



دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

گزارش کار آزمایشگاه آزمایشگاه سیستم‌های عامل

گزارش آزمایش شماره ۵
(ارتباط بین پردازهای)

۲۰

ارشیا یوسف‌نیا (۴۰۱۱۱۰۴۱۵)
محمدعارف زارع زاده (۴۰۱۱۰۶۰۱۷)
دکتر بیگی
تابستان ۱۴۰۴

شماره‌ی گروه:

گروه:

استاد درس:

تاریخ:

فهرست مطالب

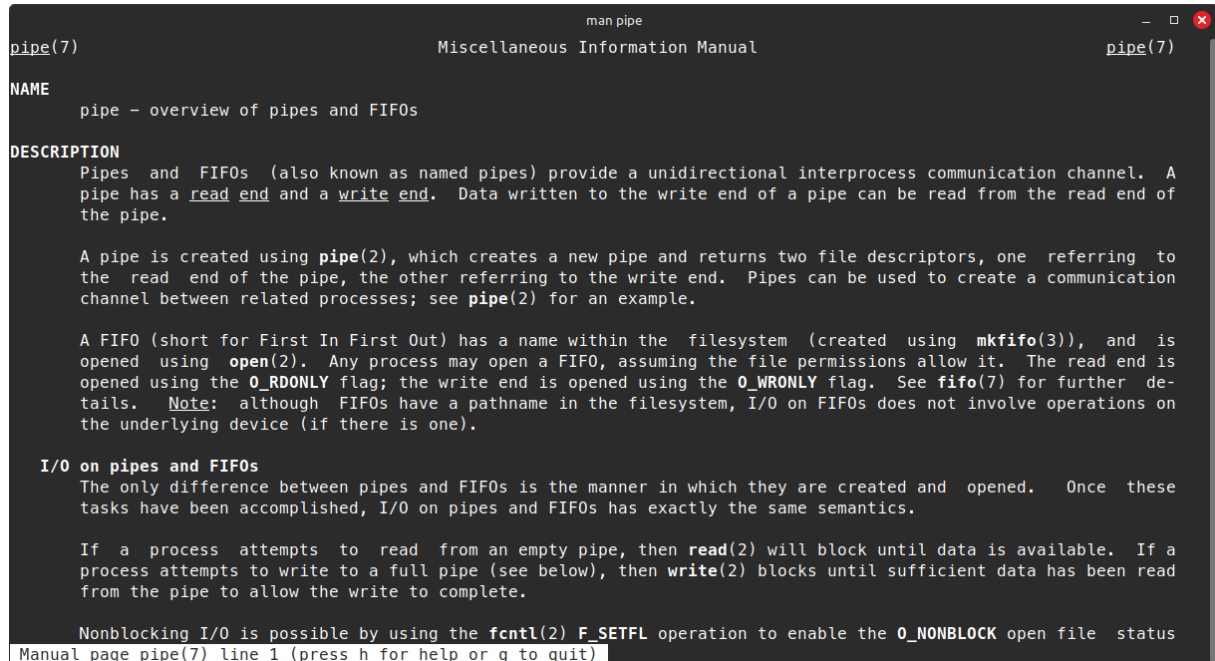
۱	ایجاد pipe یک سویه	۱
۵	۱.۱ فعالیت ها	۵
۷	۲ سیگنال	۷
۱۰	۱.۲ تمرین	۱۰

لیست تصاویر

۱	توضیحات man pipe	۱
۱	برنامه حداقلی برای ساخت موفق یک pipe یک‌سویه	۲
۲	مراحل کامپایل و اجرای موفق برنامه شکل ۲	۳
۳	برنامه انتقال پیام متنی از پردازنده والد به فرزند و چاپ آن در فرزند	۴
۴	مراحل کامپایل و اجرای موفق برنامه شکل ۴	۵
۶	برنامه اجرای ls در پردازنده والد و انتقال آن با pipe به پردازنده فرزند و اجرای wc روی این ورودی و خروجی دادن آن.	۵
۶	مراحل کامپایل و اجرای موفق برنامه شکل ۶	۷
۷	توضیحات man signal	۸
۷	صفحه manual درباره alarm	۹
۸	استفاده از سیگنال alarm در یک برنامه ساده به همراه نتیجه اجرا	۱۰
۸	تغییر برنامه شکل ۱۰ با signal و pause برای کارایی خواسته شده	۱۱
۹	مراحل کامپایل و اجرای برنامه شکل ۱۱	۱۲
۱۰	برنامه حداقلی برای خروج از برنامه با دوبار CTRL+C به جای پیشفرض یک‌بار	۱۳
۱۰	مرحله کامپایل و اجرای برنامه شکل ۱۳	۱۴

لیست جداول

۱ ایجاد pipe یک سویه



```
man pipe
pipe(7)
NAME
    pipe - overview of pipes and FIFOs
DESCRIPTION
    Pipes and FIFOs (also known as named pipes) provide a unidirectional interprocess communication channel. A pipe has a read end and a write end. Data written to the write end of a pipe can be read from the read end of the pipe.

    A pipe is created using pipe(2), which creates a new pipe and returns two file descriptors, one referring to the read end of the pipe, the other referring to the write end. Pipes can be used to create a communication channel between related processes; see pipe(2) for an example.

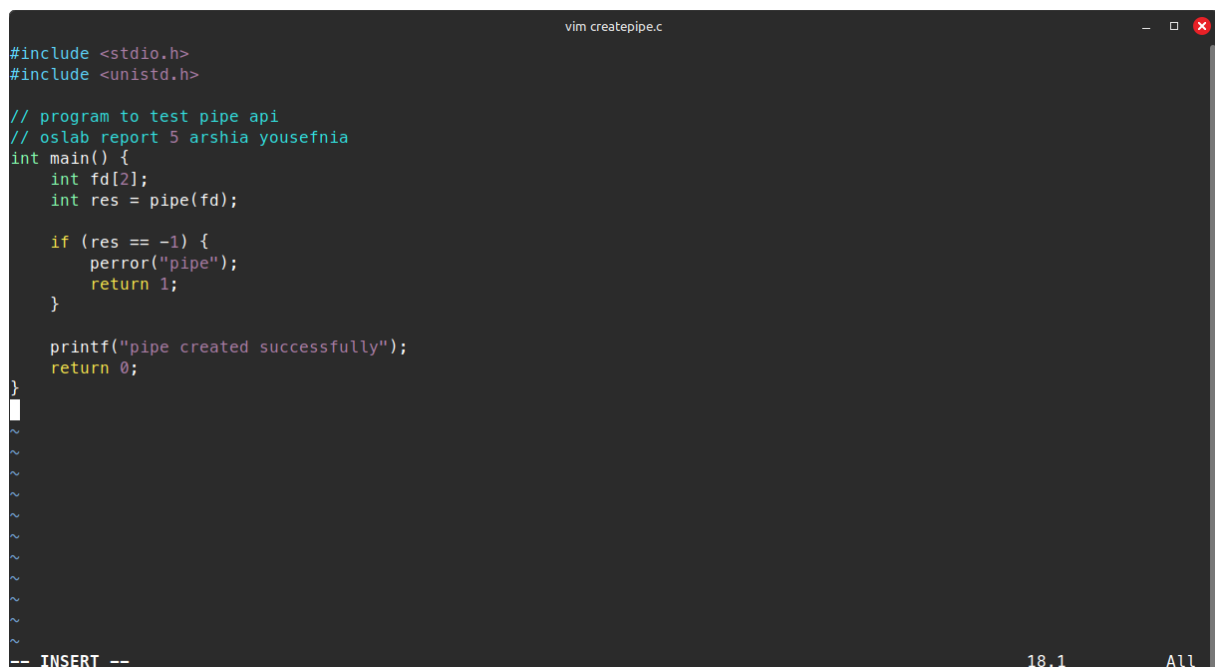
    A FIFO (short for First In First Out) has a name within the filesystem (created using mkfifo(3)), and is opened using open(2). Any process may open a FIFO, assuming the file permissions allow it. The read end is opened using the O_RDONLY flag; the write end is opened using the O_WRONLY flag. See fifo(7) for further details. Note: although FIFOs have a pathname in the filesystem, I/O on FIFOs does not involve operations on the underlying device (if there is one).

    I/O on pipes and FIFOs
    The only difference between pipes and FIFOs is the manner in which they are created and opened. Once these tasks have been accomplished, I/O on pipes and FIFOs has exactly the same semantics.

    If a process attempts to read from an empty pipe, then read(2) will block until data is available. If a process attempts to write to a full pipe (see below), then write(2) blocks until sufficient data has been read from the pipe to allow the write to complete.

    Nonblocking I/O is possible by using the fcntl(2) F_SETFL operation to enable the O_NONBLOCK open file status
Manual page pipe(7) line 1 (press h for help or q to quit)
```

شکل ۱: توضیحات man pipe



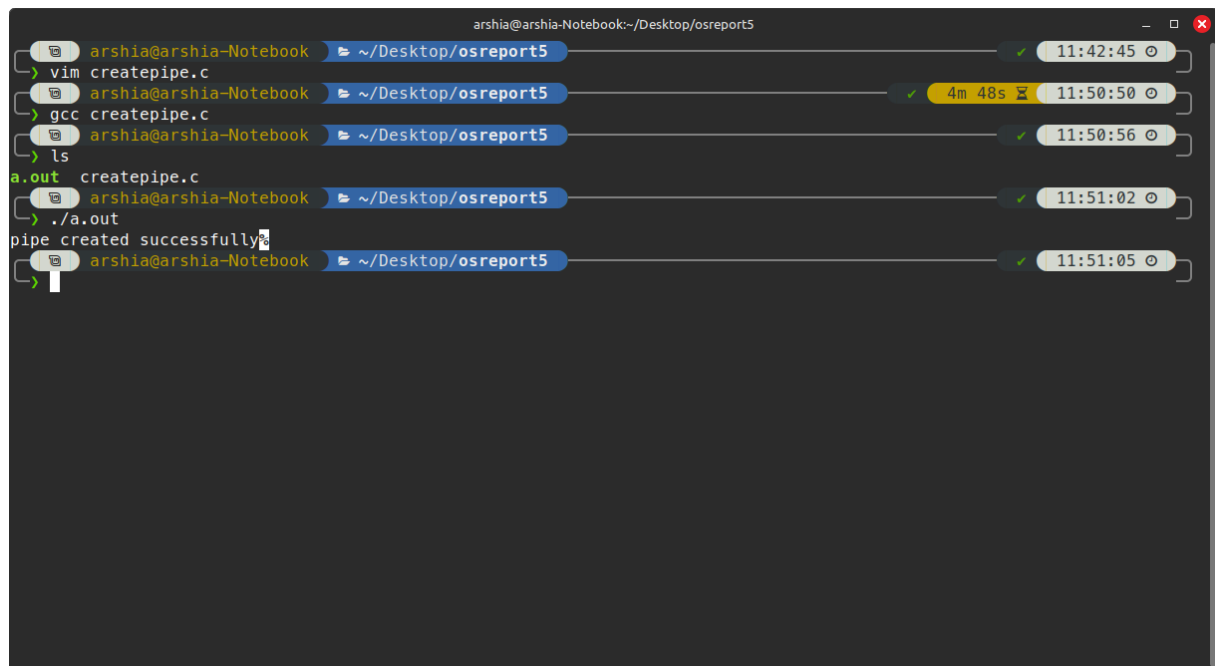
```
vim createpipe.c
#include <stdio.h>
#include <unistd.h>

// program to test pipe api
// oslab report 5 arshia yousefnia
int main() {
    int fd[2];
    int res = pipe(fd);

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

    printf("pipe created successfully");
    return 0;
}
-- INSERT --
18,1 All
```

شکل ۲: برنامه حداقلی برای ساخت موفق یک pipe یک سویه



```
arshia@arshia-Notebook:~/Desktop/osreport5
> vim createpipe.c
arshia@arshia-Notebook:~/Desktop/osreport5
> gcc createpipe.c
arshia@arshia-Notebook:~/Desktop/osreport5
> ls
a.out createpipe.c
arshia@arshia-Notebook:~/Desktop/osreport5
> ./a.out
pipe created successfully
arshia@arshia-Notebook:~/Desktop/osreport5
```

شکل ۳: مراحل کامپایل و اجرای موفق برنامه شکل ۲

```
vim simplemessagepipe.c

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

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

    res = pipe(fd);
    if (res == -1) {
        perror("pipe");
        return 1;
    }
    pid = fork();
    if (pid == -1) {
        perror("fork");
        return 1;
    }
    if (pid > 0) {
        close(fd[0]);

        const char *msg = "Hello World";
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);

        wait(NULL);
    }
    else {
        close(fd[1]);

        char buf[100];
        read(fd[0], buf, sizeof(buf));
        printf("Child received: %s\n", buf);

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

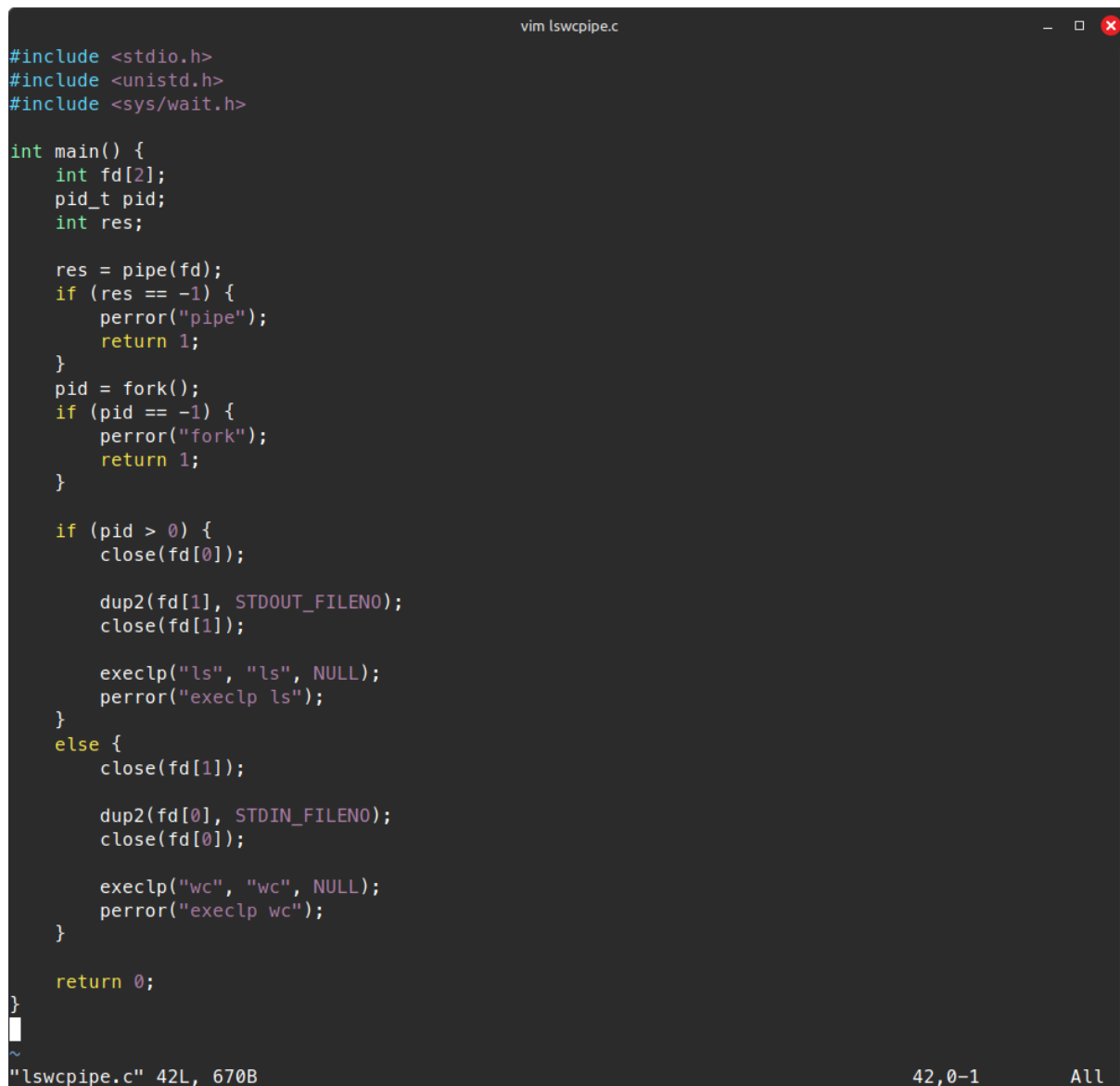
1,1 Top

شکل ۴: برنامه انتقال پیام متنی از پردازه والد به فرزند و چاپ آن در فرزند

```
arshia@arshia-Notebook:~/Desktop/osreport5
> gcc simplemessagepipe.c
> ls
a.out createpipe.c simplemessagepipe.c
> ./a.out
Child received: Hello World
```

شکل ۵: مراحل کامپایل و اجرای موفق برنامه شکل ۴

۱.۱ فعالیت‌ها



```
vim lswcpipe.c

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

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

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

    if (pid > 0) {
        close(fd[0]);

        dup2(fd[1], STDOUT_FILENO);
        close(fd[1]);

        execlp("ls", "ls", NULL);
        perror("execlp ls");
    }
    else {
        close(fd[1]);

        dup2(fd[0], STDIN_FILENO);
        close(fd[0]);

        execlp("wc", "wc", NULL);
        perror("execlp wc");
    }

    return 0;
}

~
"lswcpipe.c" 42L, 670B                                42,0-1                                All
```

شکل ۶: برنامه اجرای `ls` در پردازش والد و انتقال آن با `pipe` به پردازش فرزند و اجرای `wc` روی این ورودی و خروجی دادن آن.

```
arshia@arshia-Notebook:~/Desktop/osreport5
> vim lswcpipе.c
> gcc lswcpipе.c
> ls
a.out createpipe.c lswcpipе.c simplemessagepipe.c
> ./a.out
4 4 50
> ls | wc
4 4 50
```

شکل ۷: مراحل کامپایل و اجرای موفق برنامه شکل ۶

۲ سیگنال

```
signal(7) Miscellaneous Information Manual man signal signal(7)
NAME
  signal - overview of signals
DESCRIPTION
  Linux supports both POSIX reliable signals (hereinafter "standard signals") and POSIX real-time signals.

  Signal dispositions
  Each signal has a current disposition, which determines how the process behaves when it is delivered the signal.
  The entries in the "Action" column of the table below specify the default disposition for each signal, as follows:
  Term  Default action is to terminate the process.
  Ign   Default action is to ignore the signal.
  Core  Default action is to terminate the process and dump core (see core(5)).
  Stop  Default action is to stop the process.
  Cont  Default action is to continue the process if it is currently stopped.

  A process can change the disposition of a signal using sigaction(2) or signal(2). (The latter is less portable when establishing a signal handler; see signal(2) for details.)
  Using these system calls, a process can elect one of the following behaviors to occur on delivery of the signal: perform the default action; ignore the signal; or catch the signal with a signal handler, a programmer-defined function that is automatically invoked when the signal is delivered.

  By default, a signal handler is invoked on the normal process stack. It is possible to arrange that the signal handler uses an alternate stack; see sigaltstack(2) for a discussion of how to do this and when it might be useful.

  The signal disposition is a per-process attribute: in a multithreaded application, the disposition of a particular signal is the same for all threads.
  A child created via fork(2) inherits a copy of its parent's signal dispositions. During an execve(2), the dispositions of handled signals are reset to the default; the dispositions of ignored signals are left unchanged.

  Sending a signal
  The following system calls and library functions allow the caller to send a signal:
  raise(3)
    Sends a signal to the calling thread.
  kill(2)
    Sends a signal to a specified process, to all members of a specified process group, or to all processes on the system.
  pidfd_send_signal(2)
    Sends a signal to a process identified by a PID file descriptor.
  killpg(3)
    Sends a signal to all of the members of a specified process group.
  pthread_kill(3)
    Sends a signal to a specified POSIX thread in the same process as the caller.
  tkill(2)
    Sends a signal to a specified thread within a specific process. (This is the system call used to implement pthread_kill(3).)
  sigqueue(3)
    Sends a real-time signal with accompanying data to a specified process.

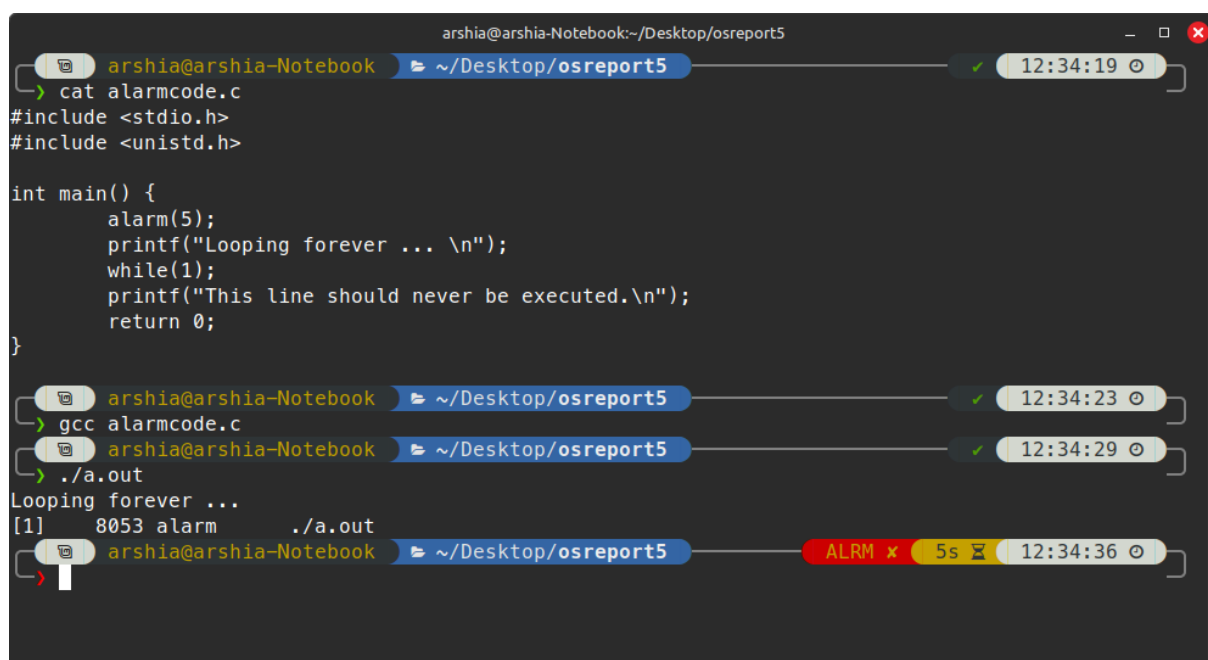
  Waiting for a signal to be caught
  The following system calls suspend execution of the calling thread until a signal is caught (or an unhandled signal terminates the process):
  Manual page signal(7) line 1 (press h for help or q to quit)
```

شکل ۸: توضیحات man signal

```
alarm(2) — Linux manual page
NAME | LIBRARY | SYNOPSIS | DESCRIPTION | RETURN VALUE | STANDARDS | HISTORY | NOTES | SEE ALSO | COLOPHON
[Search online pages]

alarm(2) System Calls Manual alarm(2)
NAME
  alarm - set an alarm clock for delivery of a signal
LIBRARY
  Standard C library (libc, -lc)
SYNOPSIS
  #include <unistd.h>
  unsigned int alarm(unsigned int seconds);
DESCRIPTION
  alarm() arranges for a SIGALRM signal to be delivered to the calling process in seconds seconds.
  If seconds is zero, any pending alarm is canceled.
  In any event any previously set alarm() is canceled.
RETURN VALUE
  alarm() returns the number of seconds remaining until any previously scheduled alarm was due to be delivered, or zero if there was no previously scheduled alarm.
```

شکل ۹: صفحه manual درباره alarm

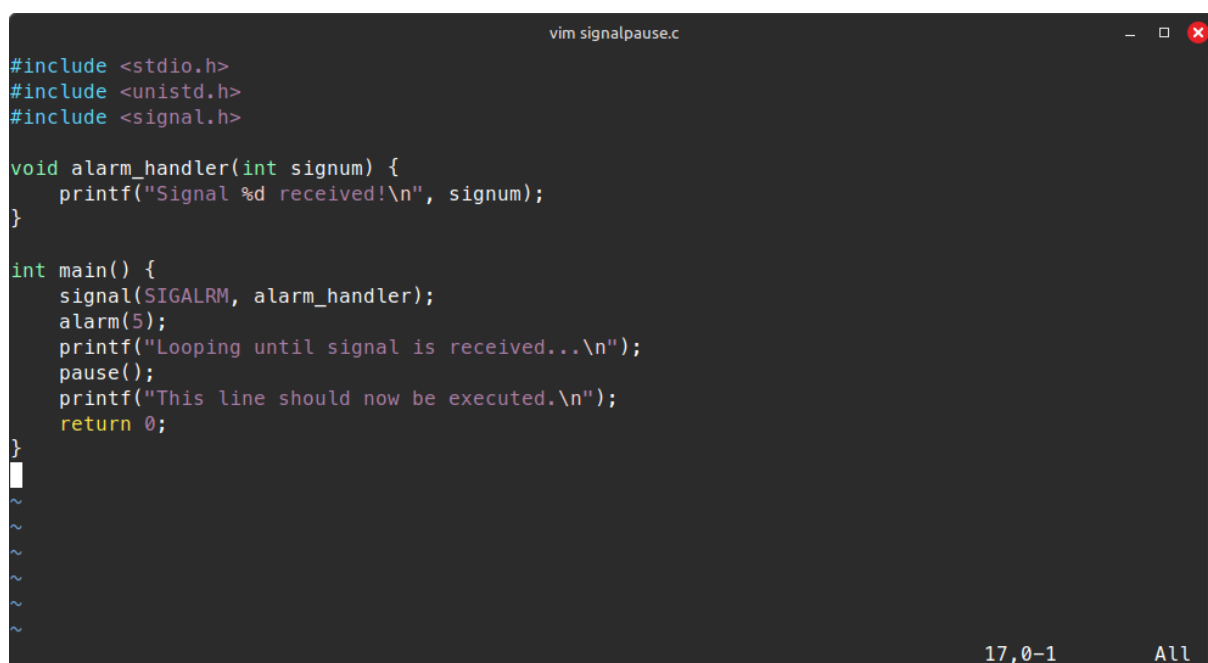


```
arshia@arshia-Notebook: ~/Desktop/osreport5
> cat alarmcode.c
#include <stdio.h>
#include <unistd.h>

int main() {
    alarm(5);
    printf("Looping forever ... \n");
    while(1);
    printf("This line should never be executed.\n");
    return 0;
}

arshia@arshia-Notebook: ~/Desktop/osreport5
> gcc alarmcode.c
arshia@arshia-Notebook: ~/Desktop/osreport5
> ./a.out
Looping forever ...
[1] 8053 alarm ./a.out
arshia@arshia-Notebook: ~/Desktop/osreport5
ALRM x 5s 12:34:36
```

شکل ۱۰: استفاده از سیگنال alarm در یک برنامه ساده به همراه نتیجه اجرا



```
vim signalpause.c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void alarm_handler(int signum) {
    printf("Signal %d received!\n", signum);
}

int main() {
    signal(SIGALRM, alarm_handler);
    alarm(5);
    printf("Looping until signal is received...\n");
    pause();
    printf("This line should now be executed.\n");
    return 0;
}

17,0-1 All
```

شکل ۱۱: تغییر برنامه شکل ۱۰ با signal و pause برای کارایی خواسته شده

```
arshia@arshia-Notebook:~/Desktop/osreport5
> cat signalpause.c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void alarm_handler(int signum) {
    printf("Signal %d received!\n", signum);
}

int main() {
    signal(SIGALRM, alarm_handler);
    alarm(5);
    printf("Looping until signal is received...\n");
    pause();
    printf("This line should now be executed.\n");
    return 0;
}

arshia@arshia-Notebook:~/Desktop/osreport5
> gcc signalpause.c

arshia@arshia-Notebook:~/Desktop/osreport5
> ./a.out
Looping until signal is received...
Signal 14 received!
This line should now be executed.

arshia@arshia-Notebook:~/Desktop/osreport5
>
```

شکل ۱۲: مراحل کامپایل و اجرای برنامه شکل ۱۱

```
vim doublectrlc.c

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

int interrupt_count = 0;

void sigint_handler(int signum) {
    interrupt_count++;
    if (interrupt_count == 1) {
        printf("\nFirst Ctrl+C detected. Press again to exit.\n");
    } else {
        printf("\nSecond Ctrl+C detected. Exiting.\n");
        exit(0);
    }
}

int main() {
    signal(SIGINT, sigint_handler);
    printf("Running... Press Ctrl+C to interrupt.\n");

    while (1) {
        pause();
    }

    return 0;
}

"doublectrlc.c" 28L, 519B                                28,0-1    All
```

شکل ۱۳: برنامه حداقلی برای خروج از برنامه با دوبار CTRL+C به جای پیشفرض یکبار

```
arshia@arshia-Notebook:~/Desktop/osreport5

arshia@arshia-Notebook ~/Desktop/osreport5 20s 13:36:19
> gcc doublectrlc.c
arshia@arshia-Notebook ~/Desktop/osreport5 13:36:23
> ./a.out
Running... Press Ctrl+C to interrupt.
^C
First Ctrl+C detected. Press again to exit.
^C
Second Ctrl+C detected. Exiting.
arshia@arshia-Notebook ~/Desktop/osreport5 5s 13:36:35
>
```

شکل ۱۴: مراحل کامپایل و اجرای برنامه شکل ۱۳