מגיש: שילה גילאור
תז: 302537394

שאלה 2 (25 נק'): signals

המשימה שלכם היא לבדוק האם קיים תהליך (process) עם pid מסויים. דרך אחת היא להשתמש בשליחת **kill(<pid>, 0),** כאשר סיגנל 0 (zero signal) נתון ואפשר להשתמש בו. אם שליחת סיגנל 0 נכשלת עם הודעת שגיאה ESRCH, אנחנו יודעים שהתהליך אינו קיים. אם הקריאה נכשלה עם הודעת שגיאה EPERM (התהליך קיים אבל אין לנו הרשאה לשלוח לו סיגנל כזה) או מצליחה (אם יש לנו הרשאה לשלוח סיגנל כזה), אז אנחנו יודעים שהתהליך קיים.
צריך להוסיף include errno.h כדי לקבל הודעות שגיאה כאלה.

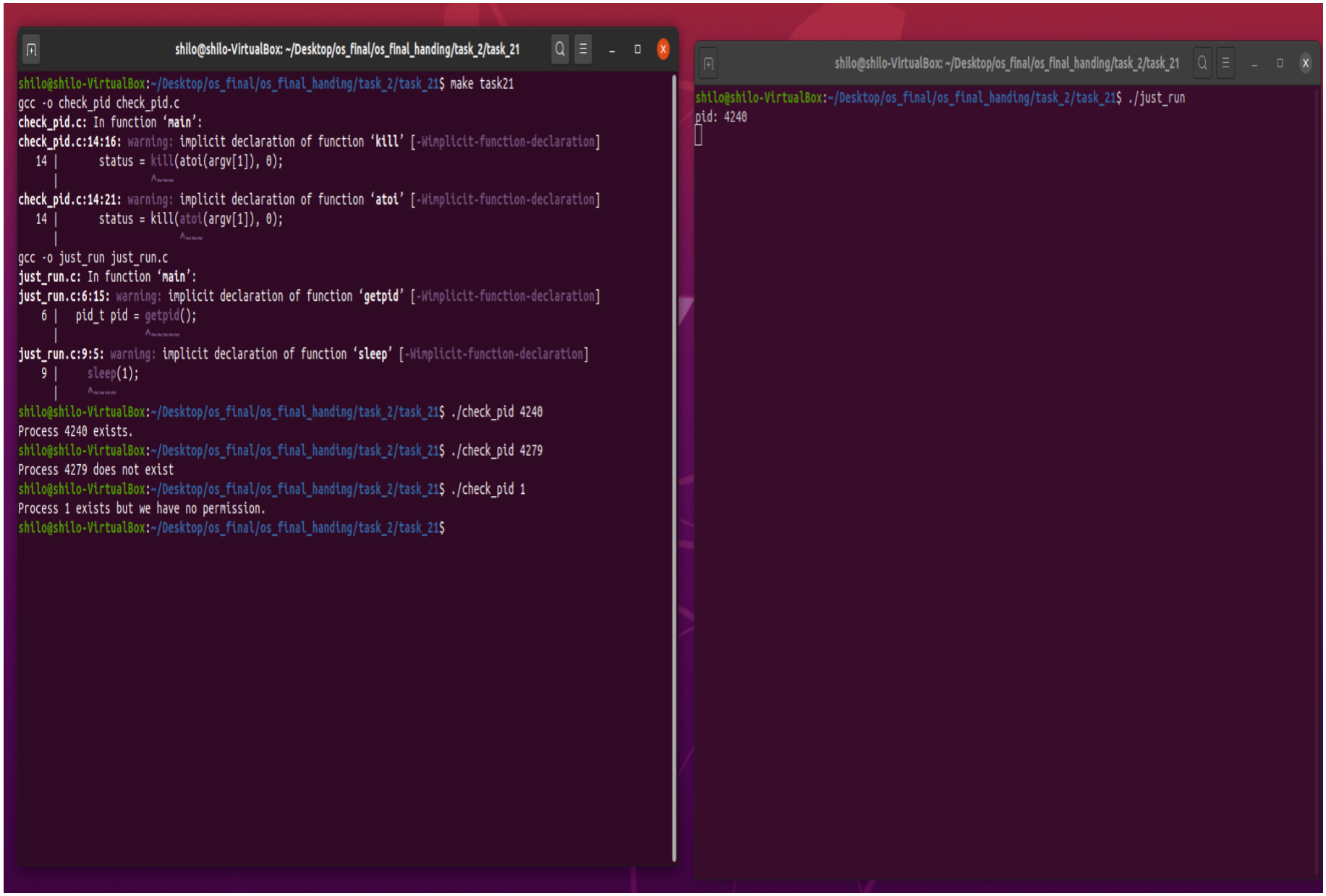
מה עליכם לבצע:
0) כתבו תכנית בשם **check_pid.c** המקבלת פרמטר יחיד pid והפלט של התכנית הוא :
If EPERM, Process <pid> exists but we have no permission.
If ESRCH, Process <pid> does not exist.
If kill is successful, Process <pid> exists.

Running example: check_pid 2003
Process 2003 exists.

המקורות שלמדתי מהם:
https://stackoverflow.com/questions/3658668/implementing-einval-eperm-esrch-in-kill
https://man7.org/linux/man-pages/man2/kill.2.html

הסבר על התמונה הבאה:

הכנתי קובץ שמדפיס את הפרוסס איידי שלו ואז רק רץ בלולאה אינסופית כך שיהיה לי פרוסס שאוכל לבדוק שקיים.
הרצתי 3 ריצות שמראות את שלושת ההדפסות האפשריות:
הראשון שהפרוסס קיים.
השני שהפרוסס לא קיים.
השלישי שהפרוסס לא מרשה גישה אליו.


shilo@shilo-VirtualBox: ~/Desktop/os_final/os_final_handing/task_2/task_21
shilo@shilo-VirtualBox:~/Desktop/os_final/os_final_handing/task_2/task_21$ make task21
gcc -o check_pid check_pid.c
check_pid.c: In function 'main':
check_pid.c:14:16: warning: implicit declaration of function 'kill' [-Wimplicit-function-declaration]
14 | status = kill(atoi(argv[1]), 0);
check_pid.c:14:21: warning: implicit declaration of function 'atoi' [-Wimplicit-function-declaration]
14 | status = kill(atoi(argv[1]), 0);
gcc -o just_run just_run.c
just_run.c: In function 'main':
just_run.c:6:15: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
6 | pid_t pid = getpid();
just_run.c:9:5: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
9 | sleep(1);
shilo@shilo-VirtualBox:~/Desktop/os_final/os_final_handing/task_2/task_21$ ./check_pid 4240
Process 4240 exists.
shilo@shilo-VirtualBox:~/Desktop/os_final/os_final_handing/task_2/task_21$ ./check_pid 4279
Process 4279 does not exist
shilo@shilo-VirtualBox:~/Desktop/os_final/os_final_handing/task_2/task_21$ ./check_pid 1
Process 1 exists but we have no permission.
shilo@shilo-VirtualBox:~/Desktop/os_final/os_final_handing/task_2/task_21$
shilo@shilo-VirtualBox: ~/Desktop/os_final/os_final_handing/task_2/task_21
shilo@shilo-VirtualBox:~/Desktop/os_final/os_final_handing/task_2/task_21$ ./just_run
pid: 4240

1) הציעו 2 שיטות נוספות לבדיקה הנ"ל. פרטו יתרונות וחסרונות של 3 השיטות הנ"ל. כתבו את תשובתכם בקובץ pdf נפרד בשם **q21_<your_id>.pdf**

https://man7.org/linux/man-pages/man2/pidfd_send_signal.2.html

NAME top

pidfd_send_signal - send a signal to a process specified by a file descriptor

SYNOPSIS top

```
#include <signal.h>

int pidfd_send_signal(int pidfd, int sig, siginfo_t *info,
                      unsigned int flags);
```

DESCRIPTION top

The **pidfd_send_signal**() system call sends the signal *sig* to the target process referred to by *pidfd*, a PID file descriptor that refers to a process.

If the *info* argument points to a *siginfo_t* buffer, that buffer should be populated as described in rt_sigqueueinfo(2).

If the *info* argument is a NULL pointer, this is equivalent to specifying a pointer to a *siginfo_t* buffer whose fields match the values that are implicitly supplied when a signal is sent using kill(2):

ERRORS top

EBADF *pidfd* is not a valid PID file descriptor.

EINVAL *sig* is not a valid signal.

EINVAL The calling process is not in a PID namespace from which it can send a signal to the target process.

EINVAL *flags* is not 0.

EPERM The calling process does not have permission to send the signal to the target process.

EPERM *pidfd* doesn't refer to the calling process, and *info.si_code* is invalid (see rt_sigqueueinfo(2)).

ESRCH The target process does not exist (i.e., it has terminated and been waited on).

NOTES top

Currently, there is no glibc wrapper for this system call; call it using syscall(2).

PID file descriptors

The *pidfd* argument is a PID file descriptor, a file descriptor that refers to process. Such a file descriptor can be obtained in any of the following ways:

* by opening a */proc/[pid]* directory;

* using pidfd_open(2); or

* via the PID file descriptor that is returned by a call to clone(2) or clone3(2) that specifies the **CLONE_PIDFD** flag.

The **pidfd_send_signal**() system call allows the avoidance of race conditions that occur when using traditional interfaces (such as kill(2)) to signal a process. The problem is that the traditional interfaces specify the target process via a process ID (PID), with the result that the sender may accidentally send a signal to the wrong process if the originally intended target process has terminated and its PID has been recycled for another process. By contrast, a PID file descriptor is a stable reference to a specific process; if that process terminates, **pidfd_send_signal**() fails with the error **ESRCH**.

https://man7.org/linux/man-pages/man3/killpg.3.html

NAME top

killpg - send signal to a process group

SYNOPSIS top

```
#include <signal.h>

int killpg(int pgrp, int sig);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

```
killpg():
    _XOPEN_SOURCE >= 500
        || /* Since glibc 2.19: */ _DEFAULT_SOURCE
        || /* Glibc versions <= 2.19: */ _BSD_SOURCE
```

DESCRIPTION top

killpg() sends the signal *sig* to the process group *pgrp*. See signal(7) for a list of signals.

If *pgrp* is 0, **killpg**() sends the signal to the calling process's process group. (POSIX says: if *pgrp* is less than or equal to 1, the behavior is undefined.)

For the permissions required to send a signal to another process, see kill(2).

RETURN VALUE top

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS top

EINVAL *sig* is not a valid signal number.

EPERM The process does not have permission to send the signal to any of the target processes. For the required permissions, see kill(2).

ESRCH No process can be found in the process group specified by *pgrp*.

ESRCH The process group was given as 0 but the sending process does not have a process group.

While running this I found that it does not show that I have no permission, it just says it exists