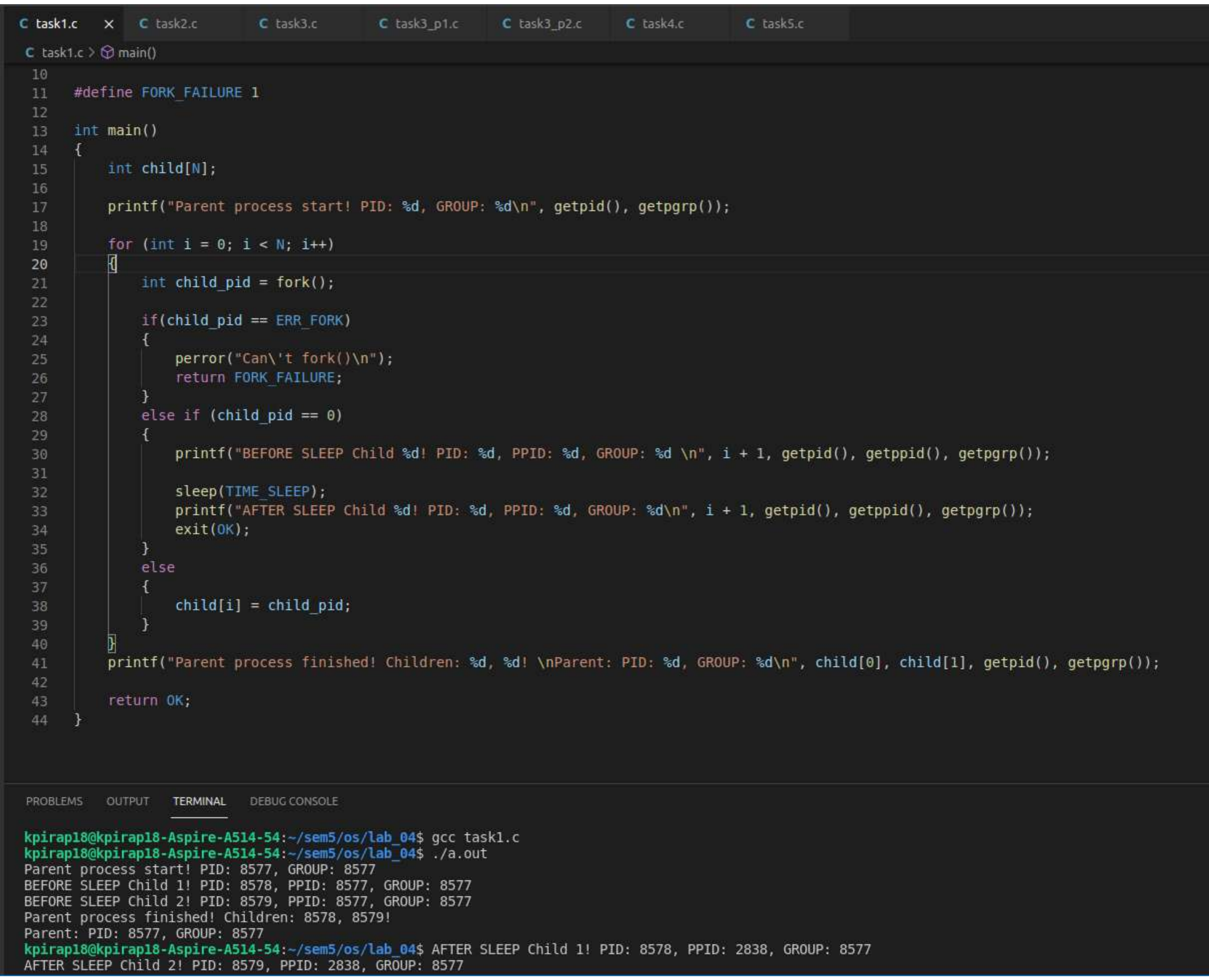
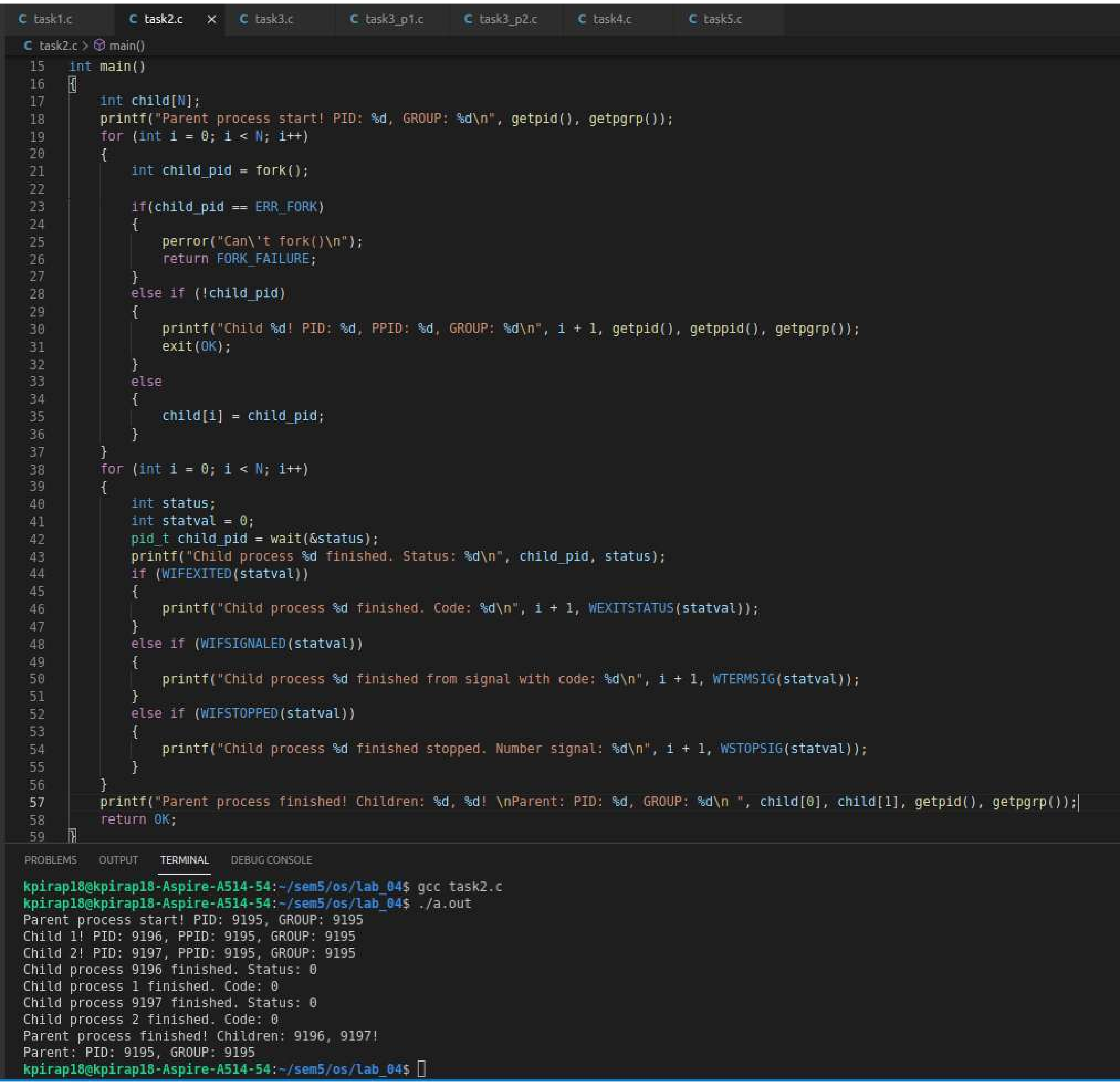
Задание 1



```c

#define FORK_FAILURE 1

int main()
{
    int child[N];

    printf("Parent process start! PID: %d, GROUP: %d\n", getpid(), getpgrp());

    for (int i = 0; i < N; i++)
    {
        int child_pid = fork();

        if(child_pid == ERR_FORK)
        {
            perror("Can\'t fork()\n");
            return FORK_FAILURE;
        }
        else if (child_pid == 0)
        {
            printf("BEFORE SLEEP Child %d! PID: %d, PPID: %d, GROUP: %d \n", i + 1, getpid(), getppid(), getpgrp());

            sleep(TIME_SLEEP);
            printf("AFTER SLEEP Child %d! PID: %d, PPID: %d, GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
            exit(OK);
        }
        else
        {
            child[i] = child_pid;
        }
    }
    printf("Parent process finished! Children: %d, %d! \nParent: PID: %d, GROUP: %d\n", child[0], child[1], getpid(), getpgrp());

    return OK;
}
```

```
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task1.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 8577, GROUP: 8577
BEFORE SLEEP Child 1! PID: 8578, PPID: 8577, GROUP: 8577
BEFORE SLEEP Child 2! PID: 8579, PPID: 8577, GROUP: 8577
Parent process finished! Children: 8578, 8579!
Parent: PID: 8577, GROUP: 8577
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ AFTER SLEEP Child 1! PID: 8578, PPID: 2838, GROUP: 8577
AFTER SLEEP Child 2! PID: 8579, PPID: 2838, GROUP: 8577
```

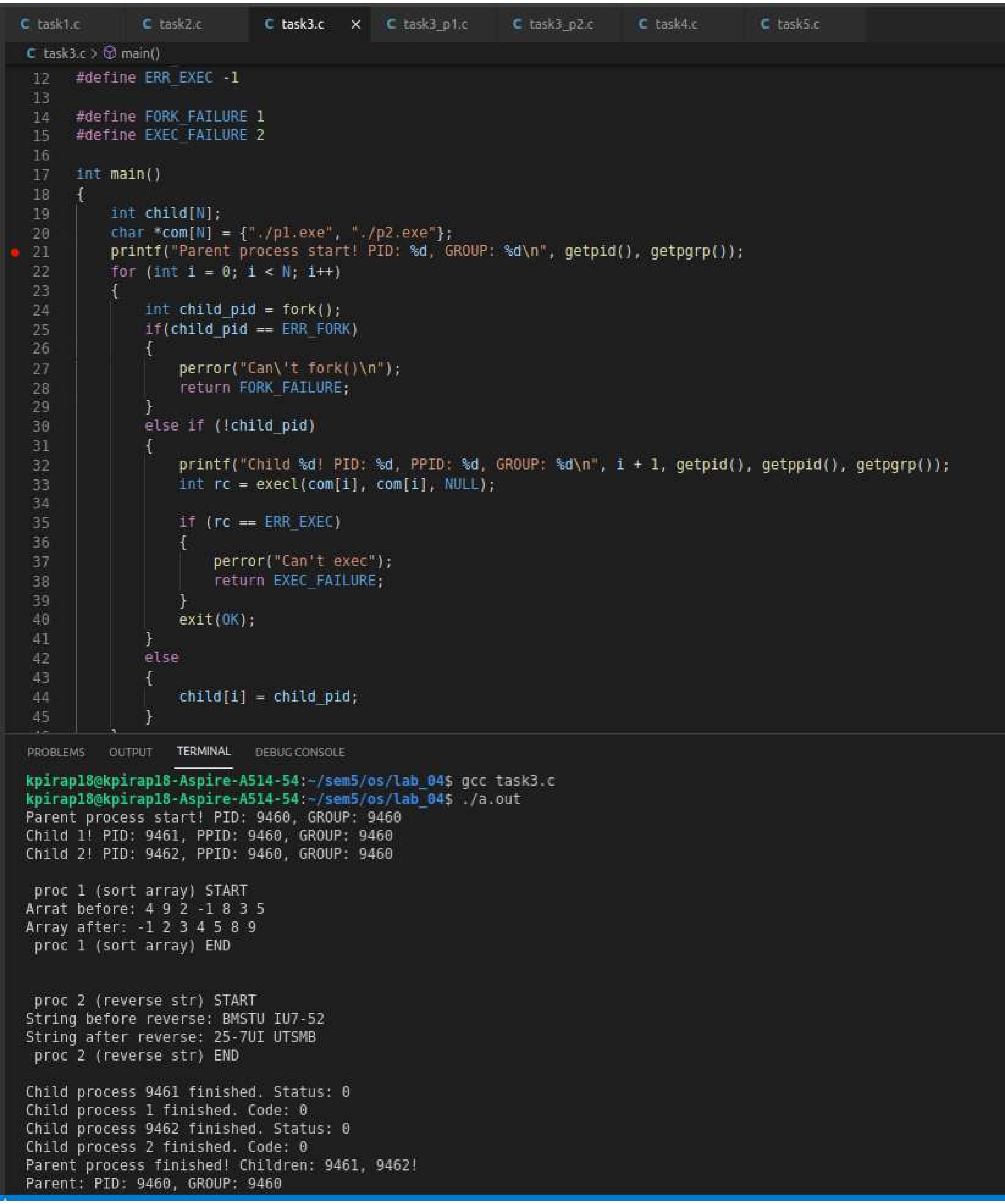
Задание 2



```c
int main()
{
    int child[N];
    printf("Parent process start! PID: %d, GROUP: %d\n", getpid(), getpgrp());
    for (int i = 0; i < N; i++)
    {
        int child_pid = fork();

        if(child_pid == ERR_FORK)
        {
            perror("Can\'t fork()\n");
            return FORK_FAILURE;
        }
        else if (!child_pid)
        {
            printf("Child %d! PID: %d, PPID: %d, GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
            exit(OK);
        }
        else
        {
            child[i] = child_pid;
        }
    }
    for (int i = 0; i < N; i++)
    {
        int status;
        int statval = 0;
        pid_t child_pid = wait(&status);
        printf("Child process %d finished. Status: %d\n", child_pid, status);
        if (WIFEXITED(statval))
        {
            printf("Child process %d finished. Code: %d\n", i + 1, WEXITSTATUS(statval));
        }
        else if (WIFSIGNALED(statval))
        {
            printf("Child process %d finished from signal with code: %d\n", i + 1, WTERMSIG(statval));
        }
        else if (WIFSTOPPED(statval))
        {
            printf("Child process %d finished stopped. Number signal: %d\n", i + 1, WSTOPSIG(statval));
        }
    }
    printf("Parent process finished! Children: %d, %d! \nParent: PID: %d, GROUP: %d\n ", child[0], child[1], getpid(), getpgrp());
    return OK;
}
```

```
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task2.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 9195, GROUP: 9195
Child 1! PID: 9196, PPID: 9195, GROUP: 9195
Child 2! PID: 9197, PPID: 9195, GROUP: 9195
Child process 9196 finished. Status: 0
Child process 1 finished. Code: 0
Child process 9197 finished. Status: 0
Child process 2 finished. Code: 0
Parent process finished! Children: 9196, 9197!
Parent: PID: 9195, GROUP: 9195
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$
```

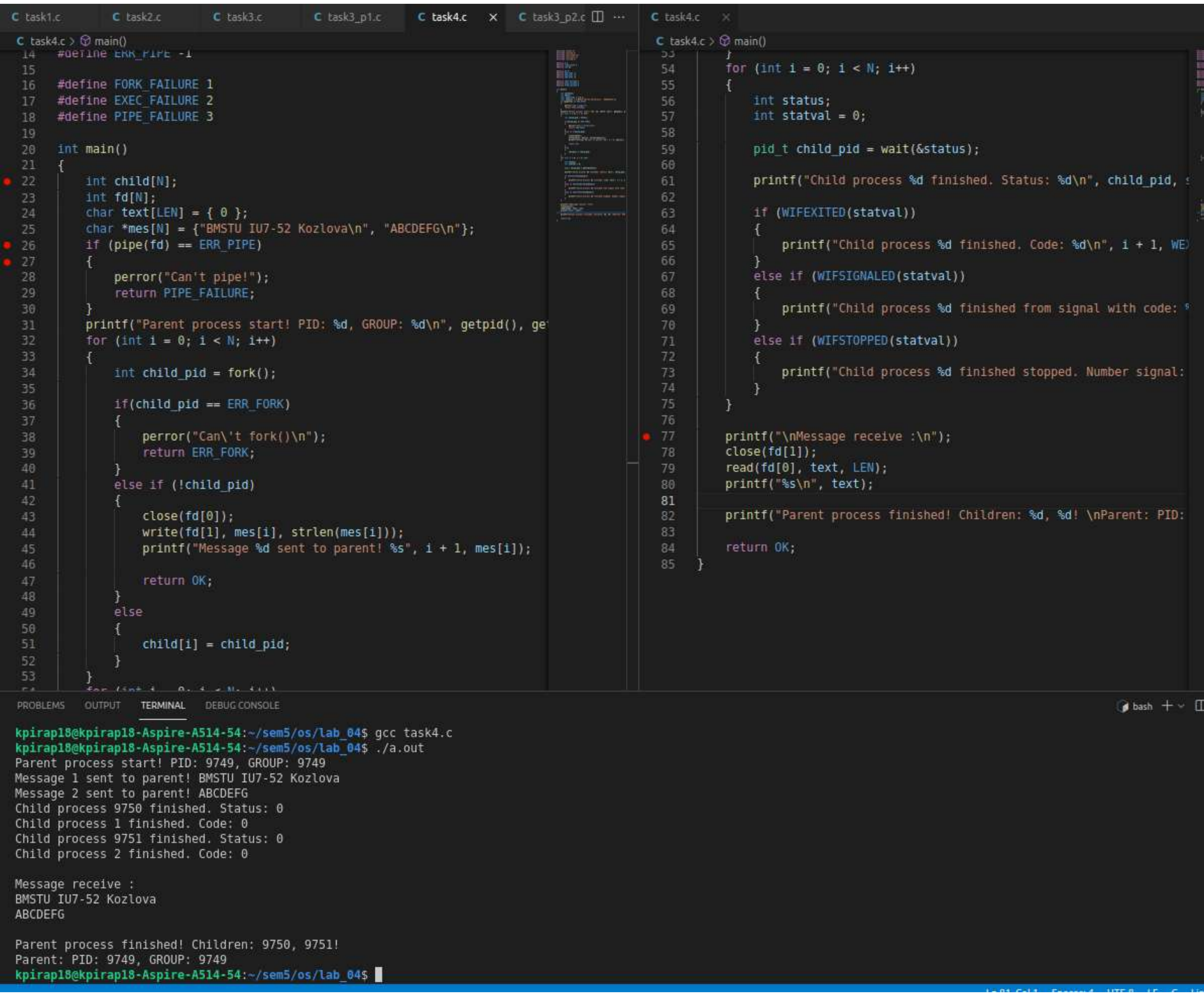
Задание 3



```c
#define ERR_EXEC -1

#define FORK_FAILURE 1
#define EXEC_FAILURE 2

int main()
{
    int child[N];
    char *com[N] = {"./p1.exe", "./p2.exe"};
    printf("Parent process start! PID: %d, GROUP: %d\n", getpid(), getpgrp());
    for (int i = 0; i < N; i++)
    {
        int child_pid = fork();
        if(child_pid == ERR_FORK)
        {
            perror("Can\'t fork()\n");
            return FORK_FAILURE;
        }
        else if (!child_pid)
        {
            printf("Child %d! PID: %d, PPID: %d, GROUP: %d\n", i + 1, getpid(), getppid(), getpgrp());
            int rc = execl(com[i], com[i], NULL);

            if (rc == ERR_EXEC)
            {
                perror("Can't exec");
                return EXEC_FAILURE;
            }
            exit(OK);
        }
        else
        {
            child[i] = child_pid;
        }
```

```
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task3.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 9460, GROUP: 9460
Child 1! PID: 9461, PPID: 9460, GROUP: 9460
Child 2! PID: 9462, PPID: 9460, GROUP: 9460

 proc 1 (sort array) START
Arrat before: 4 9 2 -1 8 3 5
Array after: -1 2 3 4 5 8 9
 proc 1 (sort array) END


 proc 2 (reverse str) START
String before reverse: BMSTU IU7-52
String after reverse: 25-7UI UTSMB
 proc 2 (reverse str) END

Child process 9461 finished. Status: 0
Child process 1 finished. Code: 0
Child process 9462 finished. Status: 0
Child process 2 finished. Code: 0
Parent process finished! Children: 9461, 9462!
Parent: PID: 9460, GROUP: 9460
```

Задание 4

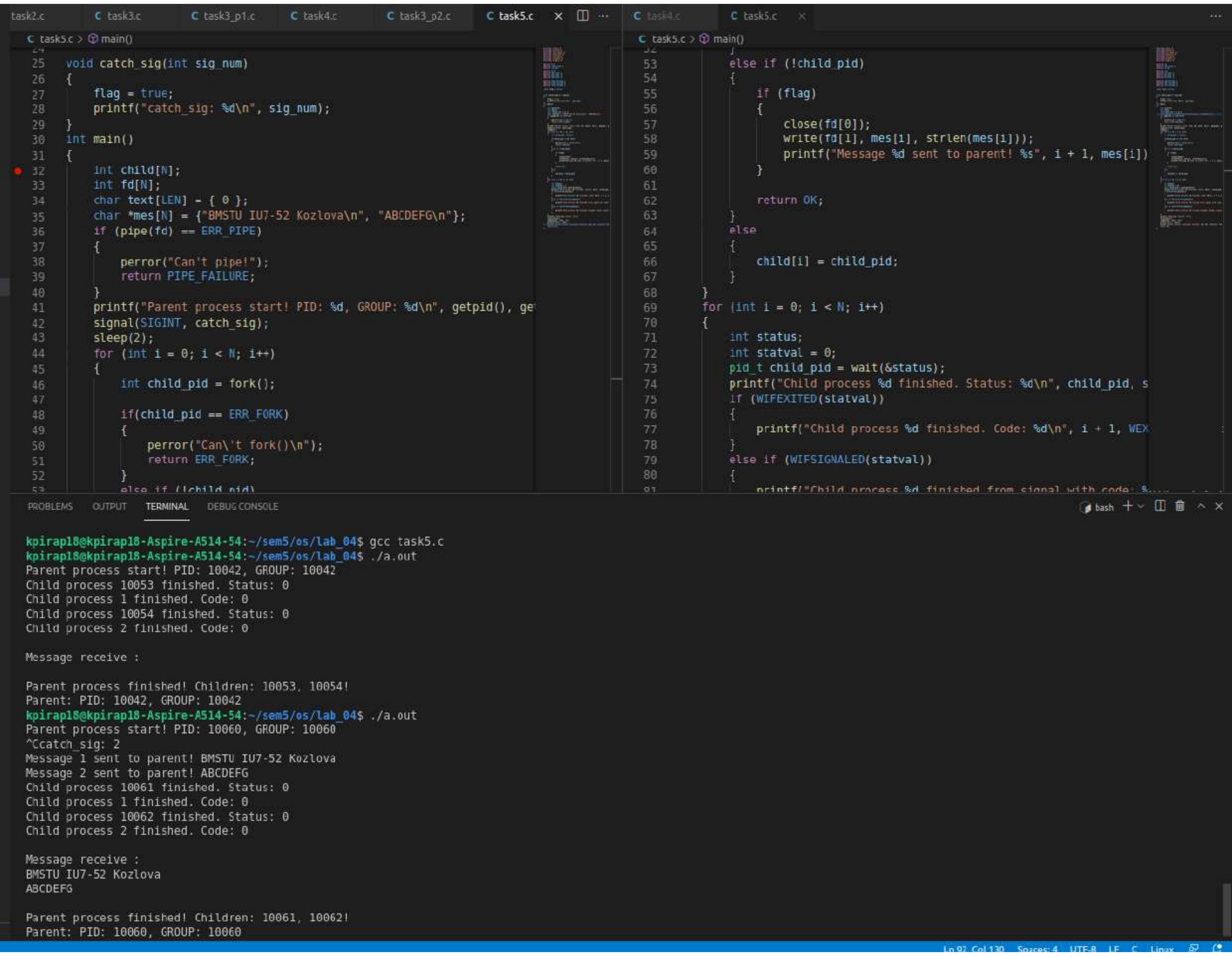


```c
#define ERR_PIPE -1

#define FORK_FAILURE 1
#define EXEC_FAILURE 2
#define PIPE_FAILURE 3

int main()
{
    int child[N];
    int fd[N];
    char text[LEN] = { 0 };
    char *mes[N] = {"BMSTU IU7-52 Kozlova\n", "ABCDEFG\n"};
    if (pipe(fd) == ERR_PIPE)
    {
        perror("Can't pipe!");
        return PIPE_FAILURE;
    }
    printf("Parent process start! PID: %d, GROUP: %d\n", getpid(), ge
    for (int i = 0; i < N; i++)
    {
        int child_pid = fork();

        if(child_pid == ERR_FORK)
        {
            perror("Can\'t fork()\n");
            return ERR_FORK;
        }
        else if (!child_pid)
        {
            close(fd[0]);
            write(fd[1], mes[i], strlen(mes[i]));
            printf("Message %d sent to parent! %s", i + 1, mes[i]);

            return OK;
        }
        else
        {
            child[i] = child_pid;
        }
    }
```

```c
    }
    for (int i = 0; i < N; i++)
    {
        int status;
        int statval = 0;

        pid_t child_pid = wait(&status);

        printf("Child process %d finished. Status: %d\n", child_pid, s

        if (WIFEXITED(statval))
        {
            printf("Child process %d finished. Code: %d\n", i + 1, WE
        }
        else if (WIFSIGNALED(statval))
        {
            printf("Child process %d finished from signal with code: %
        }
        else if (WIFSTOPPED(statval))
        {
            printf("Child process %d finished stopped. Number signal:
        }
    }

    printf("\nMessage receive :\n");
    close(fd[1]);
    read(fd[0], text, LEN);
    printf("%s\n", text);

    printf("Parent process finished! Children: %d, %d! \nParent: PID:

    return OK;
}
```

```
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task4.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 9749, GROUP: 9749
Message 1 sent to parent! BMSTU IU7-52 Kozlova
Message 2 sent to parent! ABCDEFG
Child process 9750 finished. Status: 0
Child process 1 finished. Code: 0
Child process 9751 finished. Status: 0
Child process 2 finished. Code: 0

Message receive :
BMSTU IU7-52 Kozlova
ABCDEFG

Parent process finished! Children: 9750, 9751!
Parent: PID: 9749, GROUP: 9749
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$
```

Задание 5



```c
void catch_sig(int sig_num)
{
    flag = true;
    printf("catch_sig: %d\n", sig_num);
}
int main()
{
    int child[N];
    int fd[N];
    char text[LEN] = { 0 };
    char *mes[N] = {"BMSTU IU7-52 Kozlova\n", "ABCDEFG\n"};
    if (pipe(fd) == ERR_PIPE)
    {
        perror("Can't pipe!");
        return PIPE_FAILURE;
    }
    printf("Parent process start! PID: %d, GROUP: %d\n", getpid(), ge
    signal(SIGINT, catch_sig);
    sleep(2);
    for (int i = 0; i < N; i++)
    {
        int child_pid = fork();

        if(child_pid == ERR_FORK)
        {
            perror("Can\'t fork()\n");
            return ERR_FORK;
        }
```

```c
        else if (!child_pid)
        {
            if (flag)
            {
                close(fd[0]);
                write(fd[1], mes[i], strlen(mes[i]));
                printf("Message %d sent to parent! %s", i + 1, mes[i])
            }

            return OK;
        }
        else
        {
            child[i] = child_pid;
        }
    }
    for (int i = 0; i < N; i++)
    {
        int status;
        int statval = 0;
        pid_t child_pid = wait(&status);
        printf("Child process %d finished. Status: %d\n", child_pid, s
        if (WIFEXITED(statval))
        {
            printf("Child process %d finished. Code: %d\n", i + 1, WEX
        }
        else if (WIFSIGNALED(statval))
        {
```

```
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ gcc task5.c
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 10042, GROUP: 10042
Child process 10053 finished. Status: 0
Child process 1 finished. Code: 0
Child process 10054 finished. Status: 0
Child process 2 finished. Code: 0

Message receive :

Parent process finished! Children: 10053, 10054!
Parent: PID: 10042, GROUP: 10042
kpirap18@kpirap18-Aspire-A514-54:~/sem5/os/lab_04$ ./a.out
Parent process start! PID: 10060, GROUP: 10060
^Ccatch_sig: 2
Message 1 sent to parent! BMSTU IU7-52 Kozlova
Message 2 sent to parent! ABCDEFG
Child process 10061 finished. Status: 0
Child process 1 finished. Code: 0
Child process 10062 finished. Status: 0
Child process 2 finished. Code: 0

Message receive :
BMSTU IU7-52 Kozlova
ABCDEFG

Parent process finished! Children: 10061, 10062!
Parent: PID: 10060, GROUP: 10060
```