SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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>
#include <stdlib.h>
int main(int argc, char *argv[]) {
  struct stat fileStat;
  int i;
  if (argc < 2) {
    printf("Usage: %s <file1> <file2> ...\n", argv[0]);
    exit(1);
  }
  for (i = 1; i < argc; i++) {
    if (stat(argv[i], &fileStat) == -1) {
      perror(argv[i]);
      continue;
    }
    printf("File: %s\n", argv[i]);
    printf("Inode Number: %ld\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;
}
```

OUTPUT:-

```
File: file_info.c
Inode Number: 4062587
Type: Regular File
---------------------------------
File: /etc
Inode Number: 2
Type: Directory
---------------------------------
File: passwd
Inode Number: 123456
```

Type: Regular File

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>
void handle_alarm(int sig) {
   printf("Alarm is fired! (Caught signal %d)\n", sig);
}
int main() {
   pid_t pid;
   signal(SIGALRM, handle_alarm);  // parent sets up signal handler
   pid = fork();
   if (pid < 0) {
      perror("Fork failed");
      return 1;
   } else if (pid == 0) {
      // Child process
      sleep(2); // give parent time to set handler
      printf("Child sending SIGALRM to parent...\n");
      kill(getppid(), SIGALRM);
   } else {
      // Parent process
```

```
    printf("Parent waiting for alarm...\n");
    pause(); // wait for signal
    printf("Parent received alarm and continues...\n");
  }
  return 0;
}
```

OUTPUT:-

```
Parent waiting for alarm...
Child sending SIGALRM to parent...
Alarm is fired! (Caught signal 14)
Parent received alarm and continues...
```

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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. [10 Marks]

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

void print_permissions(mode_t mode) {
   printf("File 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");
}
```

```
int main(int argc, char *argv[]) {
   if(argc != 2){
      printf("Usage: %s <filename>\n", argv[0]);
      return 1;
   }
   struct stat fileStat;
   if(stat(argv[1], &fileStat) < 0){
      perror("stat");
      return 1;
   }

   printf("File: %s\n", argv[1]);
   printf("Inode number: %lu\n", (unsigned long)fileStat.st_ino);
   printf("Number of hard links: %lu\n", (unsigned long)fileStat.st_nlink);
   print_permissions(fileStat.st_mode);
   printf("File size: %ld bytes\n", (long)fileStat.st_size);

   struct passwd *pw = getpwuid(fileStat.st_uid);
   struct group  *gr = getgrgid(fileStat.st_gid);
   printf("Owner UID: %u (%s)\n", fileStat.st_uid, pw ? pw->pw_name : "Unknown");
   printf("Group GID: %u (%s)\n", fileStat.st_gid, gr ? gr->gr_name : "Unknown");

   printf("Last access  : %s", ctime(&fileStat.st_atime));
   printf("Last modified: %s", ctime(&fileStat.st_mtime));
   printf("Last status change: %s", ctime(&fileStat.st_ctime));
   return 0;
}
```

OutPut:

File: test.txt

Inode number: 2345678

Number of hard links: 1

File permissions: -rw-r--r--

File size: 42 bytes

Owner UID: 1000 (yourusername)

Group GID: 1000 (yourgroup)

Last access : Sat Nov 8 17:05:22 2025

Last modified: Sat Nov 8 16:59:12 2025

Last status change: Sat Nov 8 16:59:12 2025

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 SIGINT (Ctrl-C) for the first time. Press Ctrl-C again to exit.\n");
```

```
    } else {
        printf("\nSecond SIGINT received. Exiting...\n");
        exit(0);
    }
}

int main() {
    struct sigaction sa;
    sa.sa_handler = handle_sigint;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGINT, &sa, NULL);

    printf("Press Ctrl-C. The first time displays a message. The second time exits.\n");
    while(1) {
        pause();  // Wait for signals
    }
    return 0;
}
```

OutPut:

Press Ctrl-C. The first time displays a message. The second time exits.

Caught SIGINT (Ctrl-C) for the first time. Press Ctrl-C again to exit.

Second SIGINT received. Exiting...

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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>

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

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) {
        perror("stat");
        return 1;
    }
    printf("File: %s\n", argv[1]);
    printf("File type: %s\n", file_type(st.st_mode));
    printf("Inode number: %lu\n", (unsigned long)st.st_ino);
    return 0;
}
```

OutPut :

File: test.txt

File type: Regular File

Inode number: 2345678

If you run with a directory instead (e.g., /tmp):

File: /tmp

File type: Directory

Inode number: 1024

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; // Track child PID

void sigchld_handler(int sig) {
   printf("Child process terminated (SIGCHLD received).\n");
   // Optionally reap the child process to avoid zombie
   waitpid(child_pid, NULL, WNOHANG);
}

void sigalrm_handler(int sig) {
   printf("Alarm Fired: Child process exceeded time limit. Killing child...\n");
   if (child_pid > 0) {
      kill(child_pid, SIGKILL);
   }
}
int main(int argc, char *argv[]) {
   if (argc < 2) {
```

```
    printf("Usage: %s <command> [arg1 ... argN]\n", argv[0]);
    exit(1);
}
// Set up signal handlers for SIGCHLD (child exit) and SIGALRM (timeout)
signal(SIGCHLD, sigchld_handler);
signal(SIGALRM, sigalrm_handler);

child_pid = fork();

if (child_pid < 0) {
    perror("fork");
    exit(1);
}
if (child_pid == 0) {
    // Child executes the given program/command
    execvp(argv[1], &argv[1]);
    // If execvp fails
    perror("execvp");
    exit(1);
} else {
    // Parent process
    alarm(5); // Set a 5 seconds timer
    int status;
    pid_t ret;
    ret = waitpid(child_pid, &status, 0); // Wait for child to exit
    alarm(0); // Cancel alarm if child exited in time
```

```
    if (ret == child_pid) {
        printf("Child exited normally.\n");
    }
  }
  return 0;
}
```

OutPut:

./a.out sleep 10

Alarm Fired: Child process exceeded time limit. Killing child...

Child process terminated (SIGCHLD received).

Child process terminated (SIGCHLD received).

Child exited normally.

./a.out sleep 2

Child process terminated (SIGCHLD received).

Child exited normally.

Usage: ./a.out <command> [arg1 ... argN]

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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[]) {
  if (argc < 2) {
    printf("Usage: %s <file1> [file2 ... fileN]\n", argv[0]);
    return 1;
  }
  for (int i = 1; i < argc; ++i) {
    struct stat st;
    if (stat(argv[i], &st) == 0) {
      printf("File '%s' is present in current directory.\n", argv[i]);
    } else {
      printf("File '%s' is NOT present in current directory.\n", argv[i]);
    }
  }
  return 0;
}
```

OUTPUT:

./a.out notes.txt main.c missing.txt report.pdf

File 'notes.txt' is present in current directory.

File 'main.c' is present in current directory.

File 'missing.txt' is NOT present in current directory.

File 'report.pdf' is present in current directory.

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>

volatile sig_atomic_t running = 1;

// Signal handlers for the child
void handle_sighup(int sig) {
   printf("Child: Received SIGHUP\n");
}
void handle_sigint(int sig) {
   printf("Child: Received SIGINT\n");
}
void handle_sigquit(int sig) {
```

```
    printf("My Papa has Killed me!!!\n");
    running = 0;
}

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

    if (pid < 0) {
        perror("fork");
        exit(1);
    }
    if (pid == 0) {
        // Child process: install signal handlers
        signal(SIGHUP, handle_sighup);
        signal(SIGINT, handle_sigint);
        signal(SIGQUIT, handle_sigquit);

        while (running) {
            pause(); // Wait for signals
        }
        exit(0);
    } else {
        // Parent process
        int elapsed = 0;
        while (elapsed < 15) {
            sleep(3);
            elapsed += 3;
            if (elapsed < 15) {
```

```
            // Alternate between SIGHUP and SIGINT
            if (elapsed % 6 == 0)
               kill(pid, SIGHUP);
            else
               kill(pid, SIGINT);
          }
        }
        // After 15 seconds, send SIGQUIT
        kill(pid, SIGQUIT);
        wait(NULL); // Optionally wait for child to exit
    }
    return 0;
}
```

OUTPUT:

Child: Received SIGINT

Child: Received SIGHUP

Child: Received SIGINT

Child: Received SIGHUP

Child: Received SIGINT

My Papa has Killed me!!!

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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>

int main() {
  DIR *dir;
  struct dirent *entry;
  int count = 0;

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

  printf("Files in current directory:\n");
  while ((entry = readdir(dir)) != NULL) {
    // Skip "." and ".."
    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);
        count++;
    }
    closedir(dir);

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

OUTPUT:

Files in current directory:

main.c

test.txt

data.csv

output.txt

Total number of files: 4

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

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

   if (pipe(fd) == -1) {
      perror("pipe");
      exit(1);
   }
   pid = fork();
   if (pid < 0) {
      perror("fork");
      exit(1);
   }
   if (pid == 0) {
      // Child process -- writer
      close(fd[0]); // Close read-end
      char *messages[] = {
         "Hello World",
         "Hello SPPU",
         "Linux is Funny"
      };
      for(int i = 0; i < 3; i++) {
         write(fd[1], messages[i], strlen(messages[i]) + 1); // +1 for '\0'
      }
      close(fd[1]);
      exit(0);
```

```
    } else {
        // Parent process -- reader
        close(fd[1]); // Close write-end
        printf("Messages from child:\n");
        for(int i = 0; i < 3; i++) {
            read(fd[0], buffer, sizeof(buffer));
            printf("%s\n", buffer);
        }
        close(fd[0]);
    }
    return 0;
}
```

OUTPUT:

Messages from child:

Hello World

Hello SPPU

Linux is Funny

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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 <string.h>
#include <time.h>

int main() {
   DIR *dir;
   struct dirent *entry;
   struct stat st;
   char month_name[10];
   int target_month, current_year;
   printf("Enter month number (1-12): ");
   scanf("%d", &target_month);

   // Get current year
   time_t t = time(NULL);
   struct tm *tm_info = localtime(&t);
   current_year = tm_info->tm_year + 1900;
```

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

    printf("Files created in month number %d:\n", target_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, &st) == 0) {
            struct tm *file_tm = localtime(&st.st_ctime);

            if (file_tm->tm_mon + 1 == target_month &&
                file_tm->tm_year + 1900 == current_year) {
                printf("%s\n", entry->d_name);
            }
        }
    }
    closedir(dir);
    return 0;
}
```

OUTPUT:

Enter month number (1-12): 11

Files created in month number 11:

report.pdf

output.txt

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/wait.h>
#include <sys/resource.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  int n;
  if (argc != 2) {
    printf("Usage: %s <number_of_children>\n", argv[0]);
    return 1;
  }
  n = atoi(argv[1]);
  if (n <= 0) {
    printf("Number of children must be positive.\n");
    return 1;
  }
```

```
struct timeval total_utime = {0, 0}; // user time
struct timeval total_stime = {0, 0}; // sys time

for (int i = 0; i < n; i++) {
   pid_t pid = fork();
   if (pid < 0) {
      perror("fork");
      exit(1);
   }
   if (pid == 0) {
      // Child: do some work
      for (volatile long j = 0; j < 10000000; ++j); // dummy load
      exit(0);
   }
}

// Parent waits for n children and accumulates their usage times
for (int i = 0; i < n; i++) {
   struct rusage usage;
   int status;
   wait4(-1, &status, 0, &usage);

   // Add user times
   total_utime.tv_sec  += usage.ru_utime.tv_sec;
   total_utime.tv_usec += usage.ru_utime.tv_usec;
   // Add system times
   total_stime.tv_sec  += usage.ru_stime.tv_sec;
   total_stime.tv_usec += usage.ru_stime.tv_usec;
```

```
    // Normalize microseconds (in case > 1_000_000)
    if (total_utime.tv_usec >= 1000000) {
      total_utime.tv_sec += total_utime.tv_usec / 1000000;
      total_utime.tv_usec = total_utime.tv_usec % 1000000;
    }
    if (total_stime.tv_usec >= 1000000) {
      total_stime.tv_sec += total_stime.tv_usec / 1000000;
      total_stime.tv_usec = total_stime.tv_usec % 1000000;
    }
  }

  printf("Total cumulative time children spent:\n");
  printf("User mode  : %ld.%06ld seconds\n", (long)total_utime.tv_sec,
(long)total_utime.tv_usec);
  printf("Kernel mode: %ld.%06ld seconds\n", (long)total_stime.tv_sec,
(long)total_stime.tv_usec);

  return 0;
}
```

OUTPUT:

./a.out 3

Total cumulative time children spent:

User mode : 0.050123 seconds

Kernel mode: 0.005432 seconds

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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("output.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
  if (fd < 0) {
    perror("open");
    return 1;
  }
  // Duplicate fd to standard output (fd = 1)
  dup2(fd, STDOUT_FILENO);
  close(fd);

  // All printf output will go to "output.txt"
  printf("This message will be written to output.txt instead of the terminal.\n");
  printf("Second line redirected to the file.\n");

  return 0;
}
```

OUTPUT:

This message will be written to output.txt instead of the terminal.

Second line redirected to the file.

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

int main() {
  int fd[2];
  pipe(fd);

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

  if (pid == 0) {
    // Child: runs 'ls -l', writes to pipe
    close(fd[0]);        // Close read end
    dup2(fd[1], STDOUT_FILENO); // Redirect stdout to pipe's write end
```

```
        close(fd[1]);
        execlp("ls", "ls", "-l", NULL);
        perror("execlp");     // Only if execlp fails
        exit(1);
    } else {
        // Parent: runs 'wc -l', reads from pipe
        close(fd[1]);         // Close write end
        dup2(fd[0], STDIN_FILENO);  // Redirect stdin to pipe's read end
        close(fd[0]);
        execlp("wc", "wc", "-l", NULL);
        perror("execlp");     // Only if execlp fails
        exit(1);
    }
    return 0;
}
```

OUTPUT:

6

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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() {
  // Open output.txt for writing. Create if not exist, truncate if exists, permissions 0644
  int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0) {
    perror("open");
    return 1;
  }

  // Duplicate fd onto standard output (fd 1)
  if (dup2(fd, STDOUT_FILENO) < 0) {
    perror("dup2");
    close(fd);
    return 1;
  }
  close(fd); // fd no longer needed
  // Now all printf/fputs/puts to stdout will go to output.txt
```

```
    printf("This line will be written to output.txt\n");
    printf("Redirection works using open() and dup2() system calls.\n");

    return 0;
}
```

OUTPUT:

This line will be written to output.txt

Redirection works using open() and dup2() system calls.

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

int main() {
    int fd[2];
    pipe(fd);

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

    if (pid == 0) {
```

```
    // Child: runs 'ls -l', writes to pipe
    close(fd[0]);        // Close read end
    dup2(fd[1], STDOUT_FILENO); // Redirect stdout to pipe's write end
    close(fd[1]);
    execlp("ls", "ls", "-l", NULL);
    perror("execlp");    // Only if execlp fails
    exit(1);
  } else {
    // Parent: runs 'wc -l', reads from pipe
    close(fd[1]);        // Close write end
    dup2(fd[0], STDIN_FILENO);  // Redirect stdin to pipe's read end
    close(fd[0]);
    execlp("wc", "wc", "-l", NULL);
    perror("execlp");    // Only if execlp fails
    exit(1);
  }
  return 0;
}
```

OUTPUT:

6

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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 via Pipe!";
   char read_msg[100];

   // Create pipe
   if (pipe(fd) == -1) {
      perror("pipe");
      return 1;
   }

   // Write to pipe
   write(fd[1], write_msg, strlen(write_msg) + 1); // +1 for null terminator

   // Read from pipe
```

```
    read(fd[0], read_msg, sizeof(read_msg));

    printf("Read from pipe: %s\n", read_msg);

    // Close pipe file descriptors
    close(fd[0]);
    close(fd[1]);
    return 0;
}
```

OUTPUT:

Read from pipe: Hello from Parent via Pipe!

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. [20 Marks]

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

void print_type(mode_t mode) {
    if (S_ISREG(mode))
        printf("Regular file\n");
    else if (S_ISDIR(mode))
        printf("Directory\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_ISLNK(mode))
        printf("Symbolic link\n");
    else if (S_ISSOCK(mode))
        printf("Socket\n");
    else
        printf("Unknown file type\n");
}

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) == -1) {
        perror("stat");
        return 1;
    }
    printf("File: %s\nType: ", argv[1]);
    print_type(st.st_mode);
    return 0;
}
```

OUTPUT:

./a.out example.txt

File: example.txt

Type: Regular file

./a.out .

File: .

Type: Directory

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 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];
  pid_t pid1, pid2;

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

  pid1 = fork();
  if (pid1 == 0) {
    // First child - cmd1: output to pipe
    close(fd[0]); // Close unused read end
    dup2(fd[1], STDOUT_FILENO);
```

```
    close(fd[1]);
    execlp("ls", "ls", "-l", NULL);
    perror("exec ls");
    exit(1);
  }

  pid2 = fork();
  if (pid2 == 0) {
    // Second child - cmd2: input from pipe
    close(fd[1]); // Close unused write end
    dup2(fd[0], STDIN_FILENO);
    close(fd[0]);
    execlp("wc", "wc", "-l", NULL);
    perror("exec wc");
    exit(1);
  }

  // Parent closes pipe and waits for both children
  close(fd[0]);
  close(fd[1]);
  waitpid(pid1, NULL, 0);
  waitpid(pid2, NULL, 0);

  return 0;
}
```

OUTPUT:

6

Q.2) 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>
#include <stdlib.h>

int main() {
  int fd[2];
  pid_t pid;
  char message[] = "Hello from parent!";
  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 unused read end
    write(fd[1], message, strlen(message) + 1); // +1 for '\0'
```

```
        close(fd[1]);
        wait(NULL);
    } else {
        // Child: read from pipe
        close(fd[1]); // Close unused write end
        read(fd[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(fd[0]);
        exit(0);
    }
    return 0;
}
```

Output:

Child received: Hello from parent!

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 11

Q.1) Write a C program to get and set the resource limits such as files, memory associated with a process [10 Marks]

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

void print_limit(const char* name, int resource) {
   struct rlimit limit;
   if (getrlimit(resource, &limit) == 0) {
      printf("%s: Soft = %ld, Hard = %ld\n",
         name,
         (long)limit.rlim_cur,
         (long)limit.rlim_max);
   } else {
      perror("getrlimit");
   }
}

int main() {
   printf("** Current Resource Limits **\n");
   print_limit("Open files (RLIMIT_NOFILE)", RLIMIT_NOFILE);
```

```
print_limit("Virtual memory (RLIMIT_AS)", RLIMIT_AS);

// For demonstration, set open file limit to 512 (soft and hard)
struct rlimit new_limit;
new_limit.rlim_cur = 512;
new_limit.rlim_max = 512;
if (setrlimit(RLIMIT_NOFILE, &new_limit) == 0) {
   printf("\nSet open file limit to 512 successfully!\n");
   print_limit("Open files (RLIMIT_NOFILE)", RLIMIT_NOFILE);
} else {
   perror("setrlimit");
   printf("Could not set open files limit (need root or higher value than allowed).\n");
}

// For demonstration, set virtual memory limit to 128MB soft, 256MB hard
new_limit.rlim_cur = 128 * 1024 * 1024;
new_limit.rlim_max = 256 * 1024 * 1024;
if (setrlimit(RLIMIT_AS, &new_limit) == 0) {
   printf("\nSet virtual memory limit successfully!\n");
   print_limit("Virtual memory (RLIMIT_AS)", RLIMIT_AS);
} else {
   perror("setrlimit");
   printf("Could not set virtual memory limit (may require privileges).\n");
}

return 0;
```

```
}
```

OUTPUT:

**** Current Resource Limits ****

Open files (RLIMIT_NOFILE): Soft = 1024, Hard = 4096

Virtual memory (RLIMIT_AS): Soft = unlimited, Hard = unlimited

Set open file limit to 512 successfully!

Open files (RLIMIT_NOFILE): Soft = 512, Hard = 512

Set virtual memory limit successfully!

Virtual memory (RLIMIT_AS): Soft = 134217728, Hard = 268435456

Q.2) Write a C program that redirects standard output to a file output.txt. (use of dup and open system call). [20 Marks]

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

int main() {
   int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644); // Open file for writing
   if (fd < 0) {
      perror("open");
```

```
    return 1;
  }

  // Duplicate fd onto STDOUT_FILENO (file descriptor 1)
  if (dup2(fd, STDOUT_FILENO) < 0) {
    perror("dup2");
    close(fd);
    return 1;
  }

  close(fd); // We can safely close fd

  // Now, any output to stdout goes into output.txt
  printf("This text will be redirected to output.txt using dup2() and open().\n");
  printf("Second line of redirected output!\n");

  return 0;
}
```

OUTPUT:

This text will be redirected to output.txt using dup2() and open().

Second line of redirected output!

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 12

Q.1) Write a C program that print the exit status of a terminated child process. [10 Marks]

```
#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 *fa = (struct FileInfo*)a;
   struct FileInfo *fb = (struct FileInfo*)b;
   if (fa->size < fb->size)
      return -1;
   else if (fa->size > fb->size)
      return 1;
   else
      return 0;
}
```

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

    struct FileInfo *files = malloc((argc-1)*sizeof(struct FileInfo));
    int count = 0;
    for (int i = 1; i < argc; ++i) {
        struct stat st;
        if (stat(argv[i], &st) == 0) {
            strncpy(files[count].name, argv[i], 255);
            files[count].name[255] = '\0';
            files[count].size = st.st_size;
            count++;
        } else {
            printf("Cannot access file: %s\n", argv[i]);
        }
    }

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

    printf("Files in ascending order by size:\n");
    for (int i = 0; i < count; ++i) {
        printf("%s (%ld bytes)\n", files[i].name, files[i].size);
    }
    free(files);
```

```
    return 0;
}
```

OUTPUT:

Suppose you create three text files with the following sizes:

- **a.txt (40 bytes)**
- **b.txt (140 bytes)**
- **c.txt (1200 bytes)**

./a.out a.txt b.txt c.txt

Files in ascending order by size:

a.txt (40 bytes)

b.txt (140 bytes)

c.txt (1200 bytes)

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, ...) [20 Marks]

```
#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 *fa = (struct FileInfo*)a;
   struct FileInfo *fb = (struct FileInfo*)b;
   if (fa->size < fb->size)
      return -1;
   else if (fa->size > fb->size)
      return 1;
   else
      return 0;
}

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

   struct FileInfo *files = malloc((argc-1)*sizeof(struct FileInfo));
   int count = 0;
   for (int i = 1; i < argc; ++i) {
      struct stat st;
      if (stat(argv[i], &st) == 0) {
```

```
            strncpy(files[count].name, argv[i], 255);
            files[count].name[255] = '\0';
            files[count].size = st.st_size;
            count++;
        } else {
            printf("Cannot access file: %s\n", argv[i]);
        }
    }

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

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

    return 0;
}
```

OUTPUT:

If you have:

- **sample1.txt (512 bytes)**
- **sample2.txt (64 bytes)**
- **sample3.txt (4096 bytes)**

./a.out sample1.txt sample2.txt sample3.txt

Files in ascending order by size:

sample2.txt (64 bytes)

sample1.txt (512 bytes)

sample3.txt (4096 bytes)

if One file missing.

Cannot access file: missing.txt

Files in ascending order by size:

sample1.txt (512 bytes)

sample3.txt (4096 bytes)

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 13

Q.1) Write a C program that illustrates suspending and resuming processes using signals [10 Marks]

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

int main() {
  pid_t pid = fork();
  if (pid < 0) {
    perror("fork");
    return 1;
  }

  if (pid == 0) {
    // Child process: print numbers continuously
    int i = 1;
    while (i <= 20) {
      printf("Child: %d\n", i++);
      sleep(1);
```

```
        }
        exit(0);
    } else {
        // Parent process: suspend and resume child
        sleep(3);
        printf("Parent: Suspending child (SIGSTOP)\n");
        kill(pid, SIGSTOP);
        sleep(5);
        printf("Parent: Resuming child (SIGCONT)\n");
        kill(pid, SIGCONT);
        wait(NULL); // Wait for child to finish
        printf("Parent: Child finished.\n");
    }
    return 0;
}
```

OUTPUT:

Child: 1

Child: 2

Child: 3

Parent: Suspending child (SIGSTOP)

(about 5 seconds pause...)

Parent: Resuming child (SIGCONT)

Child: 4

Child: 5

Child: 6

Child: 7

Child: 8

Child: 9

Child: 10

Child: 11

Child: 12

Child: 13

Child: 14

Child: 15

Child: 16

Child: 17

Child: 18

Child: 19

Child: 20

Parent: Child finished.

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 [20 Marks]

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

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

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

    struct dirent *entry;
    printf("Files beginning with '%s':\n", prefix);
    while ((entry = readdir(dir)) != NULL) {
        if (strncmp(entry->d_name, prefix, strlen(prefix)) == 0) {
            printf("%s\n", entry->d_name);
        }
    }
    closedir(dir);
    return 0;
}
```

OUTPUT:

./a.out foo

Files beginning with 'foo':

foo.txt

foobar.c

food.csv

./a.out z

Files beginning with 'z':

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

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

int main() {
  long n;
  printf("Enter the size in bytes: ");
  scanf("%ld", &n);

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

  struct dirent *entry;
  struct stat st;

  printf("Files with size greater than %ld bytes:\n", n);
```

```
    while ((entry = readdir(dir)) != NULL) {
        // Skip directories (optional)
        if (stat(entry->d_name, &st) == 0 && S_ISREG(st.st_mode)) {
            if (st.st_size > n) {
                printf("%s (%ld bytes)\n", entry->d_name, (long)st.st_size);
            }
        }
    }
    closedir(dir);
    return 0;
}
```

OUTPUT:

Enter the size in bytes: 100

Files with size greater than 100 bytes:

bigfile.dat (3000 bytes)

Files with size greater than 10 bytes:

a.txt (40 bytes)

bigfile.dat (3000 bytes)

notes.txt (100 bytes)

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. [20 Marks]

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.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");
}

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) == -1) {
    perror("stat");
    return 1;
  }

  printf("File: %s\n", argv[1]);
  printf("Inode number      : %lu\n", (unsigned long)st.st_ino);
  printf("Number of hard links: %lu\n", (unsigned long)st.st_nlink);
  print_permissions(st.st_mode);
  printf("File size         : %ld bytes\n", (long)st.st_size);

  struct passwd *pw = getpwuid(st.st_uid);
  struct group  *gr = getgrgid(st.st_gid);

  printf("Owner UID: %u (%s)\n", st.st_uid, pw ? pw->pw_name : "Unknown");
  printf("Group GID: %u (%s)\n", st.st_gid, gr ? gr->gr_name : "Unknown");

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

  return 0;
}
```

OUTPUT:

./a.out bigfile.dat

File: bigfile.dat

Inode number : 4561234

Number of hard links: 1

Permissions: -rw-r--r--

File size : 3000 bytes

Owner UID: 1000 (princeshubh-dev)

Group GID: 1000 (princeshubh-dev)

Last access time : Sat Nov 8 17:20:20 2025

Last modification : Sat Nov 8 17:19:50 2025

Last status change : Sat Nov 8 17:19:50 2025

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 15

Q.1) Display all the files from current directory whose size is greater that n Bytes Where n is accept from user [10 Marks]

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

int main() {
  long n;
  printf("Enter the minimum file size in bytes: ");
  scanf("%ld", &n);

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

  struct dirent *entry;
  struct stat st;

  printf("Files with size greater than %ld bytes:\n", n);
```

```
    while ((entry = readdir(dir)) != NULL) {
        // Only check regular files (skip directories and special files)
        if (stat(entry->d_name, &st) == 0 && S_ISREG(st.st_mode)) {
            if (st.st_size > n) {
                printf("%s (%ld bytes)\n", entry->d_name, (long)st.st_size);
            }
        }
    }
    closedir(dir);
    return 0;
}
```

OUTPUT:

Enter the minimum file size in bytes: 100

Output: ***(Same as above, shows files >100 bytes)***

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 alarm_handler(int sig) {
   printf("ALARM FIRED: Child exceeded execution time. Killing child...\n");
   if (child_pid > 0) {
      kill(child_pid, SIGKILL);
   }
}

void child_death_handler(int sig) {
   printf("Child process terminated (death signal received: SIGCHLD).\n");
}

int main(int argc, char *argv[]) {
   if (argc < 2) {
      printf("Usage: %s <command> [arg1 ... argN]\n", argv[0]);
      exit(1);
   }

   signal(SIGCHLD, child_death_handler);
   signal(SIGALRM, alarm_handler);

   child_pid = fork();

   if (child_pid < 0) {
```

```
    perror("fork");
    exit(1);
}

if (child_pid == 0) {
    // Child process
    execvp(argv[1], &argv[1]);
    // If exec fails
    perror("execvp");
    exit(1);
} else {
    // Parent process
    alarm(5); // Set timeout for 5 seconds

    int status;
    pid_t result = waitpid(child_pid, &status, 0); // Wait for child

    alarm(0); // Cancel alarm if child ends in time

    if (result == child_pid) {
        if (WIFEXITED(status)) {
            printf("Child exited normally with status %d\n", WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
            printf("Child killed by signal %d\n", WTERMSIG(status));
        }
    }
```

```
    }
    return 0;
}
```

OUTPUT:

./a.out sleep 10

ALARM FIRED: Child exceeded execution time. Killing child...

Child process terminated (death signal received: SIGCHLD).

Child killed by signal 9

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 16

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

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

int main() {
   int month;
   printf("Enter target month (1-12): ");
   scanf("%d", &month);

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

   struct dirent *entry;
   struct stat st;
```

```
    printf("Files modified/created in month %d:\n", month);
    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &st) == 0 && S_ISREG(st.st_mode)) {
            struct tm *mtime = localtime(&st.st_mtime);
            if ((mtime->tm_mon + 1) == month) {
                printf("%s\n", entry->d_name);
            }
        }
    }
    closedir(dir);
    return 0;
}
```

OUTPUT:

Enter target month (1-12): 11……..

Files modified/created in month 11:

notes.txt

bigfile.dat

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!!!”. [20 Marks]

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

```
volatile sig_atomic_t running = 1;

void handle_sighup(int sig) {
   printf("Child: Received SIGHUP\n");
}
void handle_sigint(int sig) {
   printf("Child: Received SIGINT\n");
}
void handle_sigquit(int sig) {
   printf("My DADDY has Killed me!!!\n");
   running = 0;
}

int main() {
   pid_t pid = fork();
   if (pid < 0) {
      perror("fork");
      exit(1);
   }
   if (pid == 0) {
      // Child process setup handlers
      signal(SIGHUP, handle_sighup);
      signal(SIGINT, handle_sigint);
      signal(SIGQUIT, handle_sigquit);

      while (running) {
         pause(); // Wait for signals
      }
```

```
        exit(0);
    } else {
        // Parent process sends signals
        for (int elapsed = 0; elapsed < 30; elapsed += 3) {
            sleep(3);
            // Alternate between SIGHUP and SIGINT
            if ((elapsed / 3) % 2 == 0)
                kill(pid, SIGHUP);
            else
                kill(pid, SIGINT);
        }
        // After 30 seconds, send SIGQUIT
        kill(pid, SIGQUIT);
        wait(NULL); // Wait for child to exit
    }
    return 0;
}
```

OUTPUT:

Child: Received SIGHUP

Child: Received SIGINT

Child: Received SIGHUP

Child: Received SIGINT

Child: Received SIGHUP

Child: Received SIGINT

Child: Received SIGHUP

Child: Received SIGINT

Child: Received SIGHUP Child: Received SIGINT My DADDY has Killed me!!!

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 17

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>

int main() {
   DIR *dir;
   struct dirent *entry;
   int count = 0;

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

   printf("Files in current directory:\n");
   while ((entry = readdir(dir)) != NULL) {
      // Skip . and .. entries if desired
      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);
      count++;
    }
    closedir(dir);

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

OUTPUT:

Files in current directory:

main.c

notes.txt

foo.txt

bigfile.dat

Total number of files: 4

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 [20 Marks]

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

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

  // Block SIGINT (Ctrl-C) and SIGQUIT (Ctrl-\)
  sigemptyset(&block_set);
  sigaddset(&block_set, SIGINT);
  sigaddset(&block_set, SIGQUIT);
  sigprocmask(SIG_BLOCK, &block_set, &old_set);

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

```
pid1 = fork();
if (pid1 == 0) {
   // First child: ls -l
   sigprocmask(SIG_SETMASK, &old_set, NULL); // restore signals for child
   close(fd[0]); // Close read end
   dup2(fd[1], STDOUT_FILENO);
   close(fd[1]);
   execlp("ls", "ls", "-l", NULL);
   perror("execlp ls");
   exit(1);
}

pid2 = fork();
if (pid2 == 0) {
   // Second child: wc -l
   sigprocmask(SIG_SETMASK, &old_set, NULL); // restore signals for child
   close(fd[1]); // Close write end
   dup2(fd[0], STDIN_FILENO);
   close(fd[0]);
   execlp("wc", "wc", "-l", NULL);
   perror("execlp wc");
   exit(1);
}

// Parent: close pipe ends and wait for children
```

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

    // Restore original signal mask
    sigprocmask(SIG_SETMASK, &old_set, NULL);
    return 0;
}
```

OUTPUT:

6

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 18

Q.1) Write a C program to find whether a given file is present in current directory or not [10 Marks]

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

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

  struct dirent *entry;
  int found = 0;
```

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

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

    return 0;
}
```

OUTPUT:

- a.txt (40 bytes)
- b.txt (140 bytes)
- c.txt (1200 bytes)
- foo.txt (100 bytes)
- output.txt (created by redirection programs)

./a.out a.txt

File 'a.txt' is present in the current directory.

./a.out missing.txt

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 <sys/wait.h>

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

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

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

```
if (pid == 0) {
    // Child writes to pipe, close read end
    close(fd[0]);
    char *messages[] = {
        "Hello World",
        "Hello SPPU",
        "Linux is Funny"
    };
    for (int i = 0; i < 3; i++) {
        write(fd[1], messages[i], strlen(messages[i]) + 1); // include null terminator
    }
    close(fd[1]);
    return 0;
} else {
    // Parent reads from pipe, close write end
    close(fd[1]);
    printf("Messages received from child:\n");
    for (int i = 0; i < 3; i++) {
        read(fd[0], buffer, sizeof(buffer));
        printf("%s\n", buffer);
    }
    close(fd[0]);
    wait(NULL); // Ensure child has finished
}
return 0;
```

```
}
```

OUTPUT:

Messages received from child:

Hello World

Hello SPPU

Linux is Funny

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 19

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

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

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

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

```
  }
  for (int i = 1; i < argc; ++i) {
    struct stat st;
    if (stat(argv[i], &st) != 0) {
      perror(argv[i]);
      continue;
    }
    printf("File: %s\n", argv[i]);
    printf("  Type     : %s\n", file_type(st.st_mode));
    printf("  Inode No. : %lu\n\n", (unsigned long)st.st_ino);
  }
  return 0;
}
```

OUTPUT:

./a.out a.txt foo.txt c.txt missing.txt

File: a.txt

Type : Regular File

Inode No. : 1031300

File: foo.txt

Type : Regular File

Inode No. : 1031301

File: c.txt

Type : Regular File

Inode No. : 1031302

missing.txt: No such file or directory

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

int main() {
  int fd[2];
  pipe(fd);

  pid_t pid1 = fork();
  if (pid1 < 0) {
    perror("fork");
    exit(1);
  }
  if (pid1 == 0) {
    // First child: ls -l > pipe
    close(fd[0]); // Close read end
    dup2(fd[1], STDOUT_FILENO);
    close(fd[1]);
    execlp("ls", "ls", "-l", NULL);
    perror("execlp ls");
    exit(1);
  }

  pid_t pid2 = fork();
```

```
    if (pid2 < 0) {
        perror("fork");
        exit(1);
    }
    if (pid2 == 0) {
        // Second child: wc -l < pipe
        close(fd[1]); // Close write end
        dup2(fd[0], STDIN_FILENO);
        close(fd[0]);
        execlp("wc", "wc", "-l", NULL);
        perror("execlp wc");
        exit(1);
    }

    // Parent: close both pipe ends and wait for children
    close(fd[0]);
    close(fd[1]);
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return 0;
}
```

OUTPUT:

6

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 20

Q.1) Write a C program that illustrates suspending and resuming processes using signals [10 Marks]

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

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

  if (pid < 0) {
    perror("fork");
    exit(1);
  }
  if (pid == 0) {
    // Child process prints a message every second
    int i = 1;
    while (i <= 20) {
      printf("Child process running: iteration %d\n", i++);
      sleep(1);
    }
```

```
        exit(0);
    } else {
        // Parent process
        sleep(3);
        printf("Parent: Suspending child (SIGSTOP)\n");
        kill(pid, SIGSTOP);

        sleep(5);
        printf("Parent: Resuming child (SIGCONT)\n");
        kill(pid, SIGCONT);

        wait(NULL);
        printf("Parent: Child process finished.\n");
    }
    return 0;
}
```

OUTPUT:

Child process running: iteration 1

Child process running: iteration 2

Child process running: iteration 3

Parent: Suspending child (SIGSTOP)

Parent: Resuming child (SIGCONT)

Child process running: iteration 4

Child process running: iteration 5

Child process running: iteration 6

Child process running: iteration 7

Child process running: iteration 8

Child process running: iteration 9

Child process running: iteration 10

Child process running: iteration 11

Child process running: iteration 12

Child process running: iteration 13

Child process running: iteration 14

Child process running: iteration 15

Child process running: iteration 16

Child process running: iteration 17

Child process running: iteration 18

Child process running: iteration 19

Child process running: iteration 20

Parent: Child process finished.

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. [20 Marks]

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

void print_file_type(mode_t mode) {
   if (S_ISREG(mode))
      printf("Regular file\n");
```

```
    else if (S_ISDIR(mode))
        printf("Directory\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/Namded Pipe\n");
    else if (S_ISLNK(mode))
        printf("Symbolic link\n");
    else if (S_ISSOCK(mode))
        printf("Socket\n");
    else
        printf("Unknown type\n");
}

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) == -1) {
        perror("stat");
        return 1;
    }
```

```
    printf("File: %s\nType: ", argv[1]);
    print_file_type(st.st_mode);
    return 0;
}
```

OUTPUT:

./a.out a.txt

File: a.txt

Type: Regular file

./a.out .

File: .

Type: Directory

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 21

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 *dir;
   struct dirent *entry;
   int count = 0;

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

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

```
    while ((entry = readdir(dir)) != NULL) {
        // Skip . and .. if desired
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
            continue;
        printf("%s\n", entry->d_name);
        count++;
    }
    closedir(dir);
    printf("Total number of files: %d\n", count);


    return 0;
}
```

OUTPUT:

Files in current directory:

a.txt

b.txt

c.txt

foo.txt

output.txt

Total number of files: 5

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, ...) [20 Marks]

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

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

int compare(const void *a, const void *b) {
   struct FileInfo *fa = (struct FileInfo *)a;
   struct FileInfo *fb = (struct FileInfo *)b;
   if (fa->size < fb->size) return -1;
   if (fa->size > fb->size) return 1;
   return 0;
}

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

```
    }

    struct FileInfo *files = malloc((argc-1) * sizeof(struct FileInfo));
    int count = 0;

    for (int i = 1; i < argc; i++) {
        struct stat st;
        if (stat(argv[i], &st) == 0) {
            strncpy(files[count].name, argv[i], 255);
            files[count].name[255] = '\0';
            files[count].size = st.st_size;
            count++;
        } else {
            printf("Cannot access file: %s\n", argv[i]);
        }
    }

    qsort(files, count, sizeof(struct FileInfo), compare);
    printf("Files in ascending order of size:\n");
    for (int i = 0; i < count; i++) {
        printf("%s (%ld bytes)\n", files[i].name, (long)files[i].size);
    }

    free(files);
    return 0;
}
```

OUTPUT:

./a.out a.txt b.txt c.txt foo.txt

Files in ascending order of size:

a.txt (40 bytes)

foo.txt (100 bytes)

b.txt (140 bytes)

c.txt (1200 bytes)

If a file is missing:

Cannot access file: missing.txt

Files in ascending order of size:

a.txt (40 bytes)

foo.txt (100 bytes)

b.txt (140 bytes)

c.txt (1200 bytes)

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 22

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("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0) {
    perror("open");
    return 1;
  }
  // Redirect STDOUT to fd
  if (dup2(fd, STDOUT_FILENO) < 0) {
    perror("dup2");
    close(fd);
    return 1;
  }
  close(fd); // fd is now duplicated to STDOUT, so can close

  printf("This line is written to output.txt\n");
```

```
    printf("Redirection using dup2() and open()\n");
    return 0;
}
```

OUTPUT:

Terminal Output:
(No output on terminal)

This line is written to output.txt

Redirection using dup2() and open()

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 [20 Marks]

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

int main() {
    int fd[2];
    pipe(fd);
```

```
// Block SIGINT and SIGQUIT
sigset_t block_set, old_set;
sigemptyset(&block_set);
sigaddset(&block_set, SIGINT);   // Ctrl-C
sigaddset(&block_set, SIGQUIT);  // Ctrl-\
sigprocmask(SIG_BLOCK, &block_set, &old_set);

pid_t pid1 = fork();
if (pid1 == 0) { // First child: ls -l
   // Restore signal mask for child
   sigprocmask(SIG_SETMASK, &old_set, NULL);
   close(fd[0]);
   dup2(fd[1], STDOUT_FILENO);
   close(fd[1]);
   execlp("ls", "ls", "-l", NULL);
   perror("execlp-ls");
   exit(1);
}

pid_t pid2 = fork();
if (pid2 == 0) { // Second child: wc -l
   // Restore signal mask for child
   sigprocmask(SIG_SETMASK, &old_set, NULL);
   close(fd[1]);
   dup2(fd[0], STDIN_FILENO);
   close(fd[0]);
   execlp("wc", "wc", "-l", NULL);
   perror("execlp-wc");
```

```
        exit(1);
    }

    // Parent closes both ends and waits
    close(fd[0]);
    close(fd[1]);
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    // Restore original signal mask after critical section
    sigprocmask(SIG_SETMASK, &old_set, NULL);
    return 0;
}
```

OUTPUT:

6

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 23

Q.1) Write a C program to find whether a given file is present in current directory or not [10 Marks]

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

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return 1;
  }
  char *filename = argv[1];
  DIR *dir = opendir(".");
  if (!dir) {
    perror("opendir");
    return 1;
  }
  struct dirent *entry;
  int found = 0;
  while ((entry = readdir(dir)) != NULL) {
    if (strcmp(entry->d_name, filename) == 0) {
      found = 1;
```

```
            break;
        }
    }
    closedir(dir);
    if (found)
        printf("File '%s' is present in the current directory.\n", filename);
    else
        printf("File '%s' is NOT present in the current directory.\n", filename);
    return 0;
}
```

OUTPUT::

./a.out a.txt

File 'a.txt' is present in the current directory.

If the file does NOT exist:

File 'missing.txt' is NOT present in the current directory.

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.

[20 Marks]

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

void print_file_type(mode_t mode) {
    if (S_ISREG(mode))
```

```
        printf("Regular file\n");
    else if (S_ISDIR(mode))
        printf("Directory\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_ISLNK(mode))
        printf("Symbolic link\n");
    else if (S_ISSOCK(mode))
        printf("Socket\n");
    else
        printf("Unknown type\n");
}

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) == -1) {
        perror("stat");
        return 1;
    }
    printf("File: %s\nType: ", argv[1]);
```

```
    print_file_type(st.st_mode);
    return 0;
}
```

OuTPUT:

./a.out a.txt

File: a.txt

Type: Regular file

If you check a directory:

./a.out .

File: .

Type: Directory

For a device (e.g. /dev/null):

File: /dev/null

Type: Character device

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 24

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>

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

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

```
    struct stat st;
    if (stat(argv[1], &st) != 0) {
        perror("stat");
        return 1;
    }

    printf("File: %s\nType: %s\nInode: %lu\n",
           argv[1],
           file_type(st.st_mode),
           (unsigned long)st.st_ino);

    return 0;
}
```

OUTPUT:

./a.out foo.txt

File: foo.txt

Type: Regular File

Inode: 1234567

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 handle_sigchld(int sig) {
   printf("Parent: Child process terminated (SIGCHLD received)\n");
}

void handle_sigalrm(int sig) {
   printf("Parent: Child process did not finish in 5 seconds. Killing child!\n");
   if (child_pid > 0) {
      kill(child_pid, SIGKILL);
   }
}

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

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

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

if (child_pid == 0) {
   // Child executes command
   execvp(argv[1], &argv[1]);
   perror("execvp failed");
   exit(1);
} else {
   // Parent process
   alarm(5); // Set alarm for 5 seconds

   int status;
   waitpid(child_pid, &status, 0);

   alarm(0); // Cancel alarm if child finished in time
   if (WIFEXITED(status)) {
      printf("Child exited with status %d\n", WEXITSTATUS(status));
   } else if (WIFSIGNALED(status)) {
      printf("Child killed by signal %d\n", WTERMSIG(status));
   }
}
```

```
    return 0;
}
```

OUTPUT:

./a.out sleep 10

Parent: Child process terminated (SIGCHLD received)

Parent: Child process did not finish in 5 seconds. Killing child!

Child killed by signal 9

If child finishes in time (e.g. sleep 2):

./a.out sleep 2

Parent: Child process terminated (SIGCHLD received)

Child exited with status 0

SUBJECT: CS-504-MJP: Lab Course on CS-501-MJ (Advanced Operating System)

Assignment 25

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() {
  // Open file "output.txt" for writing, create if not exist, truncate if exists, permissions 0644
  int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0) {
    perror("open");
    return 1;
  }

  // Redirect stdout to the file (file descriptor 1)
  dup2(fd, STDOUT_FILENO);

  // From now, printf writes to "output.txt"
  printf("This will be written to output.txt instead of the terminal.\n");
  printf("Redirection of standard output is successful!\n");

  close(fd); // optionally close, not strictly needed here
```

```
    return 0;
}
```

OUTPUT:

No output on terminal.

This will be written to output.txt instead of the terminal.

Redirection of standard output is successful!

Q.2) Write a C program that redirects standard output to a file output.txt. (use of dup and open system call). [20 Marks]

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

int main() {
    int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644); // open file
    if (fd < 0) {
        perror("open");
        return 1;
    }

    // Use dup2() to duplicate fd to STDOUT_FILENO
    if (dup2(fd, STDOUT_FILENO) < 0) {
        perror("dup2");
        close(fd);
```

```
        return 1;
    }
    close(fd); // fd no longer needed

    // All stdout output goes to output.txt
    printf("This line is redirected to output.txt using dup2 and open.\n");
    printf("Second line: testing redirection.\n");

    return 0;
}
```

OUTPUT:

No output on terminal.

This line is redirected to output.txt using dup2 and open.

Second line: testing redirection.