SLIP 1

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>


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

   struct stat file_stat;

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

      if (stat(argv[i], &file_stat) == -1) {

         perror("stat");

         continue;

      }

      printf("File: %s, Inode: %ld, ", argv[i], file_stat.st_ino);

      if (S_ISREG(file_stat.st_mode)) printf("Regular file\n");

      else if (S_ISDIR(file_stat.st_mode)) printf("Directory\n");

      else if (S_ISCHR(file_stat.st_mode)) printf("Character device\n");

      else if (S_ISBLK(file_stat.st_mode)) printf("Block device\n");

      else if (S_ISFIFO(file_stat.st_mode)) printf("FIFO\n");

      else if (S_ISLNK(file_stat.st_mode)) printf("Symbolic link\n");

      else if (S_ISSOCK(file_stat.st_mode)) printf("Socket\n");

      else printf("Unknown file type\n");

   }

   return 0;

}
```

Q.2) Write a C program to send SIGALRM signal by child process to parent process and parent process make a provision to catch the signal and display alarm is fired.(Use Kill, fork, signal and sleep system call) [20 Marks]

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>
```

```
#include <signal.h>
#include <sys/wait.h>

void handle_sigalrm(int sig) {
    printf("Alarm is fired by the child process\n");
}

int main() {
    pid_t pid = fork();
    if (pid == 0) {
        sleep(2);
        kill(getppid(), SIGALRM);
        exit(0);
    } else {
        signal(SIGALRM, handle_sigalrm);
        pause();
        wait(NULL);
    }
    return 0;
}
```

SLIP 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 <sys/stat.h>

#include <time.h>


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

  struct stat file_stat;

  if (stat(argv[1], &file_stat) != 0) {

    perror("stat");

    return 1;

  }

  printf("Inode: %ld\n", file_stat.st_ino);

  printf("Links: %ld\n", file_stat.st_nlink);

  printf("Permissions: %o\n", file_stat.st_mode & 0777);

  printf("Size: %ld bytes\n", file_stat.st_size);

  printf("Access time: %s", ctime(&file_stat.st_atime));

  printf("Modification time: %s", ctime(&file_stat.st_mtime));

  return 0;

}
```

Q.2) Write a C program that catches the ctrl-c (SIGINT) signal for the first time and display the appropriate message and exits on pressing ctrl-c again. [20 Marks]

```
#include <stdio.h>

#include <signal.h>

#include <stdlib.h>


int sigint_count = 0;


void handle_sigint(int sig) {
    sigint_count++;
    if (sigint_count == 1)
        printf("\nCaught SIGINT (Ctrl-C), press again to exit.\n");
    else {
        printf("\nCaught SIGINT again, exiting...\n");
        exit(0);
    }
}

int main() {
    signal(SIGINT, handle_sigint);
    while (1) {
        printf("Running... Press Ctrl-C\n");
        sleep(1);
    }
    return 0;
}
```

SLIP 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>


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

  struct stat file_stat;

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

    if (stat(argv[i], &file_stat) == -1) {

      perror("stat");

      continue;

    }

    printf("File: %s, Inode: %ld, ", argv[i], file_stat.st_ino);

    if (S_ISREG(file_stat.st_mode)) printf("Regular file\n");

    else if (S_ISDIR(file_stat.st_mode)) printf("Directory\n");

    else if (S_ISCHR(file_stat.st_mode)) printf("Character device\n");

    else if (S_ISBLK(file_stat.st_mode)) printf("Block device\n");

    else if (S_ISFIFO(file_stat.st_mode)) printf("FIFO\n");

    else if (S_ISLNK(file_stat.st_mode)) printf("Symbolic link\n");

    else if (S_ISSOCK(file_stat.st_mode)) printf("Socket\n");

    else printf("Unknown file type\n");

  }

  return 0;

}
```

Q.2) Write a C program which creates a child process to run linux/ unix command or any user defined program. The parent process set the signal handler for death of child signal and Alarm signal. If a child process does not complete its execution in 5 second then parent process kills child process. [20 Marks]

```
#include <stdio.h>

#include <unistd.h>
```

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

pid_t child_pid;

void handle_alarm(int sig) {
    printf("Child did not finish in 5 seconds, killing child\n");
    kill(child_pid, SIGKILL);
}

void handle_child_death(int sig) {
    int status;
    waitpid(child_pid, &status, 0);
    if (WIFEXITED(status))
        printf("Child finished normally with status %d\n", WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        printf("Child killed by signal %d\n", WTERMSIG(status));
}

int main() {
    signal(SIGALRM, handle_alarm);
    signal(SIGCHLD, handle_child_death);

    child_pid = fork();
    if (child_pid == 0) {
        execlp("sleep", "sleep", "10", NULL); // Simulate long task
        exit(0);
    } else {
        alarm(5);
        pause();
    }
    return 0;
}
```

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. [10 Marks]

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

int main(int argc, char *argv[]) {
    DIR *dir = opendir(".");
    struct dirent *entry;
    for (int i = 1; i < argc; i++) {
        int found = 0;
        rewinddir(dir);
        while ((entry = readdir(dir)) != NULL) {
            if (strcmp(entry->d_name, argv[i]) == 0) {
                found = 1;
                break;
            }
        }
        printf("File '%s' %s\n", argv[i], found ? "is present" : "is not found");
    }
    closedir(dir);
    return 0;
}
```

Q.2) Write a C program which creates a child process and child process catches a signal SIGHUP, SIGINT and SIGQUIT. The Parent process send a SIGHUP or SIGINT signal after every 3 seconds, at the end of 15 second parent send SIGQUIT signal to child and child terminates by displaying message “My Papa has Killed me!!!”. [20 Marks]

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

void handle_signal(int sig) {
  if (sig == SIGHUP)
    printf("Child received SIGHUP signal\n");
  else if (sig == SIGINT)
    printf("Child received SIGINT signal\n");
  else if (sig == SIGQUIT) {
    printf("My Papa has Killed me !!!\n");
    exit(0);
  }
}

int main() {
  pid_t pid = fork();
  if (pid == 0) {
    signal(SIGHUP, handle_signal);
    signal(SIGINT, handle_signal);
    signal(SIGQUIT, handle_signal);
    while (1) pause();
  } else {
    sleep(1);
    for (int i = 0; i < 5; i++) {
      kill(pid, SIGHUP);
      sleep(3);
```

```
            kill(pid, SIGINT);
            sleep(3);
        }
        kill(pid, SIGQUIT);
        wait(NULL);
    }
    return 0;
}
```

SLIP 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 file_count = 0;


   dir = opendir(".");

   if (dir == NULL) {

      perror("opendir");

      return 1;

   }


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

   while ((entry = readdir(dir)) != NULL) {

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

      file_count++;

   }


   closedir(dir);

   printf("Total number of files: %d\n", file_count);

   return 0;

}
```

Q.2) Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it.

Message1 = “Hello World”

Message2 = “Hello SPPU”

Message3 = “Linux is Funny”

[20 Marks]

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

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

    pipe(fd);
    pid = fork();

    if (pid == 0) {
        close(fd[0]);
        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);
            sleep(1);
        }
        close(fd[1]);
    } else {
        close(fd[1]);
        while (read(fd[0], buffer, sizeof(buffer)) > 0) {
            printf("Parent received: %s\n", buffer);
        }
        close(fd[0]);
        wait(NULL);
    }
    return 0;
}
```

SLIP 6

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

```
#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#include <time.h>


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

    if (argc != 2) {

        printf("Usage: %s <month number>\n", argv[0]);

        return 1;

    }


    int month = atoi(argv[1]);

    DIR *dr = opendir(".");

    struct dirent *entry;

    struct stat st;


    while ((entry = readdir(dr)) != NULL) {

        stat(entry->d_name, &st);

        struct tm *timeinfo = localtime(&st.st_ctime);

        if (timeinfo->tm_mon + 1 == month) {

            printf("%s\t%ld\t%s", entry->d_name, st.st_size, ctime(&st.st_ctime));

        }

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

Q.2) Write a C program to create n child processes. When all n child processes terminates, Display total cumulative time children spent in user and kernel mode [20 Marks]

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

int main() {
    int n;
    printf("Enter the number of child processes: ");
    scanf("%d", &n);

    pid_t pid;
    struct rusage usage;
    int status;

    for (int i = 0; i < n; i++) {
        pid = fork();
        if (pid == 0) {
            sleep(2);
            exit(0);
        }
    }

    for (int i = 0; i < n; i++) {
        wait(&status);
    }

    if (getrusage(RUSAGE_CHILDREN, &usage) == 0) {
        printf("Total user time: %ld.%06ld seconds\n", usage.ru_utime.tv_sec, usage.ru_utime.tv_usec);
        printf("Total system time: %ld.%06ld seconds\n", usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);
    } else {
        perror("getrusage");
    }

    return 0;
}
```

SLIP 7

Q.1) Write a C Program that demonstrates redirection of standard output to a file [10 Marks]

```
#include <stdio.h>


int main() {
  FILE *file = freopen("output.txt", "w", stdout);
  if (file == NULL) {
    perror("freopen");
    return 1;
  }


  printf("This text will be written to output.txt\n");
  fclose(file);
  return 0;
}
```

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


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


  pid_t pid1 = fork();
  if (pid1 == 0) {
    close(pipefd[0]);
```

```
        dup2(pipefd[1], STDOUT_FILENO);
        execlp("ls", "ls", "-l", NULL);
        exit(1);
    }

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

    close(pipefd[0]);
    close(pipefd[1]);
    wait(NULL);
    wait(NULL);

    return 0;
}
```

SLIP 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 <fcntl.h>
#include <unistd.h>

int main() {
   int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
   int saved_stdout = dup(1);
   dup2(fd, 1);

   printf("This will be written to output.txt\n");
   fflush(stdout);
   dup2(saved_stdout, 1);
   close(fd);

   return 0;
}
```

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

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

   pid_t pid1 = fork();
```

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

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

    close(pipefd[0]);
    close(pipefd[1]);
    wait(NULL);
    wait(NULL);

    return 0;
}
```

SLIP 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];

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

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

  if (pid == 0) { // Child process
    close(fd[1]);
    char buffer[100];
    read(fd[0], buffer, sizeof(buffer));
    printf("Child received: %s\n", buffer);
  } else { // Parent process
    close(fd[0]);
    char message[] = "Hello from parent!";
    write(fd[1], message, strlen(message) + 1);
    wait(NULL);
  }
```

```
   return 0;
}
```

Q.2) Write a C program to Identify the type (Directory, character device, Block device, Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call. [20 Marks]

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

void check_file_type(struct stat file_stat) {
   if (S_ISREG(file_stat.st_mode)) printf("Regular file\n");
   else if (S_ISDIR(file_stat.st_mode)) printf("Directory\n");
   else if (S_ISCHR(file_stat.st_mode)) printf("Character device\n");
   else if (S_ISBLK(file_stat.st_mode)) printf("Block device\n");
   else if (S_ISFIFO(file_stat.st_mode)) printf("FIFO or pipe\n");
   else if (S_ISLNK(file_stat.st_mode)) printf("Symbolic link\n");
   else if (S_ISSOCK(file_stat.st_mode)) printf("Socket\n");
   else printf("Unknown type\n");
}

int main(int argc, char *argv[]) {
   struct stat file_stat;
   if (stat(argv[1], &file_stat) != 0) {
      perror("stat");
      return 1;
   }
   check_file_type(file_stat);
   return 0;
}
```

SLIP 10

Q.1) Write a program that illustrates how to execute two commands concurrently with a pipe. [10 Marks]

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>


int main() {

  int pipefd[2];

  pid_t pid1, pid2;


  if (pipe(pipefd) == -1) {

    perror("pipe");

    return 1;

  }


  pid1 = fork();

  if (pid1 == -1) {

    perror("fork");

    return 1;

  }


  if (pid1 == 0) {

    close(pipefd[0]);

    dup2(pipefd[1], STDOUT_FILENO);

    execlp("ls", "ls", NULL);

    perror("execlp ls failed");

    exit(EXIT_FAILURE);

  }
```

```
    pid2 = fork();
    if (pid2 == -1) {
        perror("fork");
        return 1;
    }

    if (pid2 == 0) {
        close(pipefd[1]);
        dup2(pipefd[0], STDIN_FILENO);
        execlp("wc", "wc", NULL);
        perror("execlp wc failed");
        exit(EXIT_FAILURE);
    }

    close(pipefd[0]);
    close(pipefd[1]);
    wait(NULL);
    wait(NULL);

    return 0;
}
```

Q.2) Generate parent process to write unnamed pipe and will write into it. Also generate child process which will read from pipe [20 Marks]

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

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

    pipe(fd);
    pid = fork();

    if (pid == 0) {
        close(fd[0]);
        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);
            sleep(1);
        }
        close(fd[1]);
    } else {
        close(fd[1]);
        while (read(fd[0], buffer, sizeof(buffer)) > 0) {
            printf("Parent received: %s\n", buffer);
        }
        close(fd[0]);
        wait(NULL);
    }

    return 0;
}
```

SLIP 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 <sys/resource.h>


int main() {

  struct rlimit file_limit;


  if (getrlimit(RLIMIT_NOFILE, &file_limit) == 0) {

    printf("Current file limit: soft = %ld, hard = %ld\n",

        file_limit.rlim_cur,

        file_limit.rlim_max);

  } else {

    perror("getrlimit");

  }


  file_limit.rlim_cur = 2048;

  file_limit.rlim_max = 4096;


  if (setrlimit(RLIMIT_NOFILE, &file_limit) == 0) {

    printf("New file limit: soft = %ld, hard = %ld\n",

        file_limit.rlim_cur,

        file_limit.rlim_max);

  } else {

    perror("setrlimit");

  }


  return 0;

}
```

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>


int main() {
  FILE *file = freopen("output.txt", "w", stdout);


  if (file == NULL) {
    perror("freopen");
    return 1;
  }


  printf("This text will be written to output.txt\n");


  fclose(file);


  return 0;
}
```

SLIP 12

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

```
#include <stdio.h>

#include <sys/wait.h>

#include <unistd.h>


int main() {

  pid_t pid = fork();


  if (pid == 0) {

    printf("Child process (PID: %d) is terminating...\n", getpid());

    exit(42);

  } else {

    int status;

    wait(&status);


    if (WIFEXITED(status)) {

      printf("Child exited with status %d\n", WEXITSTATUS(status));

    }

  }


  return 0;

}
```

Q.2) Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes. I) (e.g $ a.out a.txt b.txt c.txt, ...) [20 Marks]

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>


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

    int month = atoi(argv[1]);
    DIR *dr = opendir(".");
    if (dr == NULL) {
        perror("opendir");
        return 1;
    }

    struct dirent *entry;
    struct stat st;
    while ((entry = readdir(dr)) != NULL) {
        stat(entry->d_name, &st);
        struct tm *timeinfo = localtime(&st.st_ctime);
        if (timeinfo->tm_mon + 1 == month) {
            printf("%s\t%ld\t%s", entry->d_name, st.st_size, ctime(&st.st_ctime));
        }
    }

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

SLIP 13

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

```
#include <stdio.h>

#include <signal.h>

#include <unistd.h>

#include <stdlib.h>


void handle_sigcont(int sig) {

    printf("Process resumed (SIGCONT received)\n");

}


int main() {

    pid_t pid = fork();


    if (pid == 0) {

        signal(SIGCONT, handle_sigcont);

        printf("Child process running (pid: %d). Send SIGSTOP to suspend.\n", getpid());

        while (1) sleep(1);

    } else {

        printf("Parent: Suspending child process (PID: %d)\n", pid);

        kill(pid, SIGSTOP);

        sleep(3);

        printf("Parent: Resuming child process (PID: %d)\n", pid);

        kill(pid, SIGCONT);

        wait(NULL);

    }


    return 0;

}
```

Q.2) Write a C program that a string as an argument and return all the files that begins with that name in the current directory. For example > ./a.out foo will return all file names that begins with foo [20 Marks]

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

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

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

    struct dirent *entry;
    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;
}
```

SLIP 14

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 <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#include <string.h>


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

  struct dirent *entry;

  struct stat file_stat;

  DIR *dir;

  long size_threshold;


  if (argc != 2) {

    printf("Usage: %s <size in bytes>\n", argv[0]);

    return 1;

  }


  size_threshold = atol(argv[1]);


  dir = opendir(".");

  if (dir == NULL) {

    perror("opendir");

    return 1;

  }


  while ((entry = readdir(dir)) != NULL) {

    if (stat(entry->d_name, &file_stat) == 0) {

      if (S_ISREG(file_stat.st_mode) &&
        file_stat.st_size > size_threshold) {
        printf("%s (%ld bytes)\n", entry->d_name, file_stat.st_size);
      }
    } else {
      perror("stat");
    }
  }

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

Q.2) Write a C program to find file properties such as inode number, number of hard link, File permissions, File size, File access and modification time and so on of a given file using stat() system call. [20 Marks]

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

int main(int argc, char *argv[]) {
  struct stat file_stat;
  if (stat(argv[1], &file_stat) == -1) {
    perror("stat");
    return 1;
  }
  printf("File: %s\n", argv[1]);
  printf("Inode: %ld\n", file_stat.st_ino);
  printf("Size: %ld bytes\n", file_stat.st_size);
  printf("Links: %ld\n", file_stat.st_nlink);
  printf("Permissions: %o\n", file_stat.st_mode & 0777);
  printf("File type: %d\n", file_stat.st_mode & S_IFMT);
  return 0;
}
```

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

int main(int argc, char *argv[]) {
  struct dirent *entry;
  struct stat file_stat;
  DIR *dir;
  long size_threshold;

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

  size_threshold = atol(argv[1]);

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

  while ((entry = readdir(dir)) != NULL) {
    if (stat(entry->d_name, &file_stat) == 0) {
      if (S_ISREG(file_stat.st_mode) &&
            file_stat.st_size > size_threshold) {
            printf("%s (%ld bytes)\n", entry->d_name, file_stat.st_size);
         }
      } else {
         perror("stat");
      }
   }

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

Q.2) Write a C program which creates a child process to run linux/ unix command or any user defined program. The parent process set the signal handler for death of child signal and Alarm signal. If a child process does not complete its execution in 5 second then parent process kills child process [20 Marks]

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

pid_t child_pid;

void handle_alarm(int sig) {
   printf("Child did not finish in 5 seconds, killing child\n");
   kill(child_pid, SIGKILL);
}

void handle_child_death(int sig) {
   int status;
   waitpid(child_pid, &status, 0);
```

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

int main() {
    signal(SIGALRM, handle_alarm);      // Set signal handler for alarm
    signal(SIGCHLD, handle_child_death); // Set signal handler for child termination

    child_pid = fork();

    if (child_pid == 0) { // Child process
        printf("Child process running\n");
        sleep(10); // Simulate a long-running task
        printf("Child process finished\n");
        exit(0);
    } else { // Parent process
        alarm(5);  // Set alarm for 5 seconds
        pause();   // Wait for either alarm or child termination
    }

    return 0;
}
```

SLIP 16

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

```
#include <stdio.h>

#include <dirent.h>


int main() {

   DIR *dir;

   struct dirent *entry;

   int file_count = 0;


   dir = opendir(".");

   if (dir == NULL) {

      perror("opendir");

      return 1;

   }


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

   while ((entry = readdir(dir)) != NULL) {

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

      file_count++;

   }


   closedir(dir);

   printf("Total number of files: %d\n", file_count);

   return 0;

}
```

Q.2) Write a C program which create a child process which catch a signal sighup, sigint and sigquit. The Parent process send a sighup or sigint signal after every 3 seconds, at the end of 30 second parent send sigquit signal to child and child terminates my displaying message "My DADDY has Killed me!!!". [20 Marks]

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

```
#include <unistd.h>

#include <stdlib.h>

#include <signal.h>


void handle_signal(int sig) {

  if (sig == SIGHUP) {

    printf("Child received SIGHUP signal\n");

  } else if (sig == SIGINT) {

    printf("Child received SIGINT signal\n");

  } else if (sig == SIGQUIT) {

    printf("My DADDY has Killed me!!!\n");

    exit(0);

  }

}


int main() {

  pid_t pid = fork();


  if (pid == 0) { // Child process

    signal(SIGHUP, handle_signal);

    signal(SIGINT, handle_signal);

    signal(SIGQUIT, handle_signal);


    while (1) {

      pause(); // Wait for signals

    }

  } else { // Parent process

    sleep(1); // Allow child to set up signal handlers


    for (int i = 0; i < 10; i++) {

      if (i % 2 == 0) {
```

```
                kill(pid, SIGHUP);
            } else {
                kill(pid, SIGINT);
            }
            sleep(3);
        }

        // After 30 seconds, send SIGQUIT
        kill(pid, SIGQUIT);

        wait(NULL); // Wait for the child to terminate
        return 0;
    }
}
```

SLIP 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 file_count = 0;


   dir = opendir(".");

   if (dir == NULL) {

      perror("opendir");

      return 1;

   }


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

   while ((entry = readdir(dir)) != NULL) {

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

      file_count++;

   }


   closedir(dir);

   printf("Total number of files: %d\n", file_count);

   return 0;

}
```

Q.2) Write a C program to implement the following unix/linux command (use fork, pipe and exec system call). Your program should block the signal Ctrl-C and Ctrl-\ signal during the execution. i. Ls –l | wc –l [20 Marks]

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

```
#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>


void block_signals() {

  sigset_t set;

  sigemptyset(&set);

  sigaddset(&set, SIGINT);  // Block Ctrl-C

  sigaddset(&set, SIGQUIT); // Block Ctrl-\

  sigprocmask(SIG_BLOCK, &set, NULL);

}


int main() {

  int pipe_fd[2];

  pid_t pid1, pid2;


  pipe(pipe_fd);


  pid1 = fork();

  if (pid1 == 0) { // First child: executes 'ls -l'

    close(pipe_fd[0]); // Close read end

    dup2(pipe_fd[1], STDOUT_FILENO); // Redirect stdout to the pipe

    execlp("ls", "ls", "-l", NULL);

    perror("execlp ls failed");

    exit(EXIT_FAILURE);

  }


  pid2 = fork();

  if (pid2 == 0) { // Second child: executes 'wc -l'

    close(pipe_fd[1]); // Close write end
```

```
        dup2(pipe_fd[0], STDIN_FILENO); // Redirect stdin from the pipe
        execlp("wc", "wc", "-l", NULL);
        perror("execlp wc failed");
        exit(EXIT_FAILURE);
    }

    // Parent process
    block_signals(); // Block Ctrl-C and Ctrl-\ signals
    close(pipe_fd[0]);
    close(pipe_fd[1]);

    wait(NULL); // Wait for both children to finish
    wait(NULL);

    return 0;
}
```

SLIP 18

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

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

#include <sys/stat.h>

#include <unistd.h>

#include <stdio.h>


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

  struct stat file_stat;

  if (stat(argv[1], &file_stat) == -1) {

    perror("stat");

    return(1);

  }


  if (S_ISREG(file_stat.st_mode)) printf("Regular file\n");

  else if (S_ISDIR(file_stat.st_mode)) printf("Directory\n");

  else if (S_ISCHR(file_stat.st_mode)) printf("Character device\n");

  else if (S_ISBLK(file_stat.st_mode)) printf("Block device\n");

  else if (S_ISFIFO(file_stat.st_mode)) printf("FIFO\n");

  else if (S_ISLNK(file_stat.st_mode)) printf("Symbolic link\n");

  else if (S_ISSOCK(file_stat.st_mode)) printf("Socket\n");

  else printf("Unknown file type\n");


  return(0);

}
```

Q.2) Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it.

Message1 = “Hello World”

Message2 = “Hello SPPU”

Message3 = “Linux is Funny”

[20 Marks]

```
#include <stdio.h>
#include <unistd.h>
#include <string.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 process
        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);
            sleep(1);
        }

        close(fd[1]);
    } else { // Parent process
        close(fd[1]); // Close write end

        while (read(fd[0], buffer, sizeof(buffer)) > 0) {
            printf("Parent received: %s\n", buffer);
        }

        close(fd[0]);
        wait(NULL);
    }

    return 0;
}
```

SLIP 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>


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

  struct stat file_stat;

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

    if (stat(argv[i], &file_stat) == -1) {

      perror("stat");

      return 1;

    }

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

  }

  return 0;

}
```

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 <stdlib.h>

#include <unistd.h>


int main() {

  int pipefd[2];

  pid_t pid1, pid2;


  if (pipe(pipefd) == -1) {

    perror("pipe");

    return 1;

  }
```

```
pid1 = fork();
if (pid1 == -1) {
    perror("fork");
    return 1;
}

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

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

if (pid2 == 0) {
    close(pipefd[1]);
    dup2(pipefd[0], STDIN_FILENO);
    close(pipefd[0]);
    execlp("wc", "wc", "-l", NULL);
    perror("execlp wc");
    exit(1);
}
```

```
    close(pipefd[0]);
    close(pipefd[1]);
    wait(NULL);
    wait(NULL);

    return 0;
}
```

SLIP 20

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

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

void handle_sigcont(int sig) {
    printf("Process resumed (SIGCONT received)\n");
}

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

    if (pid == 0) {
        signal(SIGCONT, handle_sigcont);
        printf("Child process running (pid: %d). Send SIGSTOP to suspend.\n", getpid());
        while (1) sleep(1);
    } else {
        printf("Parent: Suspending child process (PID: %d)\n", pid);
        kill(pid, SIGSTOP);
        sleep(3);
        printf("Parent: Resuming child process (PID: %d)\n", pid);
        kill(pid, SIGCONT);
        wait(NULL);
    }

    return 0;
}
```

Q.2) Write a C program to Identify the type (Directory, character device, Block device, Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call. [20 Marks]

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

void check_file_type(struct stat file_stat) {
  if (S_ISREG(file_stat.st_mode)) printf("Regular file\n");
  else if (S_ISDIR(file_stat.st_mode)) printf("Directory\n");
  else if (S_ISCHR(file_stat.st_mode)) printf("Character device\n");
  else if (S_ISBLK(file_stat.st_mode)) printf("Block device\n");
  else if (S_ISFIFO(file_stat.st_mode)) printf("FIFO or pipe\n");
  else if (S_ISLNK(file_stat.st_mode)) printf("Symbolic link\n");
  else if (S_ISSOCK(file_stat.st_mode)) printf("Socket\n");
  else printf("Unknown type\n");
}

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

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

  printf("File type of %s: ", argv[1]);
  check_file_type(file_stat);
  return 0;
}
```

SLIP 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>


int main() {

  DIR *dir;

  struct dirent *entry;

  int file_count = 0;


  dir = opendir(".");

  if (dir == NULL) {

    perror("opendir");

    return 1;

  }


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

  while ((entry = readdir(dir)) != NULL) {

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

    file_count++;

  }


  closedir(dir);

  printf("Total number of files: %d\n", file_count);

  return 0;

}
```

Q.2) Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes. I) (e.g $ a.out a.txt b.txt c.txt, ...) [20 Marks]

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

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

    int month = atoi(argv[1]);
    DIR *dr = opendir(".");
    if (dr == NULL) {
        perror("opendir");
        return 1;
    }

    struct dirent *entry;
    struct stat st;
    while ((entry = readdir(dr)) != NULL) {
        stat(entry->d_name, &st);
        struct tm *timeinfo = localtime(&st.st_ctime);
        if (timeinfo->tm_mon + 1 == month) {
            printf("%s\t%ld\t%s", entry->d_name, st.st_size, ctime(&st.st_ctime));
        }
    }

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

SLIP 22

Q.1) Write a C Program that demonstrates redirection of standard output to a file [10 Marks]

```
#include <stdio.h>


int main() {
   FILE *file = freopen("output.txt", "w", stdout);


   if (file == NULL) {
      perror("freopen");
      return 1;
   }


   printf("This text will be written to output.txt\n");


   fclose(file);


   return 0;
}
```

Q.2) Write a C program to implement the following unix/linux command (use fork, pipe and exec system call). Your program should block the signal Ctrl-C and Ctrl-\ signal during the execution. i. ls –l | wc –l [20 Marks]

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


void block_signals() {
   sigset_t set;
```

```
    sigemptyset(&set);
    sigaddset(&set, SIGINT);  // Block Ctrl-C
    sigaddset(&set, SIGQUIT); // Block Ctrl-\
    sigprocmask(SIG_BLOCK, &set, NULL);
}

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

    pipe(pipe_fd);

    pid1 = fork();
    if (pid1 == 0) { // First child: executes 'ls -l'
        close(pipe_fd[0]); // Close read end
        dup2(pipe_fd[1], STDOUT_FILENO); // Redirect stdout to the pipe
        execlp("ls", "ls", "-l", NULL);
        perror("execlp ls failed");
        exit(EXIT_FAILURE);
    }

    pid2 = fork();
    if (pid2 == 0) { // Second child: executes 'wc -l'
        close(pipe_fd[1]); // Close write end
        dup2(pipe_fd[0], STDIN_FILENO); // Redirect stdin from the pipe
        execlp("wc", "wc", "-l", NULL);
        perror("execlp wc failed");
        exit(EXIT_FAILURE);
    }

    // Parent process
    block_signals(); // Block Ctrl-C and Ctrl-\ signals
    close(pipe_fd[0]);
    close(pipe_fd[1]);

    wait(NULL); // Wait for both children to finish
    wait(NULL);

    return 0;
}
```

SLIP 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;

  }


  DIR *dir = opendir(".");

  if (dir == NULL) {

    perror("opendir");

    return 1;

  }


  struct dirent *entry;

  int found = 0;

  while ((entry = readdir(dir)) != NULL) {

    if (strcmp(entry->d_name, argv[1]) == 0) {

      found = 1;

      break;

    }

  }


  closedir(dir);

  if (found) {

    printf("File '%s' is present in the current directory.\n", argv[1]);
  } else {
    printf("File '%s' is not found.\n", argv[1]);
  }

  return 0;
}
```

Q.2) Write a C program to Identify the type (Directory, character device, Block device, Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call. [20 Marks]

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

void check_file_type(struct stat file_stat) {
    if (S_ISREG(file_stat.st_mode)) printf("Regular file\n");
    else if (S_ISDIR(file_stat.st_mode)) printf("Directory\n");
    else if (S_ISCHR(file_stat.st_mode)) printf("Character device\n");
    else if (S_ISBLK(file_stat.st_mode)) printf("Block device\n");
    else if (S_ISFIFO(file_stat.st_mode)) printf("FIFO or pipe\n");
    else if (S_ISLNK(file_stat.st_mode)) printf("Symbolic link\n");
    else if (S_ISSOCK(file_stat.st_mode)) printf("Socket\n");
    else printf("Unknown type\n");
}

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

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

    printf("File type of %s: ", argv[1]);
    check_file_type(file_stat);
    return 0;
}
```

SLIP 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>
#include <time.h>

int main(int argc, char *argv[]) {
  struct stat file_stat;
  if (stat(argv[1], &file_stat) != 0) {
    perror("stat");
    return 1;
  }
  printf("Inode: %ld\n", file_stat.st_ino);
  printf("Links: %ld\n", file_stat.st_nlink);
  printf("Permissions: %o\n", file_stat.st_mode & 0777);
  printf("Size: %ld bytes\n", file_stat.st_size);
  printf("Access time: %s", ctime(&file_stat.st_atime));
  printf("Modification time: %s", ctime(&file_stat.st_mtime));
  return 0;
}
```

Q.2) Write a C program which creates a child process to run linux/ unix command or any user defined program. The parent process set the signal handler for death of child signal and Alarm signal. If a child process does not complete its execution in 5 second then parent process kills child process [20 Marks]

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

```
pid_t child_pid;

void handle_alarm(int sig) {
    printf("Child did not finish in 5 seconds, killing child\n");
    kill(child_pid, SIGKILL);
}

void handle_child_death(int sig) {
    int status;
    waitpid(child_pid, &status, 0);

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

int main() {
    signal(SIGALRM, handle_alarm);      // Set signal handler for alarm
    signal(SIGCHLD, handle_child_death); // Set signal handler for child termination

    child_pid = fork();

    if (child_pid == 0) { // Child process
        printf("Child process running\n");
        sleep(10); // Simulate a long-running task
        printf("Child process finished\n");
        exit(0);
    } else { // Parent process
        alarm(5);  // Set alarm for 5 seconds
        pause();   // Wait for either alarm or child termination
    }


    return 0;
}
```

SLIP 25

Q.1) Write a C Program that demonstrates redirection of standard output to a file [10 Marks]

```
#include <stdio.h>


int main() {
    FILE *file = freopen("output.txt", "w", stdout);


    if (file == NULL) {
        perror("freopen");
        return 1;
    }


    printf("This text will be written to output.txt\n");


    fclose(file);


    return 0;
}
```

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


int main() {
    int file = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);


    if (file < 0) {
        perror("open");
        return 1;
    }

    int saved_stdout = dup(1);
    dup2(file, 1);

    printf("This will be written to output.txt\n");
    printf("This is another line of text.\n");

    fflush(stdout);
    dup2(saved_stdout, 1);
    close(file);

    return 0;
}
```