit(EXIT_FAILURE);
   }

   pid2 = fork();
   if (pid2 == -1) {
      perror("fork");
      exit(EXIT_FAILURE);
   }

   if (pid2 == 0) {
        dup2(pipefd[0], STDIN_FILENO);
        close(pipefd[1]);
        close(pipefd[0])

        execlp("grep", "grep", ".c", (char *)NULL);
        perror("execlp command2");
        exit(EXIT_FAILURE);
    }
    close(pipefd[0]);
    close(pipefd[1]);

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

    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

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

int main() {
    int fd[2]; // fd[0]: read end, fd[1]: write end
    pid_t pid;
    char message[] = "Hello from parent via pipe!";
    char buffer[100];

    // Create the pipe
    if (pipe(fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Fork to create child process
    pid = fork();

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

    // Child process: read from pipe
    if (pid == 0) {
        close(fd[1]); // Close unused write end

        int bytesRead = read(fd[0], buffer, sizeof(buffer) - 1);
        if (bytesRead >= 0) {
            buffer[bytesRead] = '\0'; // Null-terminate string
            printf("Child received: %s\n", buffer);
        } else {
            perror("read");
        }

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

    // Parent process: write to pipe
    else {
        close(fd[0]); // Close unused read end

        write(fd[1], message, strlen(message));
        close(fd[1]); // Close write end after writing

        wait(NULL); // Wait for child to finish
    }

    return 0;
}
```

Slip no.11

Q.1) Write a C program to get and set the resource limits such as files, memory associated with a Process

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

void print_limit(int resource, const char *name)
 {
   struct rlimit limit;

   if (getrlimit(resource, &limit) == -1)
 {
      perror("getrlimit");
      return;
   }

   printf("%s:\n", name);
   printf("  Soft limit: %ld\n", (long)limit.rlim_cur);
   printf("  Hard limit: %ld\n\n", (long)limit.rlim_max);
}

void set_limit(int resource, rlim_t new_soft_limit)
{
   struct rlimit limit;

   if (getrlimit(resource, &limit) == -1)
 {
      perror("getrlimit");
      return;
   }

   limit.rlim_cur = new_soft_limit;

   if (setrlimit(resource, &limit) == -1)
{
      perror("setrlimit");
   } else
 {
      printf("Successfully updated soft limit.\n");
   }
}

int main() {
   printf("=== Original Resource Limits ===\n\n");

   print_limit(RLIMIT_FSIZE, "Maximum file size (bytes)");
   print_limit(RLIMIT_AS, "Maximum virtual memory size (bytes)");
   print_limit(RLIMIT_NOFILE, "Maximum number of open files");

   printf("=== Setting New Soft Limit for RLIMIT_NOFILE to 1024 ===\n");
   set_limit(RLIMIT_NOFILE, 1024);
   printf("\n=== Updated Resource Limits ===\n\n");
   print_limit(RLIMIT_NOFILE, "Maximum number of open files");

   return 0;
}
```

Q.2) Write a C program that redirects standard output to a file output.txt. (use of dup and open system call).

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

int main()
{
   int fd;
  fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
   if (fd < 0)
 {
      perror("open");
      exit(EXIT_FAILURE);
   }

   if (dup2(fd, STDOUT_FILENO) < 0)
{
      perror("dup2");
      close(fd);
      exit(EXIT_FAILURE);
   }
   printf("This will be written to output.txt instead of the terminal.\n");
   printf("Standard output successfully redirected using dup2().\n");

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

Slip no.12

Q.1) Write a C program that print the exit status of a terminated child process

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

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

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

     printf("Child process running...\n");
     sleep(2);
     exit(42);
   } else
{
      printf("Parent waiting for child to terminate...\n");
      wait(&status);
      if (WIFEXITED(status))
 {
         int exit_status = WEXITSTATUS(status);
         printf("Child terminated normally with exit status: %d\n",
exit_status);
      } else if (WIFSIGNALED(status))
 {
         printf("Child terminated by signal: %d\n", WTERMSIG(status));
      }
else
 {
         printf("Child terminated abnormally.\n");
      }
   }

   return 0;
}
```

Q.2) Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes. I) (e.g $ a.out a.txt b.txt c.txt, …)

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

struct FileInfo
 {
   char name[256];
   off_t size;
};

int compare(const void *a, const void *b)
 {
   struct FileInfo *fileA = (struct FileInfo *)a;
   struct FileInfo *fileB = (struct FileInfo *)b;

   if (fileA->size < fileB->size) return -1;
   else if (fileA->size > fileB->size) return 1;
   else return 0;
}

int main(int argc, char *argv[]) {
   if (argc < 2) {
       printf("Usage: %s <file1> <file2> ...\n", argv[0]);
       return 1;
   }

   int file_count = argc - 1;
   struct FileInfo files[file_count];

   for (int i = 1; i < argc; i++) {
       struct stat st;
       if (stat(argv[i], &st) == -1) {
           perror(argv[i]);
           continue;
       }
      strncpy(files[i - 1].name, argv[i], sizeof(files[i - 1].name) - 1);
       files[i - 1].name[sizeof(files[i - 1].name) - 1] = '\0';
       files[i - 1].size = st.st_size;
   }
   qsort(files, file_count, sizeof(struct FileInfo), compare);
  printf("Files in ascending order of size:\n");
   for (int i = 0; i < file_count; i++) {
       printf("%s (%ld bytes)\n", files[i].name, files[i].size);
   }

   return 0;
}
```

Slip no.13

Q.1) Write a C program that illustrates suspending and resuming processes using signals

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

int main()
 {
   pid_t pid;

   pid = fork();

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

   if (pid == 0)
 {
      int i = 0;
      while (1) {
         printf("Child process running... %d\n", i++);
         sleep(1);
      }
   }
else
{
      printf("Parent: Child PID is %d\n", pid);
      sleep(5);

      printf("Parent: Sending SIGSTOP to child (suspend)...\n");
      kill(pid, SIGSTOP);
      sleep(5);
      printf("Parent: Sending SIGCONT to child (resume)...\n");
      kill(pid, SIGCONT);
```

```
        sleep(5);

        printf("Parent: Sending SIGTERM to child (terminate)...\n");
        kill(pid, SIGTERM);
        wait(NULL);
        printf("Parent: Child terminated.\n");
    }

    return 0;
}
```

Q.2) Write a C program that a string as an argument and return all the files that begins with that name in the current directory. For example > ./a.out foo will return all file names that begins with foo

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

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

    const char *prefix = argv[1];
    struct dirent *entry;
    DIR *dir;
    dir = opendir(".");
    if (dir == NULL)
 {
        perror("opendir");
        return 1;
    }

    printf("Files starting with \"%s\":\n", prefix);
```

```
   while ((entry = readdir(dir)) != NULL)
 {
      if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..")
== 0)
         continue;
      if (strncmp(entry->d_name, prefix, strlen(prefix)) == 0)
 {

         struct stat st;
         if (stat(entry->d_name, &st) == 0 && S_ISREG(st.st_mode))
 {
            printf("%s\n", entry->d_name);
         }
      }
   }

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

Slip no.14

Q.1) Display all the files from current directory whose size is greater that n Bytes Where n is accept from user.

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

int main()
 {
   long n;
   char *filename;
   DIR *dir;
   struct dirent *entry;
   struct stat st;
```

```
    printf("Enter size in bytes (n): ");
    if (scanf("%ld", &n) != 1 || n < 0)
  {
        fprintf(stderr, "Invalid input.\n");
        return 1;
    }

    dir = opendir(".");
    if (dir == NULL)
  {
        perror("opendir");
        return 1;
    }

    printf("Files larger than %ld bytes:\n", n);
    while ((entry = readdir(dir)) != NULL)
  {
        filename = entry->d_name;

        if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0)
            continue;

        if (stat(filename, &st) == 0) {
            if (S_ISREG(st.st_mode) && st.st_size > n)
            {
                printf("%s (%ld bytes)\n", filename, st.st_size);
            }
            else
  {
            perror(filename);
  }
  }

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

Q.2) 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 <stdlib.h>
#include <sys/stat.h>
#include <time.h>
#include <unistd.h>
#include <pwd.h>
#include <grp.h>

void print_permissions(mode_t mode) {
   printf( (S_ISDIR(mode)) ? "d" : "-");
   printf( (mode & S_IRUSR) ? "r" : "-");
   printf( (mode & S_IWUSR) ? "w" : "-");
   printf( (mode & S_IXUSR) ? "x" : "-");
   printf( (mode & S_IRGRP) ? "r" : "-");
   printf( (mode & S_IWGRP) ? "w" : "-");
   printf( (mode & S_IXGRP) ? "x" : "-");
   printf( (mode & S_IROTH) ? "r" : "-");
   printf( (mode & S_IWOTH) ? "w" : "-");
   printf( (mode & S_IXOTH) ? "x" : "-");
}

int main(int argc, char *argv[]) {
   struct stat fileStat;

   if (argc != 2) {
      fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
      return 1;
   }

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

   printf("File: %s\n", argv[1]);
   printf("Inode Number: %ld\n", (long)fileStat.st_ino);
   printf("Number of Hard Links: %ld\n", (long)fileStat.st_nlink);
   printf("Owner UID: %d (%s)\n", fileStat.st_uid,
getpwuid(fileStat.st_uid)->pw_name);
   printf("Group GID: %d (%s)\n", fileStat.st_gid, getgrgid(fileStat.st_gid)-
>gr_name);
   printf("File Size: %ld bytes\n", (long)fileStat.st_size);

   printf("Permissions: ");
   print_permissions(fileStat.st_mode);
   printf("\n");

   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;
}
```

Slip no.15

Q.1) Display all the files from current directory whose size is greater that n Bytes Where n is accept from user

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

int main()
 {
   long n;
   struct dirent *entry;
   struct stat fileStat;
   DIR *dir;
   printf("Enter size in bytes (n): ");
   if (scanf("%ld", &n) != 1 || n < 0) {
      fprintf(stderr, "Invalid input. Please enter a positive number.\n");
      return 1;
```

```
    }

    dir = opendir(".");
    if (dir == NULL) {
        perror("opendir");
        return 1;
    }

    printf("Files larger than %ld bytes:\n", n);
    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..")
== 0)
            continue;
        if (stat(entry->d_name, &fileStat) == 0)
{
            if (S_ISREG(fileStat.st_mode) && fileStat.st_size > n) {
                printf("%s (%ld bytes)\n", entry->d_name, fileStat.st_size);
            }
        }
    }

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

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

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

pid_t child_pid = -1;
void handle_sigchld(int sig) {
    int status;
    pid_t pid = waitpid(child_pid, &status, WNOHANG);
    if (pid > 0) {
        printf("Child process %d terminated.\n", pid);
        alarm(0);
    }
}
void handle_sigalrm(int sig) {
    if (child_pid > 0) {
        printf("Child process %d exceeded time limit. Killing it...\n", child_pid);
        kill(child_pid, SIGKILL);
    }
}
int main() {

    signal(SIGCHLD, handle_sigchld);
    signal(SIGALRM, handle_sigalrm);

    child_pid = fork();

    if (child_pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }

    if (child_pid == 0)

        printf("Child process (PID: %d) started.\n", getpid());
        execlp("sleep", "sleep", "10", NULL);
        perror("execlp failed");
        exit(EXIT_FAILURE);
    } else {
        printf("Parent process (PID: %d) waiting for child...\n", getpid());
        alarm(5);  // Set 5-second timer

        while (1) {
            pause();
        }
    }

    return 0;
}
```

Slip no. 16

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

```
#define _XOPEN_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>

int main() {
    int month;
    printf("Enter month number (1-12): ");
    if (scanf("%d", &month) != 1 || month < 1 || month > 12) {
        fprintf(stderr, "Invalid month.\n");
        return 1;
    }

    DIR *dir = opendir(".");
    if (!dir) {
        perror("opendir");
        return 1;
    }

    struct dirent *entry;
    struct stat fileStat;
    struct tm *tm_info;

    printf("\nFiles modified/created in month %d:\n", month);
    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &fileStat) == 0) {
            tm_info = localtime(&fileStat.st_ctime);
            if (tm_info->tm_mon + 1 == month) {
                printf("File: %-20s  Time: %s", entry->d_name,
ctime(&fileStat.st_ctime));
            }
        }
    }

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

**Q.2) Write a C program which create a child process which catch a signal sighup, sigint and sigquit. The Parent process send a sighup or sigint signal after every 3 seconds, at the end of 30 second parent send sigquit signal to child and child terminates my displaying message “My DADDY has
Killed me!!!**

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

void signal_handler(int sig) {
    switch(sig) {
        case SIGHUP:
            printf("Child: Received SIGHUP\n");
            break;
        case SIGINT:
            printf("Child: Received SIGINT\n");
            break;
        case SIGQUIT:
            printf("My DADDY has Killed me!!!\n");
            exit(0);
    }
}

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

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid == 0) {
        signal(SIGHUP, signal_handler);
        signal(SIGINT, signal_handler);
        signal(SIGQUIT, signal_handler);

        while(1)
            pause();
    } else {
        for (int i = 1; i <= 10; i++) {
            sleep(3);
            if (i % 2 == 0)
                kill(pid, SIGINT);
            else
                kill(pid, SIGHUP);
        }

        kill(pid, SIGQUIT);
         wait(NULL);
        printf("Parent: Child has terminated.\n");
    }

    return 0;
}
```

Slip no.17

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>
#include <sys/stat.h>
#include <string.h>

int main() {
    DIR *dir;
    struct dirent *entry;
    struct stat fileStat;
    int file_count = 0;
    dir = opendir(".");
    if (dir == NULL) {
        perror("Unable to open current directory");
        return 1;
    }

    printf("Files in current directory:\n");

    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &fileStat) == 0)
            if (S_ISREG(fileStat.st_mode)) {
                printf("%s\n", entry->d_name);
                file_count++;
            }
        }
    }

    closedir(dir);
    printf("\nTotal number of files: %d\n", file_count);

    return 0;
}
```

Q.2) Write a C program which create a child process which catch a signal sighup, sigint and sigquit. The Parent process send a sighup or sigint signal after every 3 seconds, at the end of 30 second parent send sigquit signal to child and child terminates my displaying message “My DADDY has Killed me!!!”.

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

void sighup_handler(int signum) {
    printf("Child received SIGHUP\n");
}
```

```
void sigint_handler(int signum) {
    printf("Child received SIGINT\n");
}

void sigquit_handler(int signum) {
    printf("My DADDY has Killed me!!!\n");
    exit(0);
}

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

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid == 0) {
        signal(SIGHUP, sighup_handler);
        signal(SIGINT, sigint_handler);
        signal(SIGQUIT, sigquit_handler);

        while (1)
            pause();
    } else {
        for (int i = 0; i < 10; i++) {
            if (i % 2 == 0)
                kill(pid, SIGHUP);
            else
                kill(pid, SIGINT);
            sleep(3);
        }

        kill(pid, SIGQUIT);
        wait(NULL);
    }

    return 0;
}
```

Slip no.18

Q.1) Write a C program to find whether a given file is present in current directory or not

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

int main() {
    char filename[100];
    DIR *dir;
    struct dirent *entry;
    int found = 0;

    printf("Enter the file name to search: ");
    scanf("%s", filename);

    dir = opendir(".");
    if (dir == NULL) {
        perror("Unable to open directory");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, filename) == 0) {
            found = 1;
            break;
        }
    }

    closedir(dir);

    if (found)
        printf("File '%s' found in current directory.\n", filename);
    else
        printf("File '%s' not found in current directory.\n", filename);

    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>
#include <string.h>

int main() {
    int fd[2];
    pid_t pid;
    char buffer[100];

    pipe(fd);
    pid = fork();

    if (pid == 0) {
        close(fd[0]);
        write(fd[1], "Hello World\n", strlen("Hello World\n"));
        write(fd[1], "Hello SPPU\n", strlen("Hello SPPU\n"));
        write(fd[1], "Linux is Funny\n", strlen("Linux is Funny\n"));
        close(fd[1]);
    } else {
        close(fd[1]);
        read(fd[0], buffer, sizeof(buffer));
        printf("Parent received:\n%s", buffer);
        close(fd[0]);
    }

    return 0;
}
```

Slip no.19

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

```
#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(argv[i]);
         continue;
      }
 printf("File: %s\n", argv[i]);
      printf("Inode: %lu\n", fileStat.st_ino);

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

      printf("\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>
#include <stdlib.h>

int main() {
   int fd[2];
   pid_t pid;

   pipe(fd);
   pid = fork();

   if (pid == 0) {
      dup2(fd[1], STDOUT_FILENO);
      close(fd[0]);
      close(fd[1]);
      execlp("ls", "ls", "-l", NULL);
      exit(1);
   } else {
      dup2(fd[0], STDIN_FILENO);
      close(fd[1]);
      close(fd[0]);
      execlp("wc", "wc", "-l", NULL);
      exit(1);
   }

   return 0;
}
```

Slip no.20

Q.1) Write a C program that illustrates suspending and resuming processes using signals

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

```
#include <unistd.h>
#include <signal.h>

void handle_sigusr1(int sig) {
   printf("Process resumed\n");
}

void handle_sigusr2(int sig) {
   printf("Process suspended\n");
   pause();
}

int main() {
   signal(SIGUSR1, handle_sigusr1);
   signal(SIGUSR2, handle_sigusr2);

   while (1) {
      printf("Running...\n");
      sleep(2);
   }

   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.

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

int main(int argc, char *argv[]) {
   struct stat fileStat;

   if (argc < 2) {
      printf("Usage: %s <filename>\n", argv[0]);
      return 1;
   }

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

    if (S_ISREG(fileStat.st_mode))
        printf("Type: Regular File\n");
    else if (S_ISDIR(fileStat.st_mode))
        printf("Type: Directory\n");
    else if (S_ISCHR(fileStat.st_mode))
        printf("Type: Character Device\n");
    else if (S_ISBLK(fileStat.st_mode))
        printf("Type: Block Device\n");
    else if (S_ISFIFO(fileStat.st_mode))
        printf("Type: FIFO or Pipe\n");
    else if (S_ISLNK(fileStat.st_mode))
        printf("Type: Symbolic Link\n");
    else if (S_ISSOCK(fileStat.st_mode))
        printf("Type: Socket\n");
    else
        printf("Type: Unknown\n");

    return 0;
}
```

Slip no.21

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>
#include <sys/stat.h>
#include <string.h>

int main() {
    DIR *dir;
    struct dirent *entry;
    struct stat fileStat;
    int count = 0;
    dir = opendir(".");
    if (dir == NULL) {
        perror("opendir");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &fileStat) == 0) {
            if (S_ISREG(fileStat.st_mode)) {
                printf("%s\n", entry->d_name);
                count++;
            }
        }
    }

    closedir(dir);
    printf("Total number of files: %d\n", count);
    return 0;
}
```

Q.2) Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes. I) (e.g $ a.out a.txt b.txt c.txt, …)

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

struct FileInfo {
    char name[256];
    off_t size;
};

int compare(const void *a, const void *b) {
    struct FileInfo *f1 = (struct FileInfo *)a;
    struct FileInfo *f2 = (struct FileInfo *)b;
    return (f1->size - f2->size);
}
```

```
int main(int argc, char *argv[]) {
   struct FileInfo files[100];
   struct stat fileStat;
   int count = 0;

   for (int i = 1; i < argc; i++) {
      if (stat(argv[i], &fileStat) == 0) {
         strcpy(files[count].name, argv[i]);
         files[count].size = fileStat.st_size;
         count++;
      }
   }

   qsort(files, count, sizeof(struct FileInfo), compare);

   for (int i = 0; i < count; i++) {
      printf("%s (%ld bytes)\n", files[i].name, files[i].size);
   }

   return 0;
}
```

Slip no.22

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

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
int main() {
   int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
   dup2(fd, STDOUT_FILENO);
   close(fd);
  printf("This output is redirected to output.txt\n");
 return 0;
}
```

Q.2) Write a C program to implement the following unix/linux command (use fork, pipe and exec system call). Your program should block the signal Ctrl-C and Ctrl-\ signal during the execution.
i. ls –l | wc –l

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

int main() {
    int fd[2];
    pid_t pid1, pid2;
    sigset_t set;

    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGQUIT);
    sigprocmask(SIG_BLOCK, &set, NULL);

    pipe(fd);

    pid1 = fork();
    if (pid1 == 0) {
        dup2(fd[1], STDOUT_FILENO);
        close(fd[0]);
        close(fd[1]);
        execlp("ls", "ls", "-l", NULL);
        exit(1);
    }

    pid2 = fork();
    if (pid2 == 0) {
        dup2(fd[0], STDIN_FILENO);
        close(fd[1]);
        close(fd[0]);
        execlp("wc", "wc", "-l", NULL);
        exit(1);
    }
    close(fd[0]);
    close(fd[1]);
    wait(NULL);
    wait(NULL);

    return 0;
}
```

Slip no.23

Q.1) Write a C program to find whether a given file is present in current directory or not

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

int main() {
    char filename[100];
    DIR *dir;
    struct dirent *entry;
    int found = 0;

    printf("Enter the filename to search: ");
    scanf("%s", filename);

    dir = opendir(".");
    if (dir == NULL) {
        perror("opendir");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, filename) == 0) {
            found = 1;
            break;
        }
    }
    closedir(dir);

    if (found)
        printf("File '%s' found in current directory.\n", filename);
    else
        printf("File '%s' not found in current directory.\n", filename);

    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.

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

int main(int argc, char *argv[]) {
    struct stat fileStat;

    if (argc < 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

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

    if (S_ISREG(fileStat.st_mode))
        printf("Type: Regular File\n");
    else if (S_ISDIR(fileStat.st_mode))
        printf("Type: Directory\n");
    else if (S_ISCHR(fileStat.st_mode))
        printf("Type: Character Device\n");
    else if (S_ISBLK(fileStat.st_mode))
        printf("Type: Block Device\n");
    else if (S_ISFIFO(fileStat.st_mode))
```

```
        printf("Type: FIFO or Pipe\n");
    else if (S_ISLNK(fileStat.st_mode))
        printf("Type: Symbolic Link\n");
    else if (S_ISSOCK(fileStat.st_mode))
        printf("Type: Socket\n");
    else
        printf("Type: Unknown\n");

    return 0;
}
```

Slip no .24

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

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

int main(int argc, char *argv[]) {
    struct stat fileStat;

    if (argc < 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

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

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

    if (S_ISREG(fileStat.st_mode))
        printf("Type: Regular File\n");
    else if (S_ISDIR(fileStat.st_mode))
        printf("Type: Directory\n");
    else if (S_ISCHR(fileStat.st_mode))
        printf("Type: Character Device\n");
    else if (S_ISBLK(fileStat.st_mode))
        printf("Type: Block Device\n");
    else if (S_ISFIFO(fileStat.st_mode))
        printf("Type: FIFO or Pipe\n");
    else if (S_ISLNK(fileStat.st_mode))
        printf("Type: Symbolic Link\n");
    else if (S_ISSOCK(fileStat.st_mode))
        printf("Type: Socket\n");
    else
        printf("Type: Unknown\n");

    return 0;
}
```

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

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

pid_t child_pid;

void handle_sigchld(int sig) {
    wait(NULL);
    printf("Child process terminated.\n");
    exit(0);
}

void handle_sigalrm(int sig) {
    printf("Child process took too long. Killing it...\n");
    kill(child_pid, SIGKILL);
}
```

```
int main() {
   signal(SIGCHLD, handle_sigchld);
   signal(SIGALRM, handle_sigalrm);

   child_pid = fork();

   if (child_pid == 0) {
      execlp("sleep", "sleep", "10", NULL);
      exit(1);
   } else {
      alarm(5);
      while (1)
         pause();
   }

   return 0;
}
```

Slip no.25

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

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

int main() {
   int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
   dup2(fd, STDOUT_FILENO);
   close(fd);

   printf("This output is redirected to output.txt\n");

   return 0;
}
```

Q.2) Write a C program that redirects standard output to a file output.txt. (use of dup and open system call).

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

int main() {
   int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
   dup2(fd, STDOUT_FILENO);
   close(fd);

   printf("Redirected output to output.txt\n");

   return 0;
}
```