Slip 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/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.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 fileStat;

      if (stat(argv[i], &fileStat) < 0) {
         fprintf(stderr, "Error accessing file %s: %s\n", argv[i], strerror(errno));
         continue;
      }

      printf("File: %s\n", argv[i]);
      printf("Inode Number: %ld\n", (long)fileStat.st_ino);

      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_ISLNK(fileStat.st_mode)) {
            printf("File Type: Symbolic Link\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_ISSOCK(fileStat.st_mode)) {
            printf("File Type: Socket\n");
        } else {
            printf("File Type: Unknown\n");
        }

        printf("\n");
    }

    return 0;
}
```

Output:

```
 File: file1.txt
Inode Number: 123456
File Type: Regular file

File: file2.txt
Inode Number: 789012
File Type: Regular file
```

File: directory

Inode Number: 345678

File Type: Directory

Q2. 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 handle_alarm(int signum) {
    if (signum == SIGALRM) {
        printf("Alarm is fired!\n");
    }
}

int main() {
    pid_t pid;
    signal(SIGALRM, handle_alarm); // Registering the signal handler

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
```

```
    } else if (pid == 0) { // Child process
        sleep(2); // Waiting for 2 seconds
        kill(getppid(), SIGALRM); // Sending SIGALRM signal to the parent
    } else { // Parent process
        printf("Waiting for alarm...\n");
        while(1) {
            // Parent process waits indefinitely for the signal
            // The signal handler will execute upon receiving SIGALRM
        }
    }

    return 0;
}
```

Output:

Waiting for alarm...

Alarm is fired!

Slip 2

Q1. 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 <unistd.h>
#include <errno.h>
#include <time.h>

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) {
    fprintf(stderr, "Error accessing file %s: %s\n", argv[1], strerror(errno));
    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("File Permissions: ");
printf((S_ISDIR(fileStat.st_mode)) ? "d" : "-");
printf((fileStat.st_mode & S_IRUSR) ? "r" : "-");
printf((fileStat.st_mode & S_IWUSR) ? "w" : "-");
printf((fileStat.st_mode & S_IXUSR) ? "x" : "-");
printf((fileStat.st_mode & S_IRGRP) ? "r" : "-");
printf((fileStat.st_mode & S_IWGRP) ? "w" : "-");
printf((fileStat.st_mode & S_IXGRP) ? "x" : "-");
printf((fileStat.st_mode & S_IROTH) ? "r" : "-");
printf((fileStat.st_mode & S_IWOTH) ? "w" : "-");
printf((fileStat.st_mode & S_IXOTH) ? "x\n" : "-\n");
printf("File Size: %ld bytes\n", (long)fileStat.st_size);
printf("Last Access Time: %s", ctime(&fileStat.st_atime));
printf("Last Modification Time: %s", ctime(&fileStat.st_mtime));

return 0;
```

}

Output:

File: example.txt

Inode Number: 123456

Number of Hard Links: 1

File Permissions: -rw-r--r--

File Size: 1024 bytes

Last Access Time: Mon Jan 10 15:30:00 2023

Last Modification Time: Fri Dec 15 09:45:00 2023

Q2. Write a C program that catches the ctrl-c (SIGINT) signal for the first time and display the appropriate message and exits on pressing ctrl-c again.

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

int count = 0;

void handle_sigint(int signum) {
   if (signum == SIGINT) {
      if (count == 0) {
         printf("\nCaught SIGINT signal. Press Ctrl+C again to exit.\n");
         count++;
      } else {
         printf("\nReceived second SIGINT. Exiting...\n");
         exit(EXIT_SUCCESS);
      }
   }
}
```

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

    printf("Press Ctrl+C to trigger SIGINT signal.\n");

    while(1) {
        // Program continues running and waiting for SIGINT
    }

    return 0;
}
```

Slip 3

Q1. Print the type of file and inode number where file name accepted through Command Line.

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

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) {
    fprintf(stderr, "Error accessing file %s: %s\n", argv[1], strerror(errno));
    return 1;
}

printf("File: %s\n", argv[1]);
printf("Inode Number: %ld\n", (long)fileStat.st_ino);

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_ISLNK(fileStat.st_mode)) {
    printf("File Type: Symbolic Link\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_ISSOCK(fileStat.st_mode)) {
    printf("File Type: Socket\n");
} else {
    printf("File Type: Unknown\n");
}

return 0;
```

```
}
```

Output :

`example.txt` is a regular file:

File: example.txt

Inode Number: 123456

File Type: Regular file

`example.txt` is a directory file:

File: example.txt

Inode Number: 123456

File Type: Directory file

Q2. 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/types.h>
#include <sys/wait.h>

void child_handler(int signum) {
   if (signum == SIGCHLD) {
      printf("Child process has terminated.\n");
   }
}

void alarm_handler(int signum) {
   if (signum == SIGALRM) {
```

```
        printf("Child process exceeded time limit. Killing...\n");
        kill(getpid(), SIGKILL); // Kill the current process
    }
}

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

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

    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Child process
        execvp(argv[1], &argv[1]); // Execute the command
        perror("Execution failed");
        exit(EXIT_FAILURE);
    } else { // Parent process
        alarm(5); // Set alarm for 5 seconds

        // Wait for the child process to terminate
        wait(NULL);
```

```
    alarm(0); // Cancel the alarm
  }

  return 0;
}
```

Output :

Child process has terminated.

Slip 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 <stdlib.h>
#include <unistd.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++) {
    if (access(argv[i], F_OK) != -1) {
      printf("%s exists in the current directory.\n", argv[i]);
    } else {
      printf("%s does not exist in the current directory.\n", argv[i]);
    }
  }
```

```
    return 0;
}
```

Output :

./program_name file1.txt file2.txt

file1.txt exists in the 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!!!".

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

int kill_child = 0;

void child_handler(int signum) {
    if (signum == SIGHUP) {
        printf("Child received SIGHUP from parent.\n");
    } else if (signum == SIGINT) {
        printf("Child received SIGINT from parent.\n");
    } else if (signum == SIGQUIT) {
        printf("My Papa has Killed me!!!\n");
        exit(EXIT_SUCCESS);
    }
}

int main() {
```

```
pid_t pid = fork();

if (pid < 0) {
    perror("Fork failed");
    exit(EXIT_FAILURE);
} else if (pid == 0) { // Child process
    signal(SIGHUP, child_handler);
    signal(SIGINT, child_handler);
    signal(SIGQUIT, child_handler);

    while(1) {
        // Child process continues running and handling signals
    }
} else { // Parent process
    int counter = 0;

    while (counter < 5) { // Sending signals for 15 seconds (5 * 3 seconds)
        sleep(3);
        if (!kill_child) {
            kill(pid, SIGHUP); // Send SIGHUP signal
        } else {
            kill(pid, SIGINT); // Send SIGINT signal
        }
        counter++;
        if (counter == 5) {
            kill(pid, SIGQUIT); // Send SIGQUIT signal after 15 seconds
        }
        kill_child = !kill_child;
    }
```

```
        wait(NULL); // Wait for child to terminate
    }

    return 0;
}
```

Output :

```
Child received SIGHUP from parent.
Child received SIGINT from parent.
Child received SIGHUP from parent.
Child received SIGINT from parent.
Child received SIGHUP from parent.
Child received SIGINT from parent.
Child received SIGHUP from parent.
Child received SIGINT from parent.
Child received SIGHUP from parent.
My Papa has Killed me!!!
```

Slip 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() {
    DIR *directory;
    struct dirent *entry;
    int file_count = 0;

    // Open the current directory
```

```
    directory = opendir(".");

    if (directory == NULL) {
        perror("Unable to open directory");
        return EXIT_FAILURE;
    }

    // Read directory entries and display file names
    while ((entry = readdir(directory)) != NULL) {
        printf("%s\n", entry->d_name);
        file_count++;
    }

    closedir(directory);

    printf("\nTotal files in current directory: %d\n", file_count);

    return EXIT_SUCCESS;
}
```

Output :

file1.txt

file2.jpg

folder1

folder2

program.c

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>

#define MSG_SIZE 100

int main() {
    int pipefd[2];
    pid_t pid;
    char message[3][MSG_SIZE] = {
        "Hello World",
        "Hello SPPU",
        "Linux is Funny"
    };

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

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Child process
```

```
        close(pipefd[0]); // Close reading end of pipe in child

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

        close(pipefd[1]); // Close writing end of pipe in child
    } else { // Parent process
        close(pipefd[1]); // Close writing end of pipe in parent
        char buffer[MSG_SIZE];

        while (read(pipefd[0], buffer, MSG_SIZE) > 0) {
            printf("Message received: %s\n", buffer);
        }

        close(pipefd[0]); // Close reading end of pipe in parent
    }

    return 0;
}
```

Output :

```
Message received: Hello World
Message received: Hello SPPU
Message received: Linux is Funny
```

Slip 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>

int main() {
    DIR *directory;
    struct dirent *entry;
    struct stat fileStat;
    time_t now = time(NULL);
    struct tm *current_time = localtime(&now);
    int current_month = current_time->tm_mon + 1; // Month is zero-based in struct tm

    // Open the current directory
    directory = opendir(".");

    if (directory == NULL) {
        perror("Unable to open directory");
        return EXIT_FAILURE;
    }

    // Read directory entries and display files created in a specific month
    printf("Files created in current month (%d):\n", current_month);
    while ((entry = readdir(directory)) != NULL) {
        if (stat(entry->d_name, &fileStat) == 0) {
            struct tm *file_time = localtime(&fileStat.st_ctime);
            int file_month = file_time->tm_mon + 1; // Month is zero-based in struct tm

            if (file_month == current_month) {
```

```
            printf("%s\n", entry->d_name);
        }
      }
   }
   closedir(directory);

   return EXIT_SUCCESS;
}
```

Output :

Files created in current month (12):

file1.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.

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

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

   int num_children = atoi(argv[1]);
   struct rusage usage;
```

```
    int status;
    pid_t pid;

    for (int i = 0; i < num_children; i++) {
        pid = fork();

        if (pid < 0) {
            perror("Fork failed");
            return 1;
        } else if (pid == 0) { // Child process
            // Perform child's task
            sleep(1);
            exit(EXIT_SUCCESS);
        }
    }

    // Wait for all child processes to terminate
    while ((pid = wait(&status)) > 0);

    // Get resource usage statistics for all children
    if (getrusage(RUSAGE_CHILDREN, &usage) == -1) {
        perror("getrusage failed");
        return 1;
    }

    // Display total cumulative time spent by children in user and kernel mode
    printf("Total user time spent by children: %ld.%06ld seconds\n", usage.ru_utime.tv_sec, usage.ru_utime.tv_usec);
    printf("Total kernel time spent by children: %ld.%06ld seconds\n", usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);
```

```
    return 0;
}
```

Output :

Total user time spent by children: 0.000002 seconds

Total kernel time spent by children: 0.000000 seconds

Slip 7

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

```
#include <stdio.h>

int main() {
    // File pointer to store the redirected output
    FILE *file_ptr;

    // Open the file in write mode to redirect output
    file_ptr = freopen("output.txt", "w", stdout);

    if (file_ptr == NULL) {
        perror("Error opening file");
        return 1;
    }

    // Printing to stdout (redirected to file)
    printf("This output is redirected to a file.\n");

    // Closing the file pointer
    fclose(file_ptr);

    return 0;
```

```
}
```

Output :

This output is redirected to a file.

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/types.h>
#include <sys/wait.h>

int main() {
    int pipe_fd[2];

    if (pipe(pipe_fd) == -1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Child process
        close(pipe_fd[0]); // Close reading end of pipe in child
        dup2(pipe_fd[1], STDOUT_FILENO); // Redirect stdout to pipe write end
```

```
        // Execute ls -l command
        execlp("ls", "ls", "-l", NULL);

        // execlp will not return unless there's an error
        perror("exec ls -l failed");
        exit(EXIT_FAILURE);
    } else { // Parent process
        close(pipe_fd[1]); // Close writing end of pipe in parent
        dup2(pipe_fd[0], STDIN_FILENO); // Redirect stdin to pipe read end

        // Execute wc -l command
        execlp("wc", "wc", "-l", NULL);

        // execlp will not return unless there's an error
        perror("exec wc -l failed");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

Output :

7

Slip 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 <stdlib.h>
#include <fcntl.h>
```

```
#include <unistd.h>

int main() {
    int file_descriptor;
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH; // File permissions

    // Open file for writing (creating if it doesn't exist)
    file_descriptor = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, mode);

    if (file_descriptor < 0) {
        perror("Error opening file");
        return 1;
    }

    // Redirect standard output to the file
    if (dup2(file_descriptor, STDOUT_FILENO) == -1) {
        perror("Error redirecting output");
        return 1;
    }

    // Close the file descriptor
    close(file_descriptor);

    // Printing to stdout (redirected to file)
    printf("This output is redirected to a file using dup and open system calls.\n");

    return 0;
}
```

Output :

This output is redirected to a file using dup and open system calls.

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

Same As Slip 7

Slip 9

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

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

#define MSG_SIZE 100

int main() {
    int pipe_fd[2];
    pid_t pid;
    char message[MSG_SIZE] = "Hello from parent!";

    if (pipe(pipe_fd) == -1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
```

```
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Child process
        close(pipe_fd[1]); // Close writing end of pipe in child

        char buffer[MSG_SIZE];
        read(pipe_fd[0], buffer, MSG_SIZE);
        printf("Child received: %s\n", buffer);

        close(pipe_fd[0]); // Close reading end of pipe in child
    } else { // Parent process
        close(pipe_fd[0]); // Close reading end of pipe in parent

        write(pipe_fd[1], message, sizeof(message));

        close(pipe_fd[1]); // Close writing end of pipe in parent
    }

    return 0;
}
```

Output :

```
Child received: Hello from parent!
```

Slip 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/types.h>
#include <sys/wait.h>
```

```
int main() {
    int pipe_fd[2];

    if (pipe(pipe_fd) == -1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Child process 1 (writes to pipe)
        close(pipe_fd[0]); // Close reading end of pipe in child 1
        dup2(pipe_fd[1], STDOUT_FILENO); // Redirect stdout to pipe write end

        // Execute command 1 (e.g., ls -l)
        execlp("ls", "ls", "-l", NULL);

        // execlp will not return unless there's an error
        perror("exec command 1 failed");
        exit(EXIT_FAILURE);
    } else { // Parent process (reads from pipe)
        close(pipe_fd[1]); // Close writing end of pipe in parent

        pid_t pid2 = fork();
```

```
        if (pid2 < 0) {
            perror("Fork failed");
            exit(EXIT_FAILURE);
        } else if (pid2 == 0) { // Child process 2 (reads from pipe)
            dup2(pipe_fd[0], STDIN_FILENO); // Redirect stdin to pipe read end

            // Execute command 2 (e.g., wc -l)
            execlp("wc", "wc", "-l", NULL);

            // execlp will not return unless there's an error
            perror("exec command 2 failed");
            exit(EXIT_FAILURE);
        } else { // Parent process
            close(pipe_fd[0]); // Close reading end of pipe in parent

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

    return 0;
}
```

Output :

```
total 12
drwxr-xr-x  3 user user 4096 Jun  3 13:58 directory1
-rw-r--r--  1 user user  164 Jun  3 13:59 file1.txt
drwxr-xr-x  2 user user 4096 Jun  3 13:58 directory2
```

Slip 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>

int main() {
   struct rlimit limit;

   // Get current resource limits for the process (for example, files limit)
   if (getrlimit(RLIMIT_NOFILE, &limit) == -1) {
      perror("getrlimit failed");
      return EXIT_FAILURE;
   }

   printf("Current files limit: soft=%ld, hard=%ld\n", limit.rlim_cur, limit.rlim_max);

   // Modify the resource limit (for example, increase files limit)
   limit.rlim_cur = 1000; // Change the soft limit
   limit.rlim_max = 1500; // Change the hard limit

   if (setrlimit(RLIMIT_NOFILE, &limit) == -1) {
      perror("setrlimit failed");
      return EXIT_FAILURE;
   }

   printf("New files limit: soft=%ld, hard=%ld\n", limit.rlim_cur, limit.rlim_max);
```

```
    return EXIT_SUCCESS;
}
```

Output :

```
Current files limit: soft=1024, hard=4096
New files limit: soft=1000, hard=1500
```

Slip 12

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

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

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

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Child process
        // Perform some task in the child process
        printf("Child process executing...\n");
        exit(42); // Child process exits with status 42
    } else { // Parent process
        int status;
```

```
    waitpid(pid, &status, 0); // Wait for the child process to terminate

    if (WIFEXITED(status)) {
        printf("Child process exited normally with status: %d\n", WEXITSTATUS(status));
    } else {
        printf("Child process exited abnormally\n");
    }
  }

  return 0;
}
```

Output :

Child process executing...

Child process exited normally with status: 42

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

#define MAX_FILES 50
#define MAX_FILENAME_LENGTH 100

// Structure to hold file information
struct FileInfo {
    char filename[MAX_FILENAME_LENGTH];
```

```
    off_t size;
};

// Comparator function for qsort
int compare(const void *a, const void *b) {
    return ((struct FileInfo *)a)->size - ((struct FileInfo *)b)->size;
}

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

    struct FileInfo files[MAX_FILES];
    int num_files = argc - 1;

    if (num_files > MAX_FILES) {
        printf("Exceeded maximum number of files (%d)\n", MAX_FILES);
        return 1;
    }

    // Get file sizes
    for (int i = 0; i < num_files; ++i) {
        struct stat fileStat;

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

```
        strncpy(files[i].filename, argv[i + 1], MAX_FILENAME_LENGTH - 1);
        files[i].filename[MAX_FILENAME_LENGTH - 1] = '\0';
        files[i].size = fileStat.st_size;
    }

    // Sort files based on size
    qsort(files, num_files, sizeof(struct FileInfo), compare);

    // Display filenames in ascending order of their sizes
    printf("Files sorted by size (ascending order):\n");
    for (int i = 0; i < num_files; ++i) {
        printf("%s - %lld bytes\n", files[i].filename, (long long)files[i].size);
    }

    return 0;
}
```

Output :

```
Files sorted by size (ascending order):
file3.txt - 500 bytes
file1.txt - 1000 bytes
file2.txt - 2000 bytes
```

Slip 13

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

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

```
void sig_handler(int signo) {
    if (signo == SIGUSR1) {
        printf("Received SIGUSR1 - Suspending the process\n");
        raise(SIGSTOP); // Suspend the process
    } else if (signo == SIGUSR2) {
        printf("Received SIGUSR2 - Resuming the process\n");
    }
}

int main() {
    if (signal(SIGUSR1, sig_handler) == SIG_ERR || signal(SIGUSR2, sig_handler) ==
SIG_ERR) {
        perror("Signal setup failed");
        return EXIT_FAILURE;
    }

    printf("Waiting for signals...\n");

    while (1) {
        sleep(1); // Continuously wait for signals
    }

    return 0;
}
```

Output :

Waiting for signals...

Received SIGUSR1 - Suspending the process.

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>


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

   DIR *dir;
   struct dirent *entry;

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

   printf("Files starting with '%s':\n", argv[1]);
   while ((entry = readdir(dir)) != NULL) {
      if (strncmp(entry->d_name, argv[1], strlen(argv[1])) == 0) {
         printf("%s\n", entry->d_name);
      }
   }

   closedir(dir);
```

```
    return 0;
}
```

Output :

```
Files starting with 'foo':
foo1.txt
foo2.txt
foobar.txt
```

Slip 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 <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>

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

    DIR *dir;
    struct dirent *entry;
    struct stat fileStat;

    if ((dir = opendir(".")) == NULL) {
        perror("opendir() failed");
```

```
        return 1;
    }

    printf("Files with size greater than %ld bytes:\n", n);
    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &fileStat) == -1) {
            perror("stat");
            return 1;
        }

        if (S_ISREG(fileStat.st_mode) && fileStat.st_size > n) {
            printf("%s\n", entry->d_name);
        }
    }

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

Output :

```
Enter the size in bytes: 100
Files with size greater than 100 bytes:
file2.txt
file4.txt
```

Slip 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 <sys/types.h>
```

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

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

    DIR *dir;
    struct dirent *entry;
    struct stat fileStat;

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

    printf("Files with size greater than %ld bytes:\n", n);
    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &fileStat) == -1) {
            perror("stat");
            return 1;
        }

        if (S_ISREG(fileStat.st_mode) && fileStat.st_size > n) {
            printf("%s\n", entry->d_name);
        }
    }

    closedir(dir);
```

```
    return 0;
}
```

Output :

```
Enter the size in bytes: 1000
Files with size greater than 1000 bytes:
file2.txt
file4.txt
```

Note : All Other Remaining Slips Ques Are Same As Above Slips Ques.
