

**BAHIR DAR UNIVERSITY INSTITUTE OF TECHNOLOGY**

**FACULITY OF COMPUTING**

**DEPARTMENT OF SOFTWARE ENGINEERING**

Operating system

System programming

Name Siyanet Getahun

ID 1307205

Submitted to: Lecture wendimu

Submission date: 17/11/2014

System call

1. What / Why / How, this system call?

2. Briefly describe about the list of parameters and flags

3. List the flags, their purpose with code implementation (give Example source code with output)

*Wait for one or more child process to change its state*

**What is wait4()?:**

The *wait4()* function suspends execution of the calling thread until status information from one of its terminated child processes is available, or until the delivery of a signal whose action is either to terminate the process or execute a signal handler. If status information is available prior to the call to *wait4()*, the return is immediate.

* In order to wait for the status of a terminated child process whose real or saved user ID is different from the calling process's real or effective user ID, your process must have the PROCMGR\_AID\_WAIT ability enabled.
* If the parent process sets the action for SIGCHLD to SIG\_IGN, its children won't enter the zombie state, and it won't be able to use the *wait\*()* functions to wait on their deaths.
* If the calling process is a guardian, it may wait on processes that are not its children

The *wait4()* function behaves the same as the [*wait()*](https://www.qnx.com/developers/docs/7.0.0/com.qnx.doc.neutrino.lib_ref/topic/w/wait.html) function when passed a *pid* argument of -1, and the *options* argument has a value of zero.

Only one of the *WIFEXITED*(*stat\_val*) and *WIFSIGNALED*(*stat\_val*) macros can evaluate to a nonzero value.

The following call:

wait3( *stat\_loc*, *options*, *resource\_usage* );

is equivalent to:

waitpid( (pid\_t)-1, *stat\_loc*, *options* );

except that on successful completion, if the *resource\_usage* argument to [*wait3()*](https://www.qnx.com/developers/docs/7.0.0/com.qnx.doc.neutrino.lib_ref/topic/w/wait3.html) isn't a NULL pointer, the rusage structure that the third argument points to is filled in for the child process identified by the return value.

It's also equivalent to:

wait4( (pid\_t)-1, *stat\_loc*, *options*, *resource\_usage* );

The *waitpid()* function is POSIX; *wait3()* and *wait4* are BSD extensions.

**How?:**

#include <sys/resource.h>

#include <sys/wait.h>

pid\_t wait4( pid\_t *pid*,

int \* *stat\_loc*,

int *options*,

struct rusage \* *resource\_usage* );

**parametrs:**

***pid***

The set of child processes that you want to get status information for:

* less than -1 — any child process whose process group ID is equal to the absolute value of *pid*.
* -1 — any child process
* 0 — any child process whose process group ID is equal to that of the calling process.
* greater than 0 — the single child process with this ID.

***stat\_loc***

NULL, or a pointer a location where the function can store the terminating status of the child process.

***options***

A combination of zero or more of the following flags:

* WCONTINUED — return the status for any child that was stopped and has been continued.
* WNOHANG — return immediately if there are no children to wait for.
* WNOWAIT — keep the process in a waitable state. This doesn't affect the state of the process; the process may be waited for again after this call completion.
* WSTOPPED — wait for and return the process status of any child that has stopped because it received a signal.
* WUNTRACED — report the status of a stopped child process. In QNX Neutrino, this is the same as WSTOPPED.

***resource\_usage***

NULL, or a pointer to a rusage structure where the function can store information about resource usage.

**Returns:**

If successful, *wait4()* returns the process ID of the terminating child process. If *wait4()* was invoked with WNOHANG set in *options*, it has at least one child process specified by *pid* for which status is not available, and status is not available for any process specified by *pid*, a value of zero is returned. On delivery of a signal *waitpid()* returns -1, and errno set to EINTR.

**Errors:**

**ECHILD**

The calling process has no existing unwaited-for child processes that meet the criteria set by *pid*.

**EINTR**

The function was interrupted by a signal. The value of the location pointed to by *stat\_loc* is undefined.

**EINVAL**

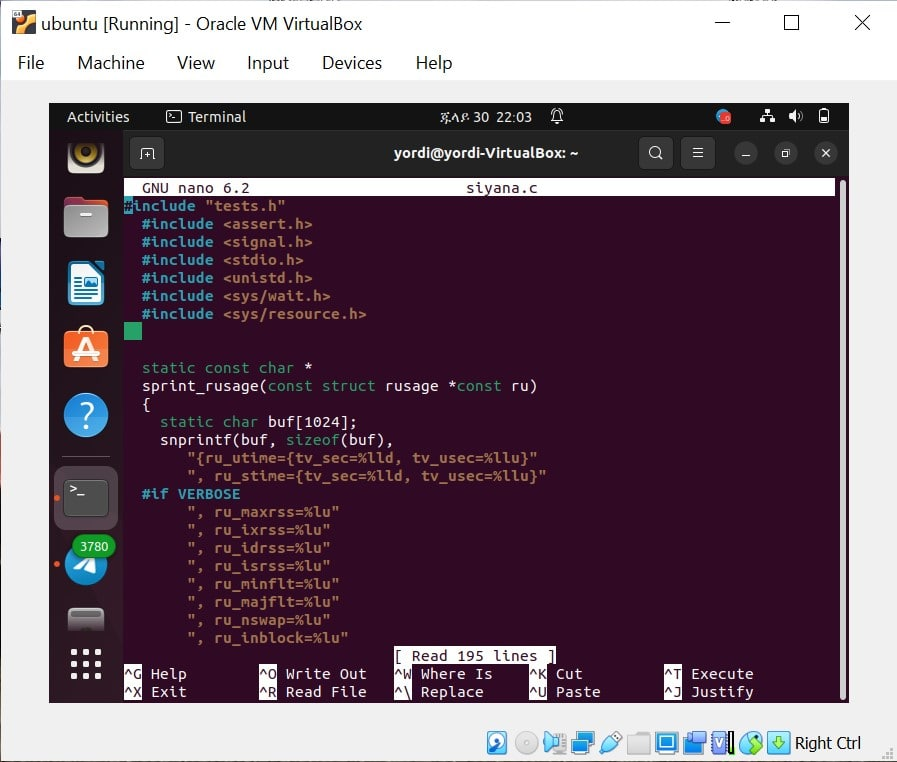
The value of the *options* argument isn't valid.

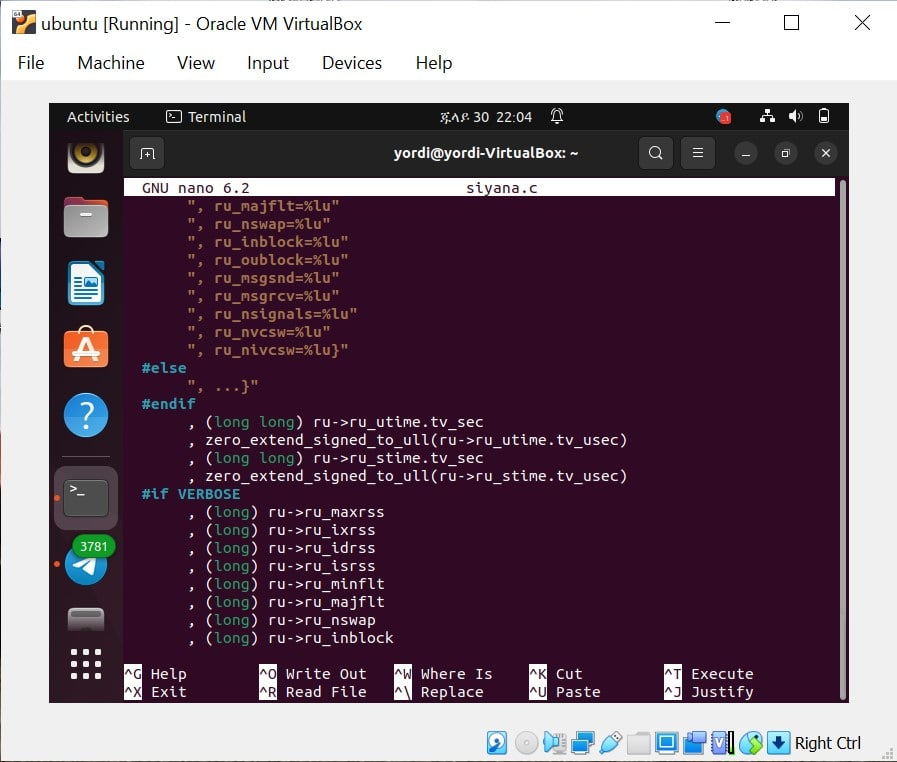
**EPERM**

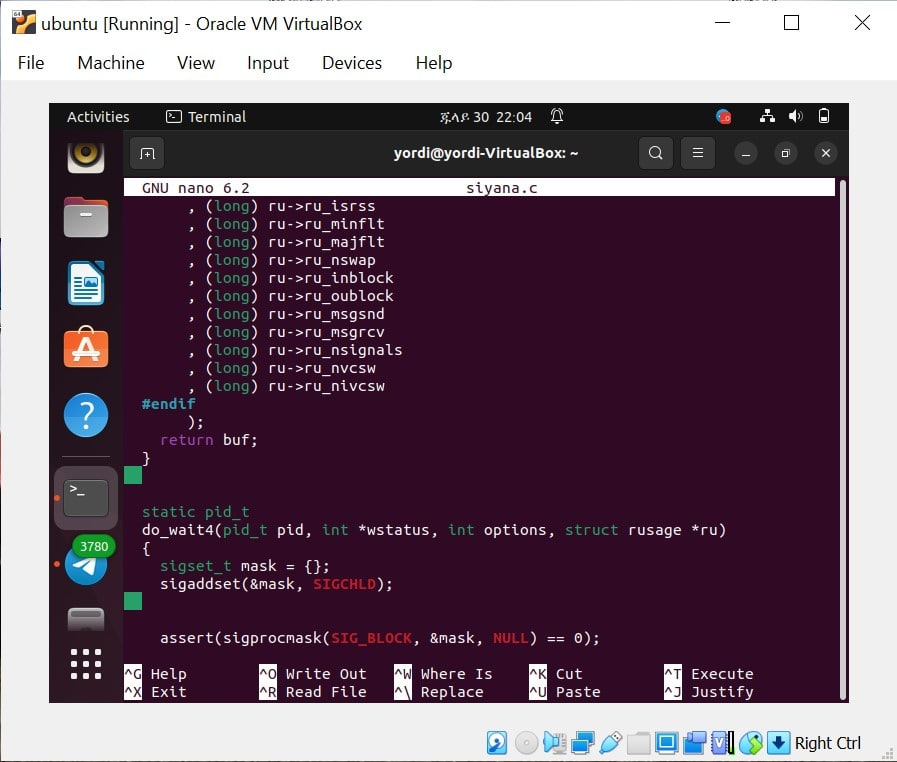
The calling process doesn't have the required permission;

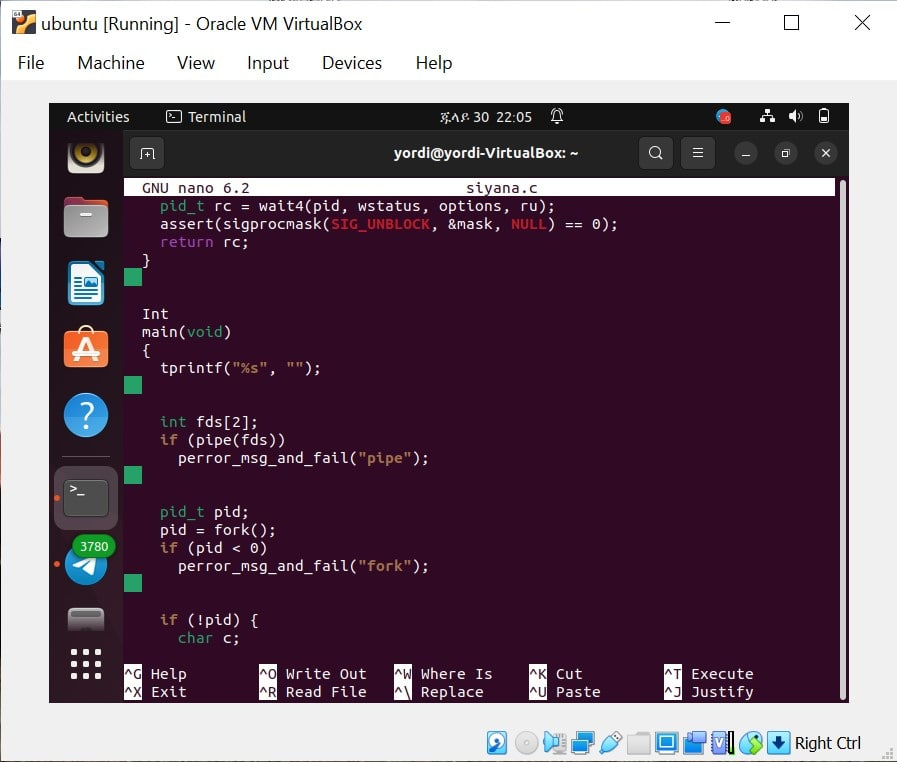
The implementation code

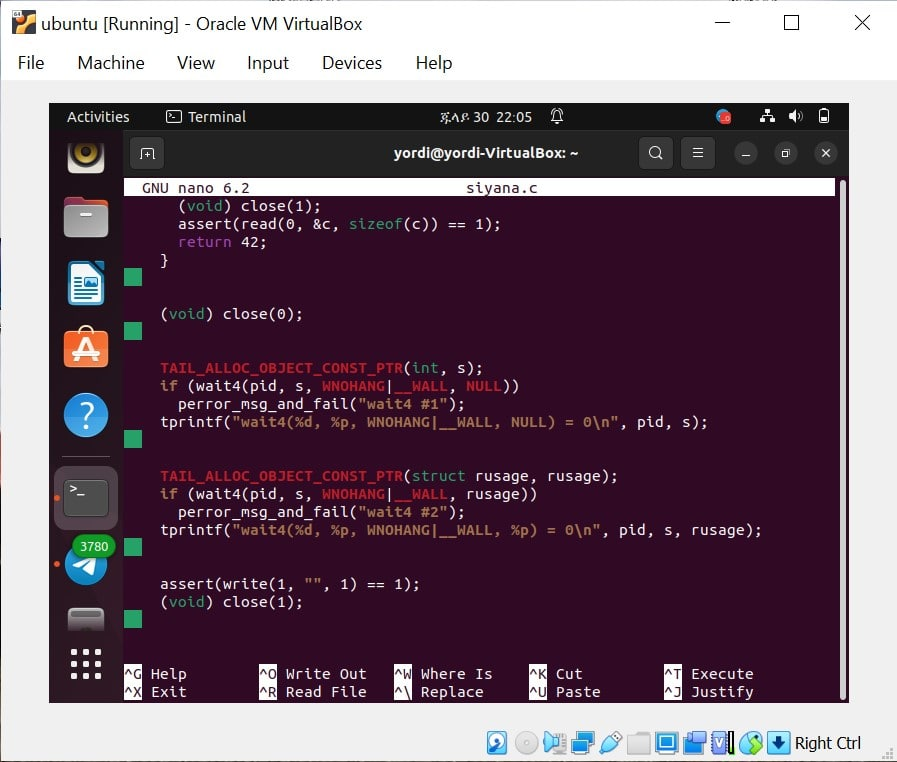
|  |
| --- |
| #include "tests.h" |
| #include <assert.h> |
| #include <signal.h> |
| #include <stdio.h> |
| #include <unistd.h> |
| #include <sys/wait.h> |
| #include <sys/resource.h> |
|  |
| static const char \* |
| sprint\_rusage(const struct rusage \*const ru) |
| { |
| static char buf[1024]; |
| snprintf(buf, sizeof(buf), |
| "{ru\_utime={tv\_sec=%lld, tv\_usec=%llu}" |
| ", ru\_stime={tv\_sec=%lld, tv\_usec=%llu}" |
| #if VERBOSE |
| ", ru\_maxrss=%lu" |
| ", ru\_ixrss=%lu" |
| ", ru\_idrss=%lu" |
| ", ru\_isrss=%lu" |
| ", ru\_minflt=%lu" |
| ", ru\_majflt=%lu" |
| ", ru\_nswap=%lu" |
| ", ru\_inblock=%lu" |
| ", ru\_oublock=%lu" |
| ", ru\_msgsnd=%lu" |
| ", ru\_msgrcv=%lu" |
| ", ru\_nsignals=%lu" |
| ", ru\_nvcsw=%lu" |
| ", ru\_nivcsw=%lu}" |
| #else |
| ", ...}" |
| #endif |
| , (long long) ru->ru\_utime.tv\_sec |
| , zero\_extend\_signed\_to\_ull(ru->ru\_utime.tv\_usec) |
| , (long long) ru->ru\_stime.tv\_sec |
| , zero\_extend\_signed\_to\_ull(ru->ru\_stime.tv\_usec) |
| #if VERBOSE |
| , (long) ru->ru\_maxrss |
| , (long) ru->ru\_ixrss |
| , (long) ru->ru\_idrss |
| , (long) ru->ru\_isrss |
| , (long) ru->ru\_minflt |
| , (long) ru->ru\_majflt |
| , (long) ru->ru\_nswap |
| , (long) ru->ru\_inblock |
| , (long) ru->ru\_oublock |
| , (long) ru->ru\_msgsnd |
| , (long) ru->ru\_msgrcv |
| , (long) ru->ru\_nsignals |
| , (long) ru->ru\_nvcsw |
| , (long) ru->ru\_nivcsw |
| #endif |
| ); |
| return buf; |
| } |
|  |
| static pid\_t |
| do\_wait4(pid\_t pid, int \*wstatus, int options, struct rusage \*ru) |
| { |
| sigset\_t mask = {}; |
| sigaddset(&mask, SIGCHLD); |
|  |
| assert(sigprocmask(SIG\_BLOCK, &mask, NULL) == 0); |
| pid\_t rc = wait4(pid, wstatus, options, ru); |
| assert(sigprocmask(SIG\_UNBLOCK, &mask, NULL) == 0); |
| return rc; |
| } |
|  |
| Int |
| main(void) |
| { |
| tprintf("%s", ""); |
|  |
| int fds[2]; |
| if (pipe(fds)) |
| perror\_msg\_and\_fail("pipe"); |
|  |
| pid\_t pid; |
| pid = fork(); |
| if (pid < 0) |
| perror\_msg\_and\_fail("fork"); |
|  |
| if (!pid) { |
| char c; |
| (void) close(1); |
| assert(read(0, &c, sizeof(c)) == 1); |
| return 42; |
| } |
|  |
| (void) close(0); |
|  |
| TAIL\_ALLOC\_OBJECT\_CONST\_PTR(int, s); |
| if (wait4(pid, s, WNOHANG|\_\_WALL, NULL)) |
| perror\_msg\_and\_fail("wait4 #1"); |
| tprintf("wait4(%d, %p, WNOHANG|\_\_WALL, NULL) = 0\n", pid, s); |
|  |
| TAIL\_ALLOC\_OBJECT\_CONST\_PTR(struct rusage, rusage); |
| if (wait4(pid, s, WNOHANG|\_\_WALL, rusage)) |
| perror\_msg\_and\_fail("wait4 #2"); |
| tprintf("wait4(%d, %p, WNOHANG|\_\_WALL, %p) = 0\n", pid, s, rusage); |
|  |
| assert(write(1, "", 1) == 1); |
| (void) close(1); |
|  |
| assert(do\_wait4(pid, s, 0, rusage) == pid); |
| assert(WIFEXITED(\*s) && WEXITSTATUS(\*s) == 42); |
| tprintf("wait4(%d, [{WIFEXITED(s) && WEXITSTATUS(s) == 42}], 0, %s)" |
| " = %d\n", pid, sprint\_rusage(rusage), pid); |
|  |
| pid = fork(); |
| if (pid < 0) |
| perror\_msg\_and\_fail("fork"); |
|  |
| if (!pid) { |
| (void) raise(SIGUSR1); |
| return 1; |
| } |
|  |
| assert(do\_wait4(pid, s, \_\_WALL, rusage) == pid); |
| assert(WIFSIGNALED(\*s) && WTERMSIG(\*s) == SIGUSR1); |
| tprintf("wait4(%d, [{WIFSIGNALED(s) && WTERMSIG(s) == SIGUSR1}]" |
| ", \_\_WALL, %s) = %d\n", pid, sprint\_rusage(rusage), pid); |
|  |
| if (pipe(fds)) |
| perror\_msg\_and\_fail("pipe"); |
| pid = fork(); |
| if (pid < 0) |
| perror\_msg\_and\_fail("fork"); |
|  |
| if (!pid) { |
| (void) close(1); |
| raise(SIGSTOP); |
| char c; |
| assert(read(0, &c, sizeof(c)) == 1); |
| return 0; |
| } |
|  |
| (void) close(0); |
|  |
| assert(do\_wait4(pid, s, WSTOPPED, rusage) == pid); |
| assert(WIFSTOPPED(\*s) && WSTOPSIG(\*s) == SIGSTOP); |
| tprintf("wait4(%d, [{WIFSTOPPED(s) && WSTOPSIG(s) == SIGSTOP}]" |
| ", WSTOPPED, %s) = %d\n", pid, sprint\_rusage(rusage), pid); |
|  |
| if (kill(pid, SIGCONT)) |
| perror\_msg\_and\_fail("kill(SIGCONT)"); |
|  |
| #if defined WCONTINUED && defined WIFCONTINUED |
| assert(do\_wait4(pid, s, WCONTINUED, rusage) == pid); |
| assert(WIFCONTINUED(\*s)); |
| tprintf("wait4(%d, [{WIFCONTINUED(s)}], WCONTINUED" |
| ", %s) = %d\n", pid, sprint\_rusage(rusage), pid); |
| #endif /\* WCONTINUED && WIFCONTINUED \*/ |
|  |
| assert(write(1, "", 1) == 1); |
| (void) close(1); |
|  |
| assert(do\_wait4(pid, s, 0, rusage) == pid); |
| assert(WIFEXITED(\*s) && WEXITSTATUS(\*s) == 0); |
| tprintf("wait4(%d, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0" |
| ", %s) = %d\n", pid, sprint\_rusage(rusage), pid); |
|  |
| assert(wait4(-1, s, WNOHANG|WSTOPPED|\_\_WALL, rusage) == -1); |
| tprintf("wait4(-1, %p, WNOHANG|WSTOPPED|\_\_WALL, %p) = -1 %s (%m)\n", |
| s, rusage, errno2name()); |
|  |
| tprintf("%s\n", "+++ exited with 0 +++"); |
| return 0; |
| } |

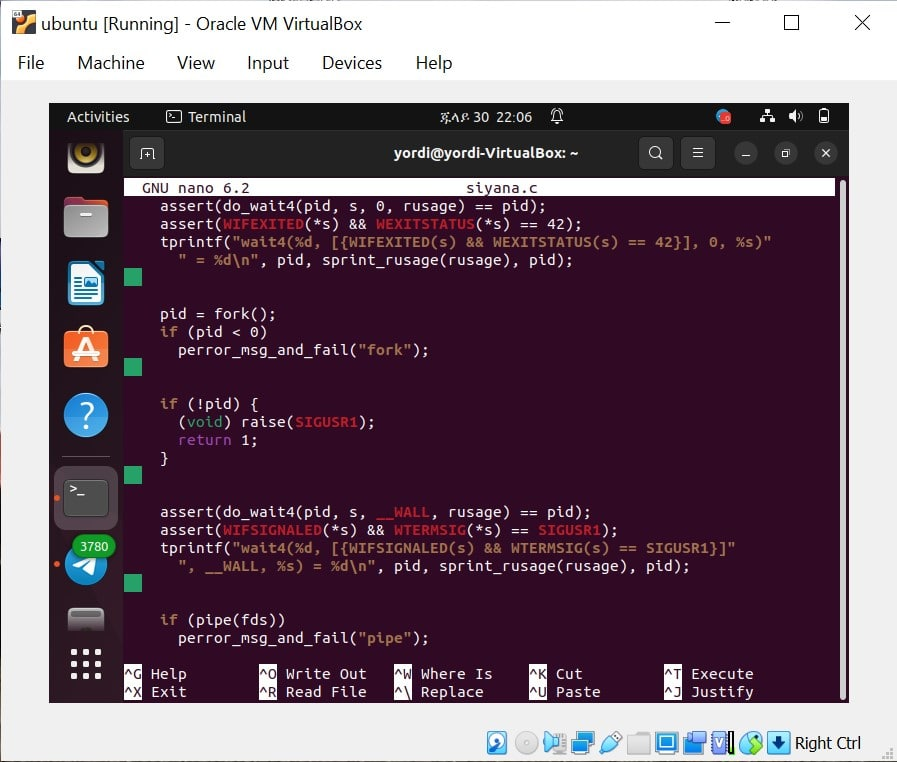


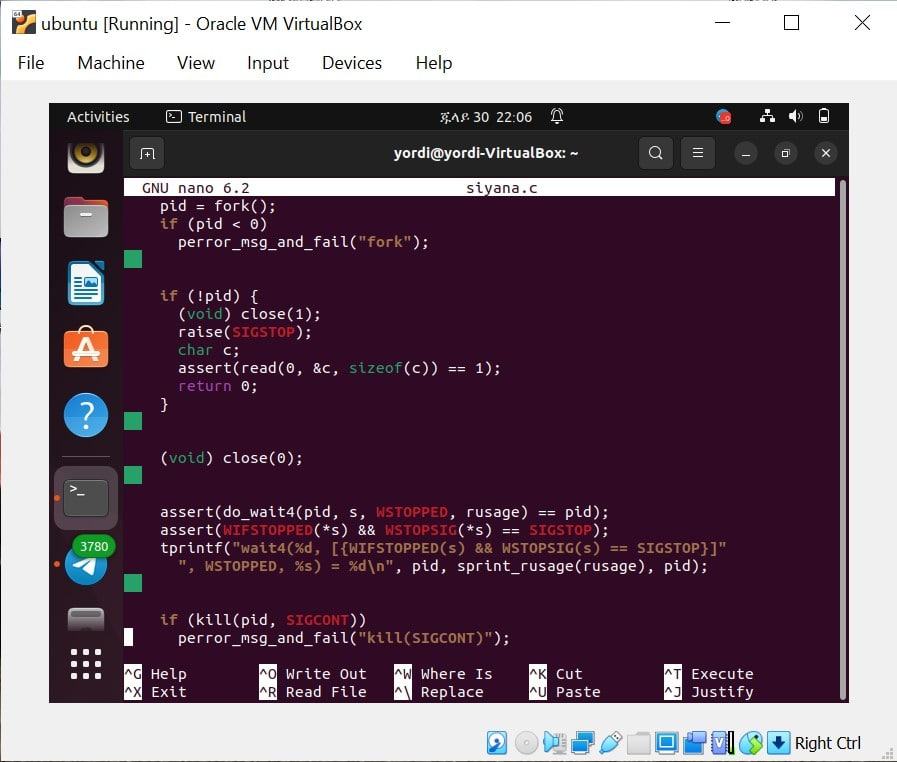


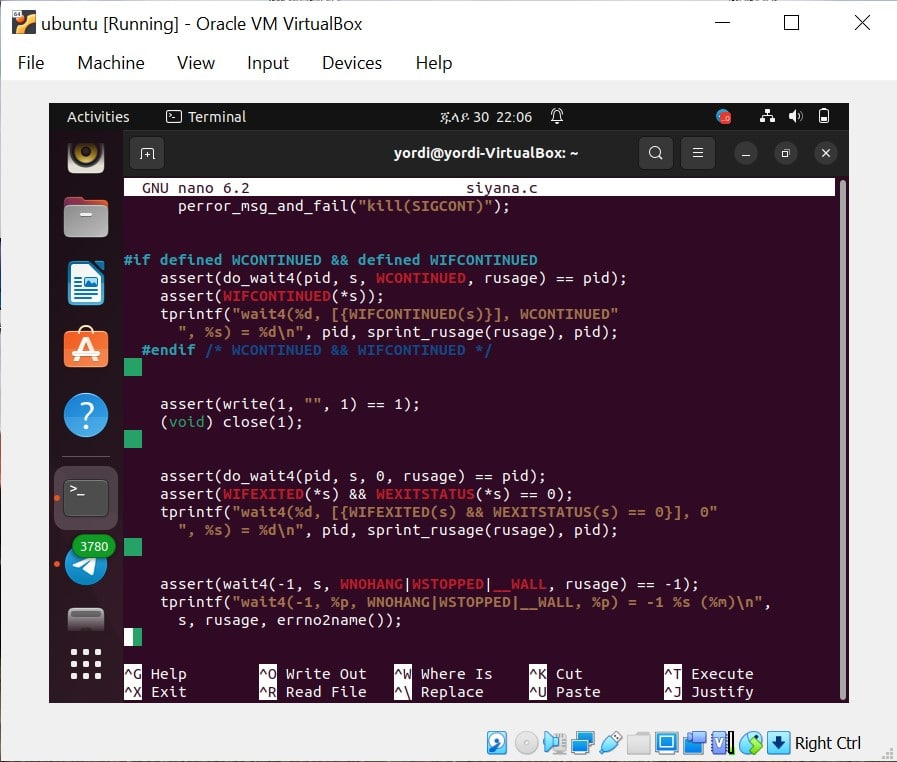


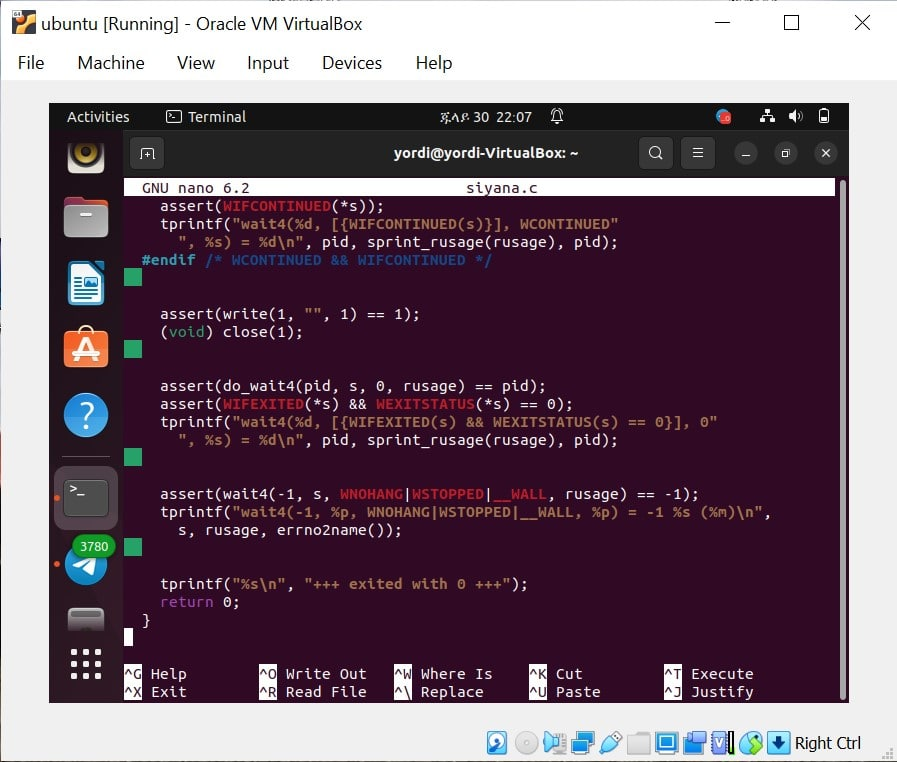












Error

