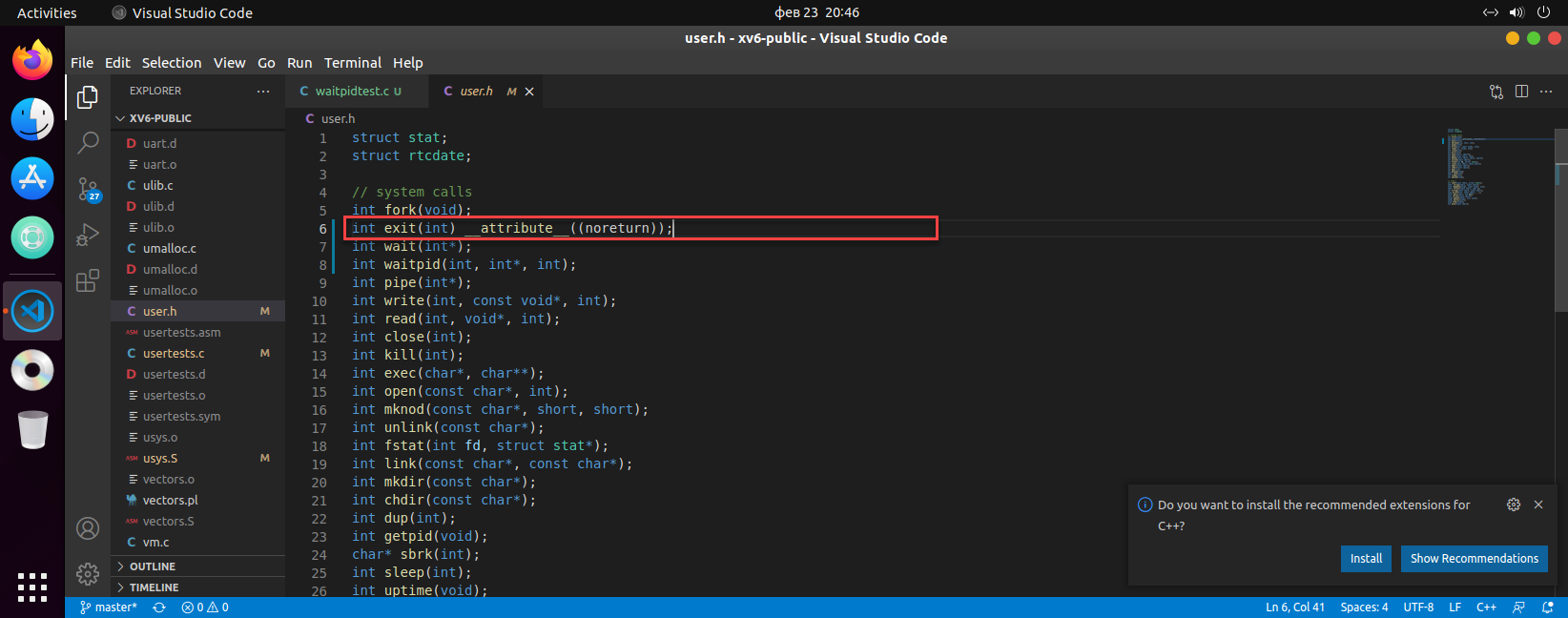
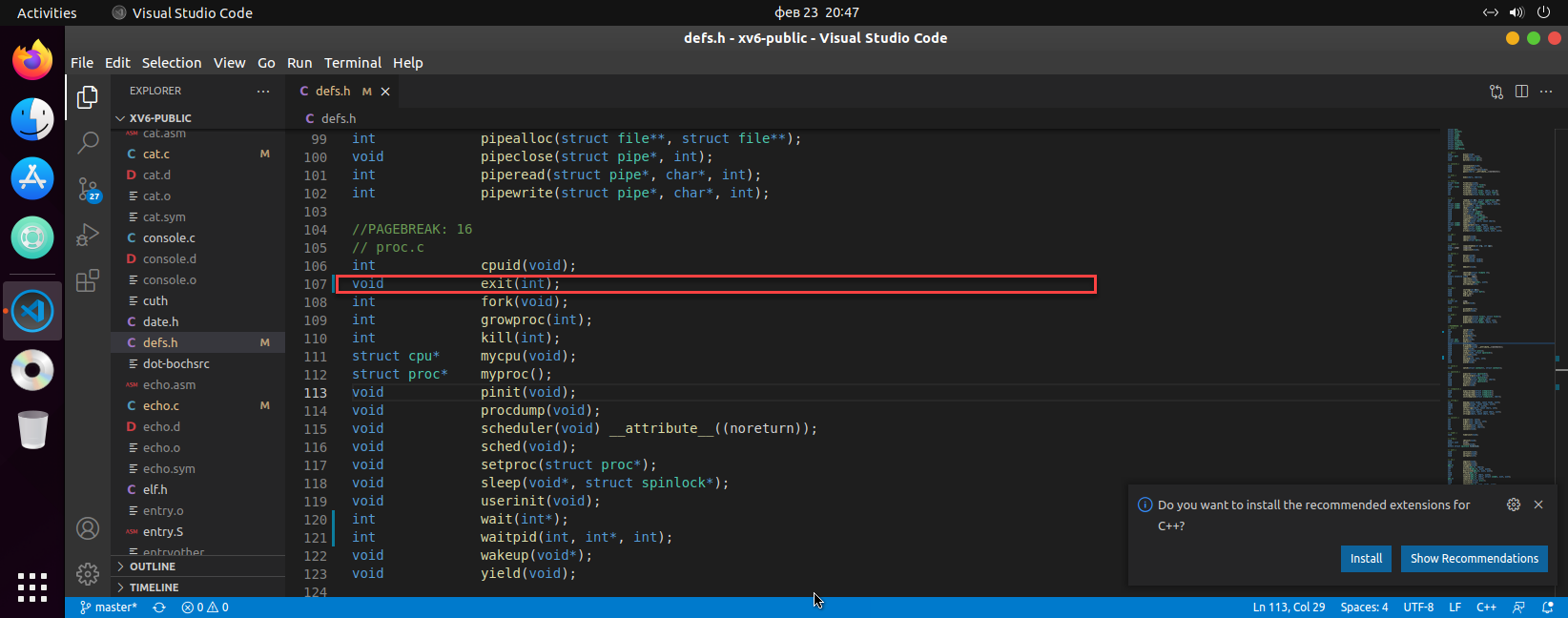
# Task 1: Change the exit system call signature to void exit(int status)

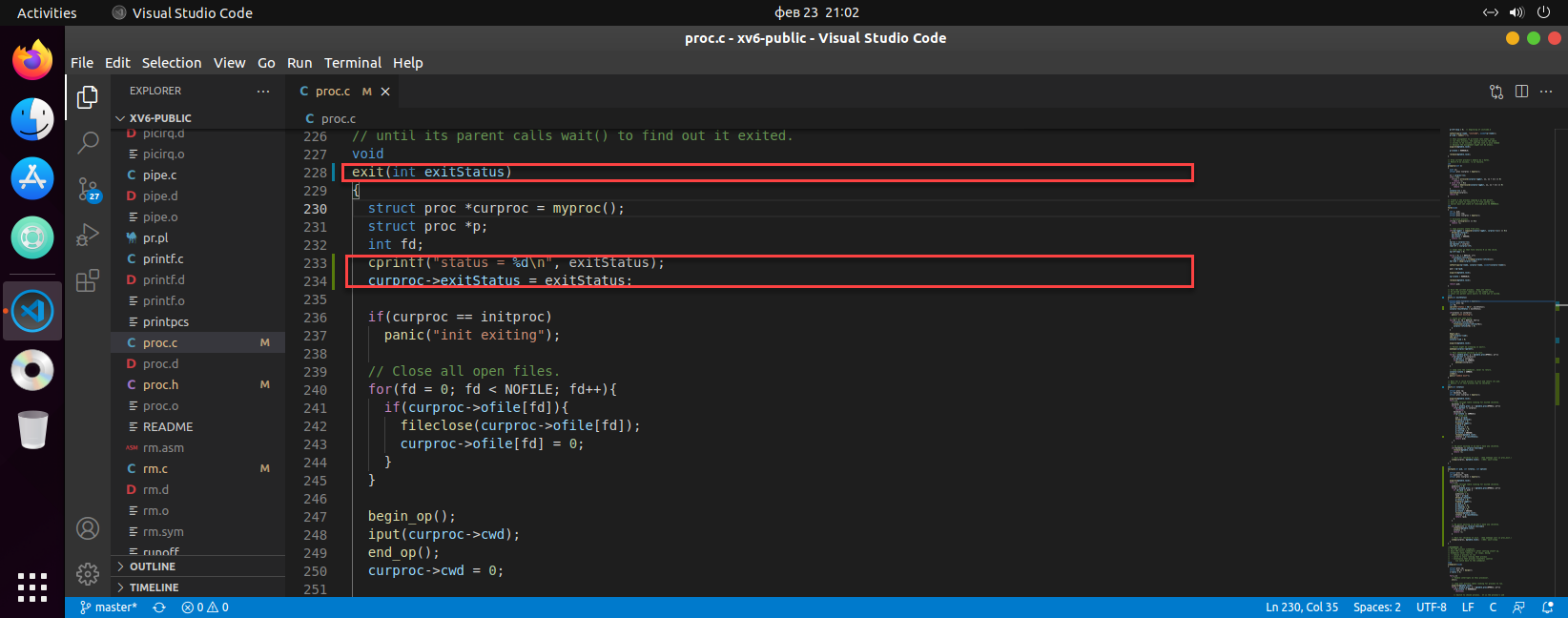
user.h



defs.h

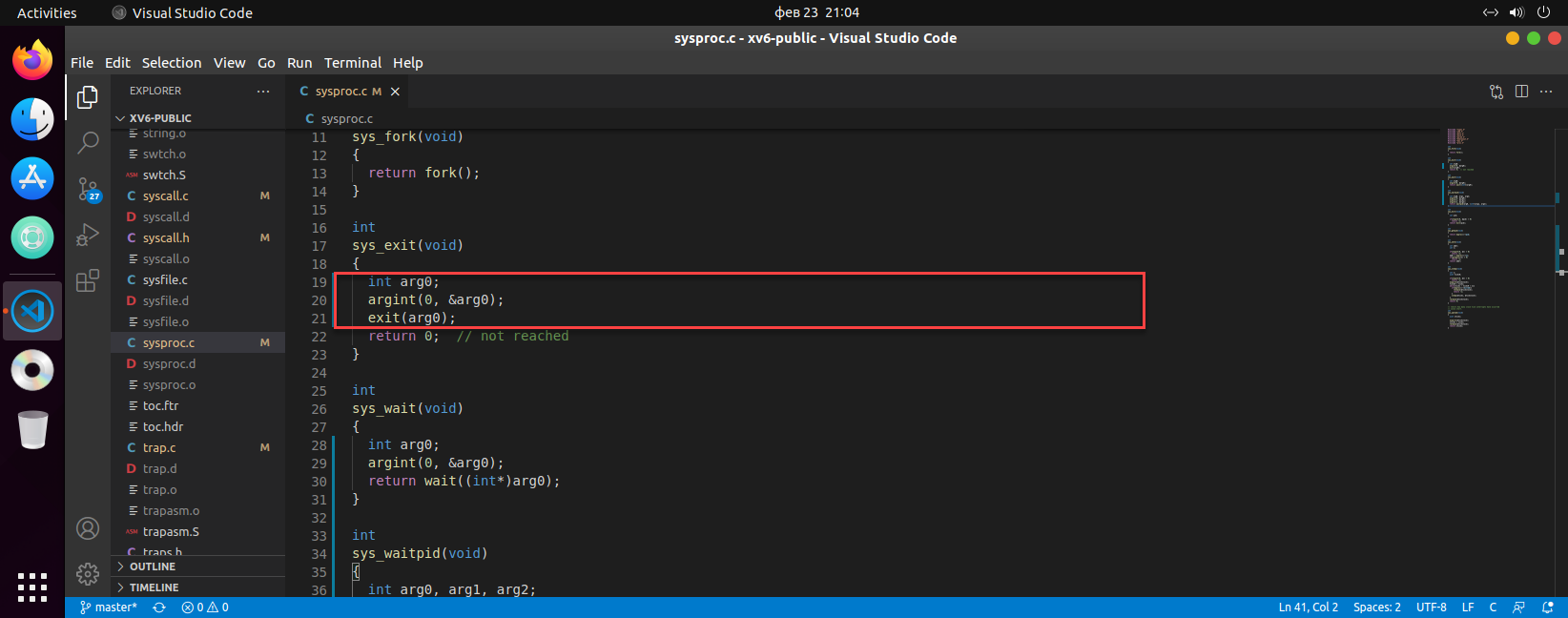


proc.c



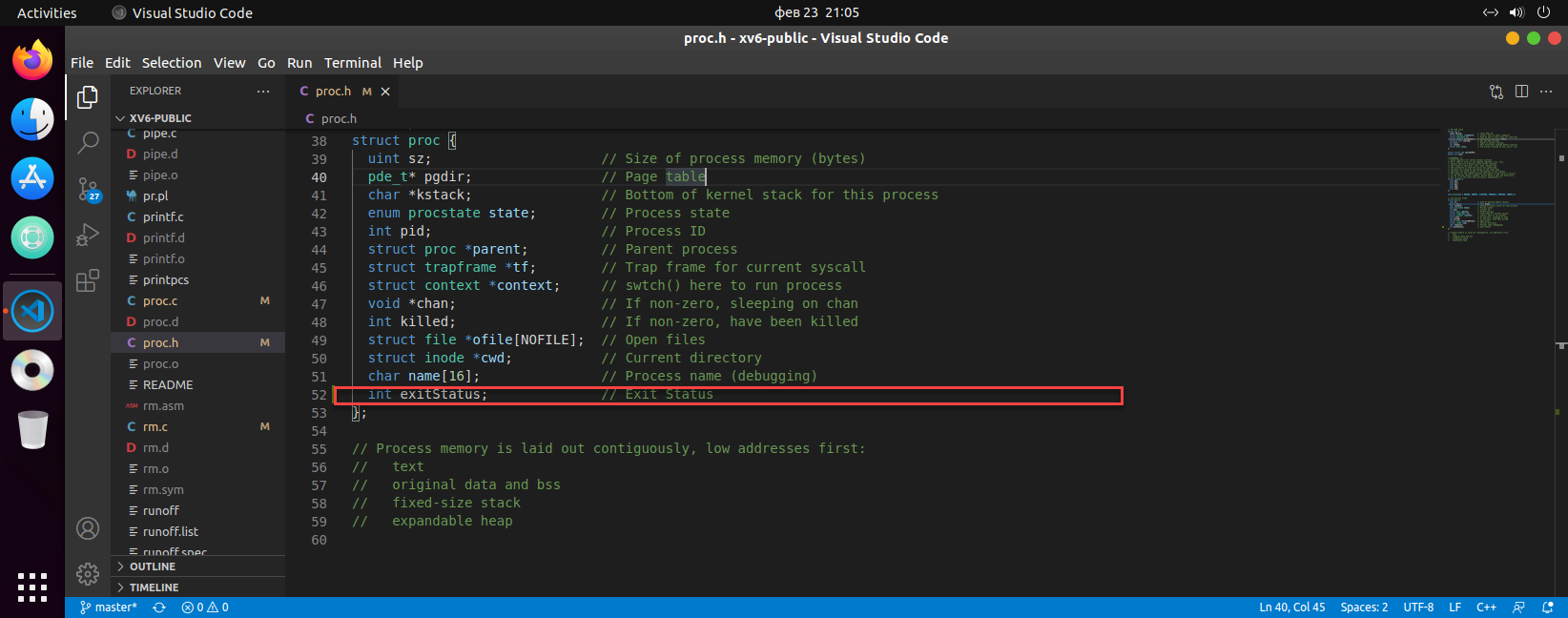
print parameter once with prefix and store it to proc structure.

sysproc.c



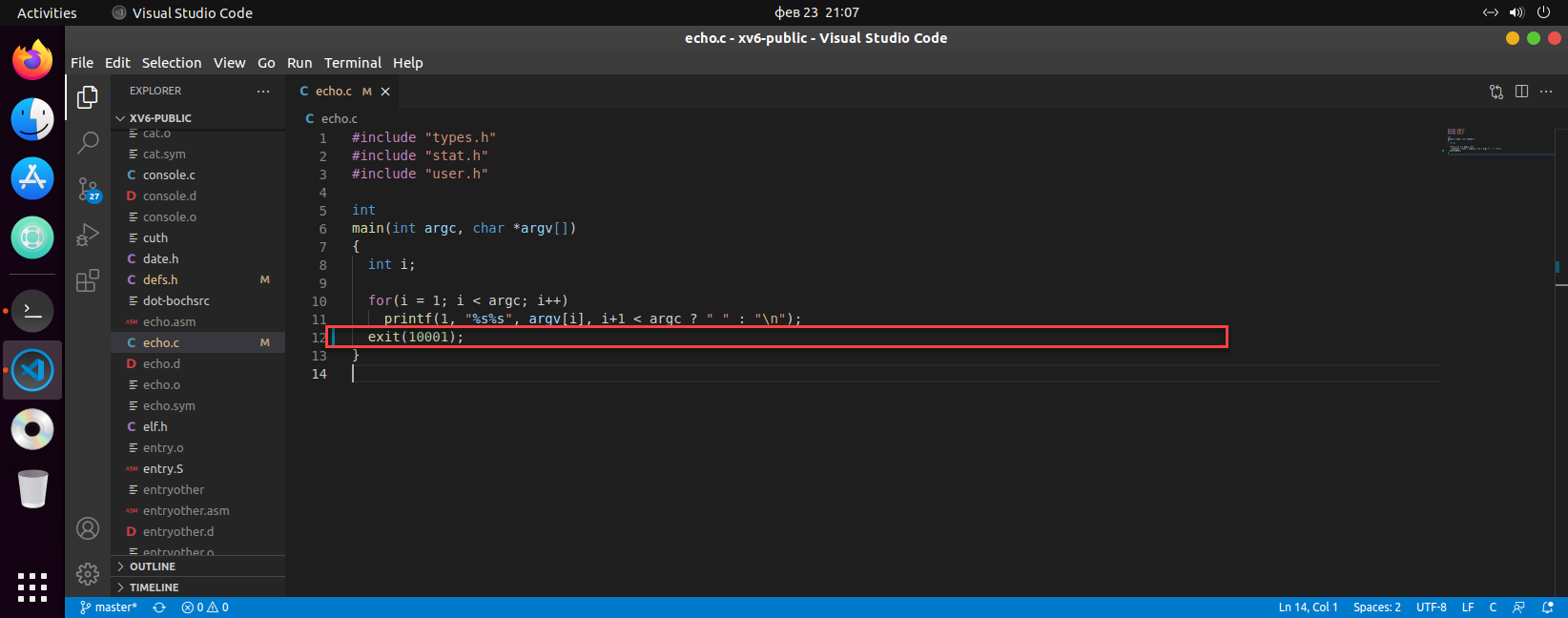
Get parameter by using argint method.

proc.h



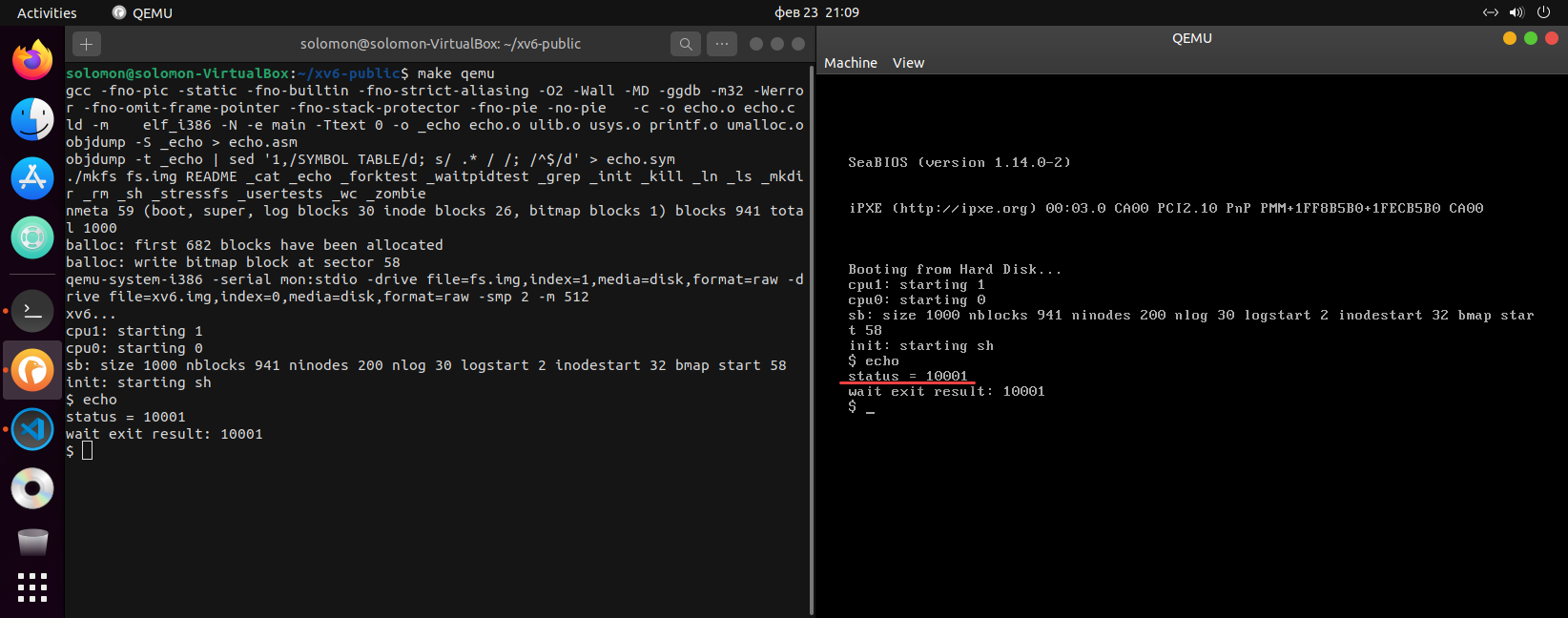
Change structure to make field to store new data – exitStatus

echo.h



Change all exit sys\_call (ex: exit(1001) from echo.h)

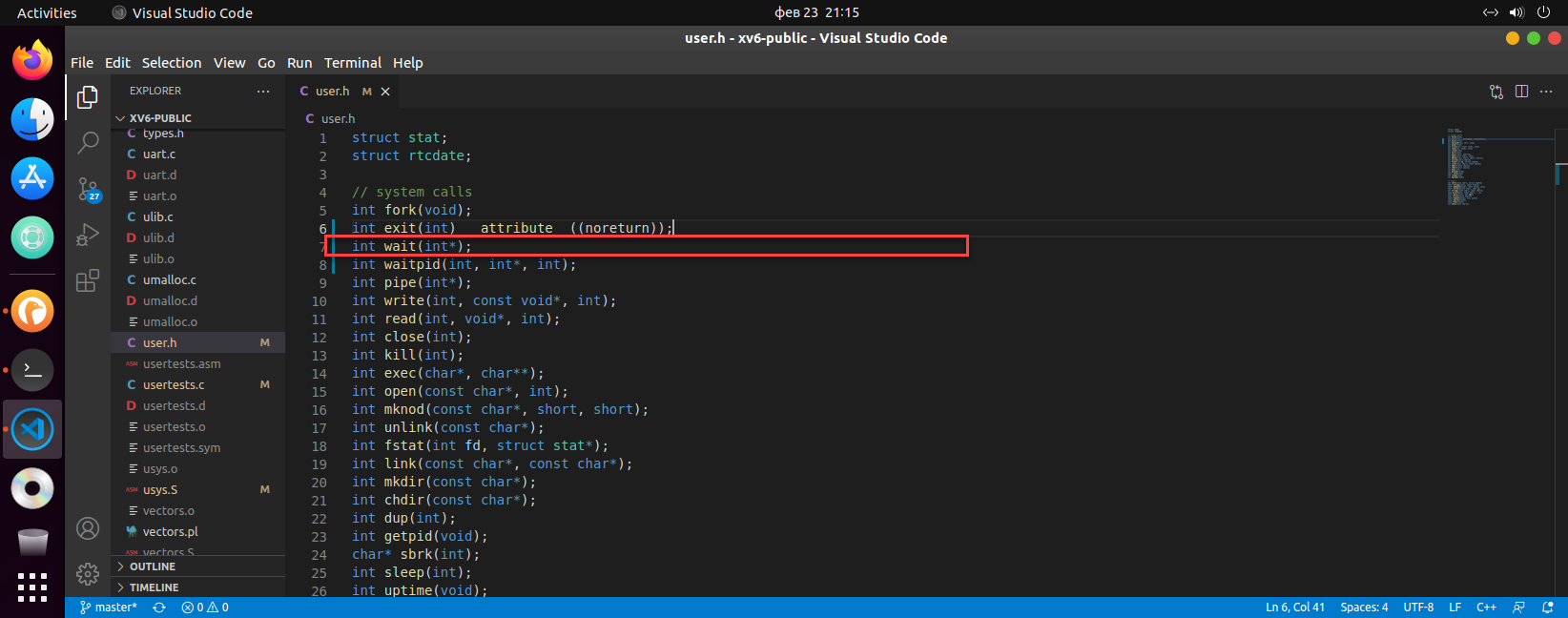
terminal



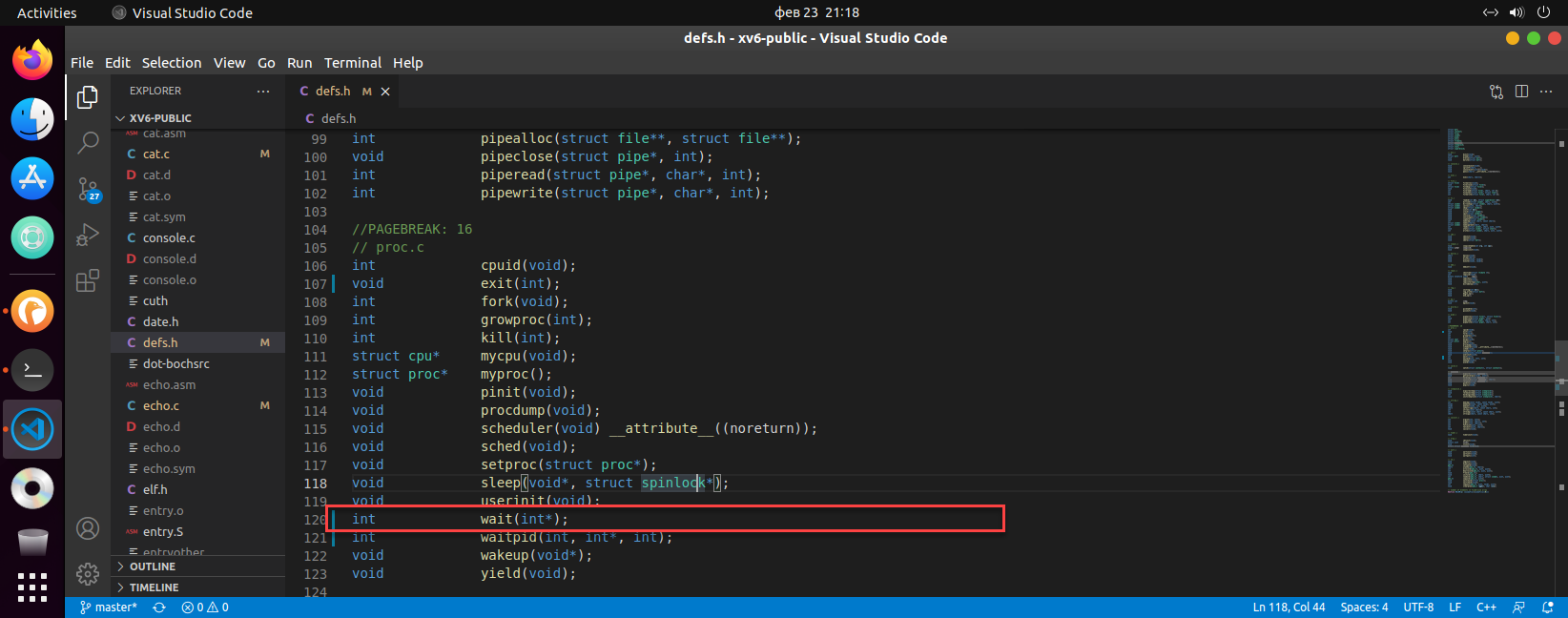
Exit code has saved correctly.

# Task 2: Update the wait system call signature to int wait(int \*status)

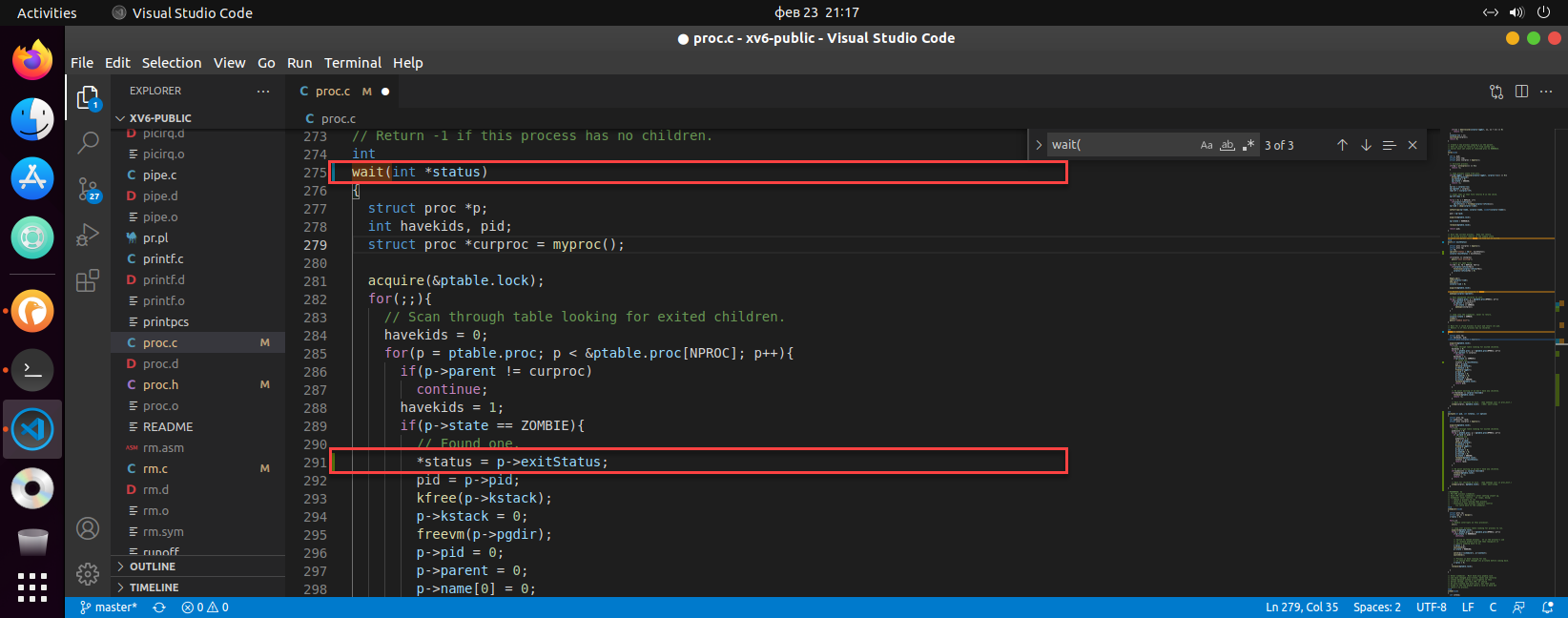
user.h



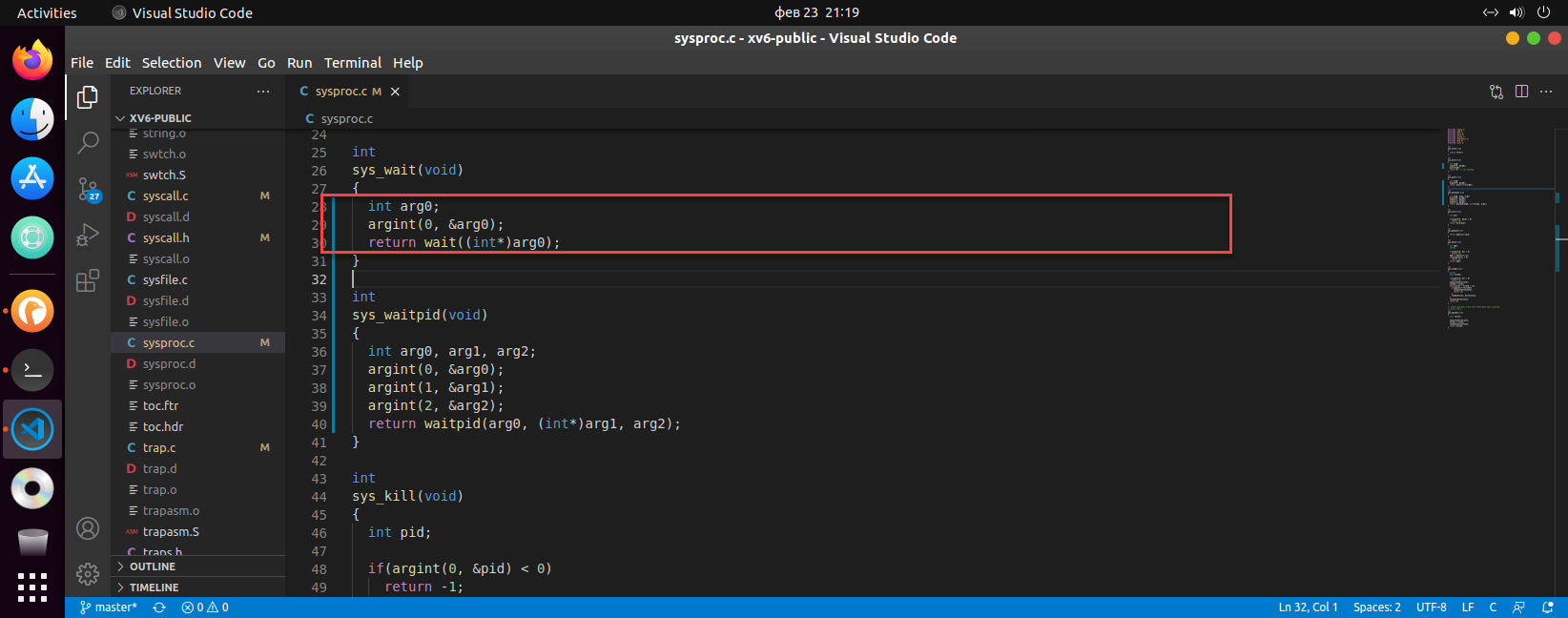
defs.h



proc.c

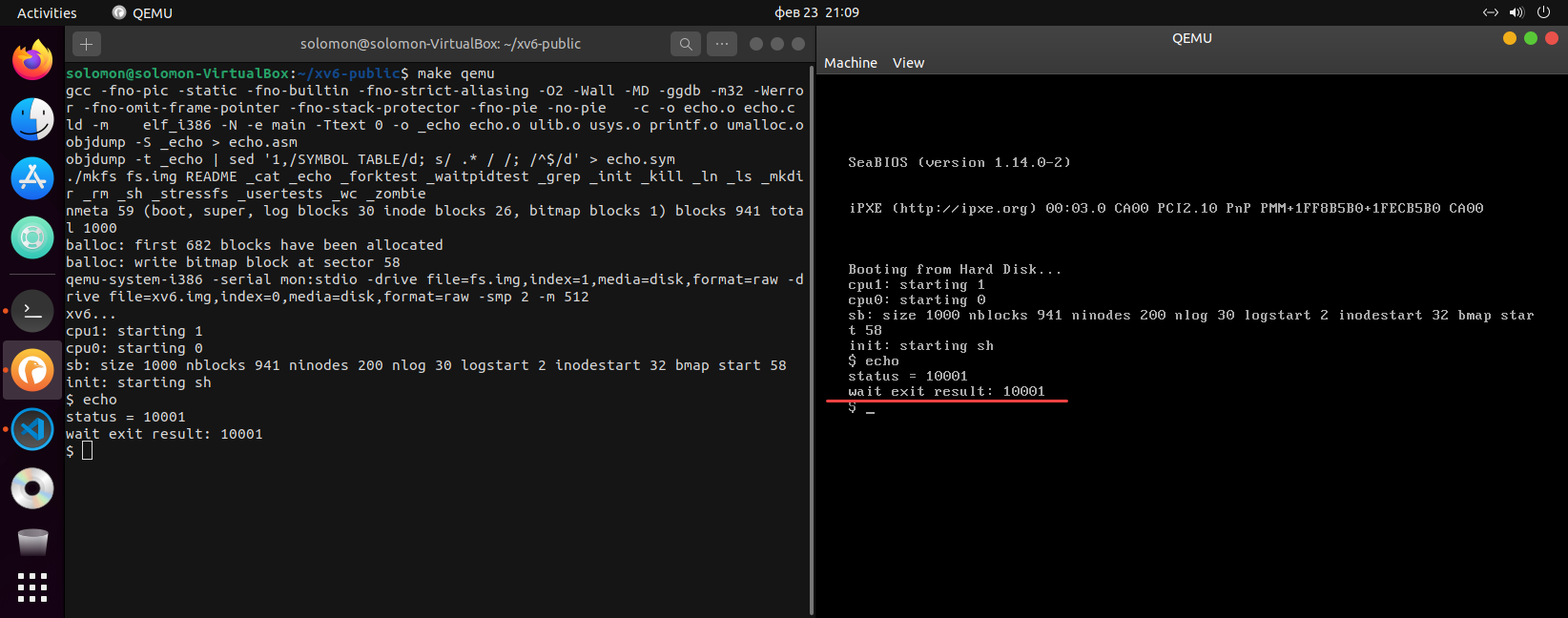


sysproc.c



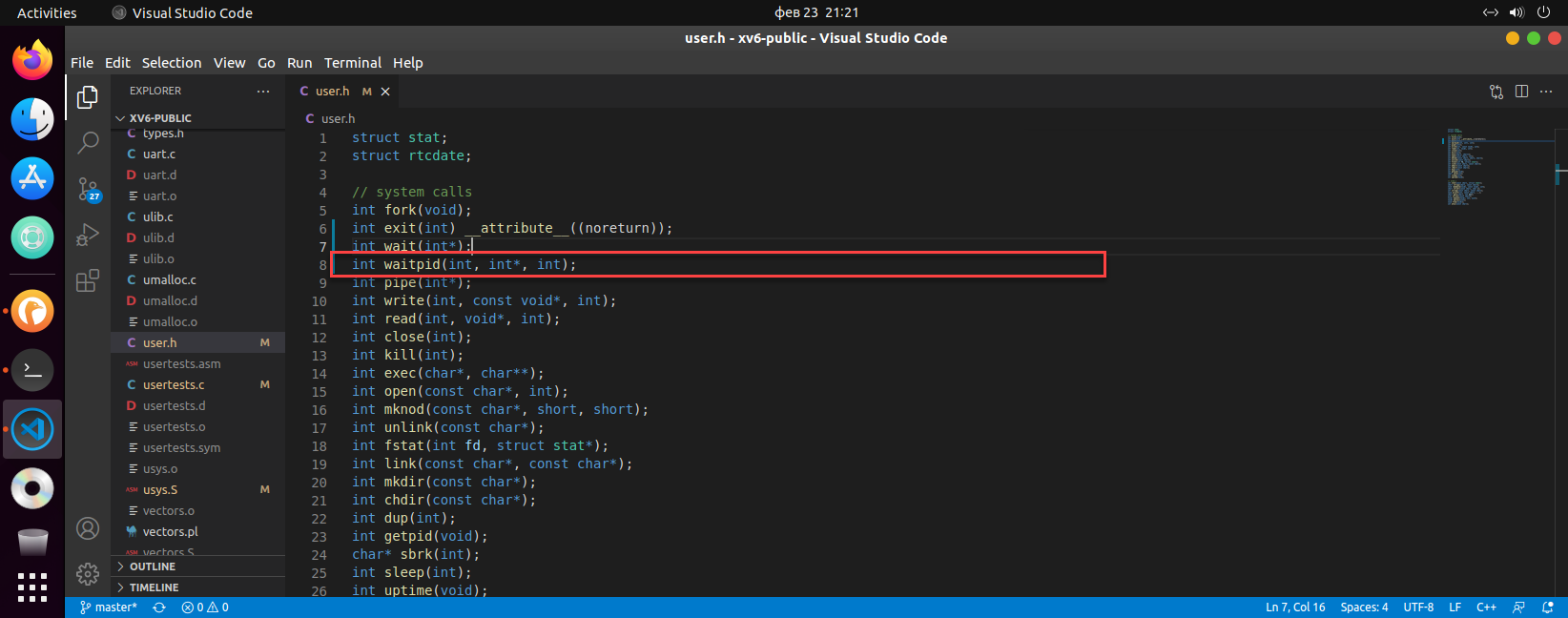
Get parameter as int pointer to return data from kernel to user mode.

terminal

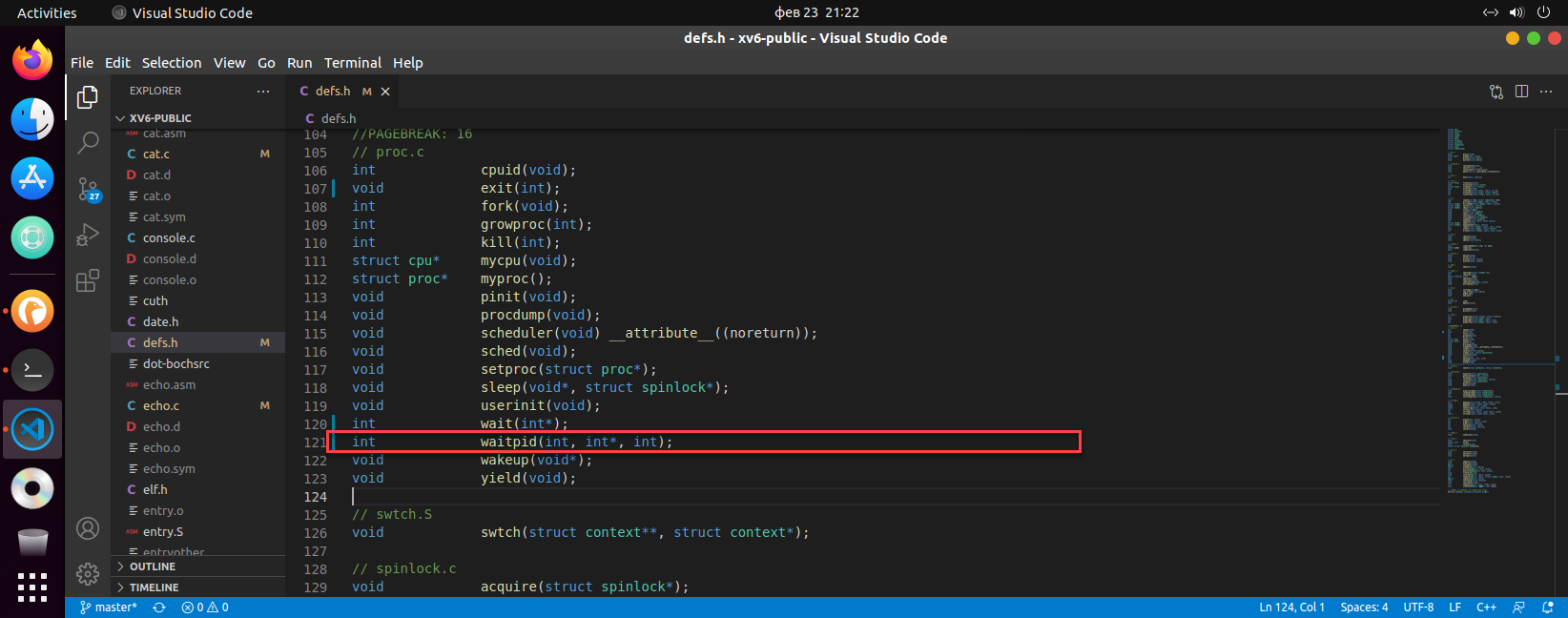


# Task 3: Add a waitpid system call: int waitpid(int pid, int \*status, int options)

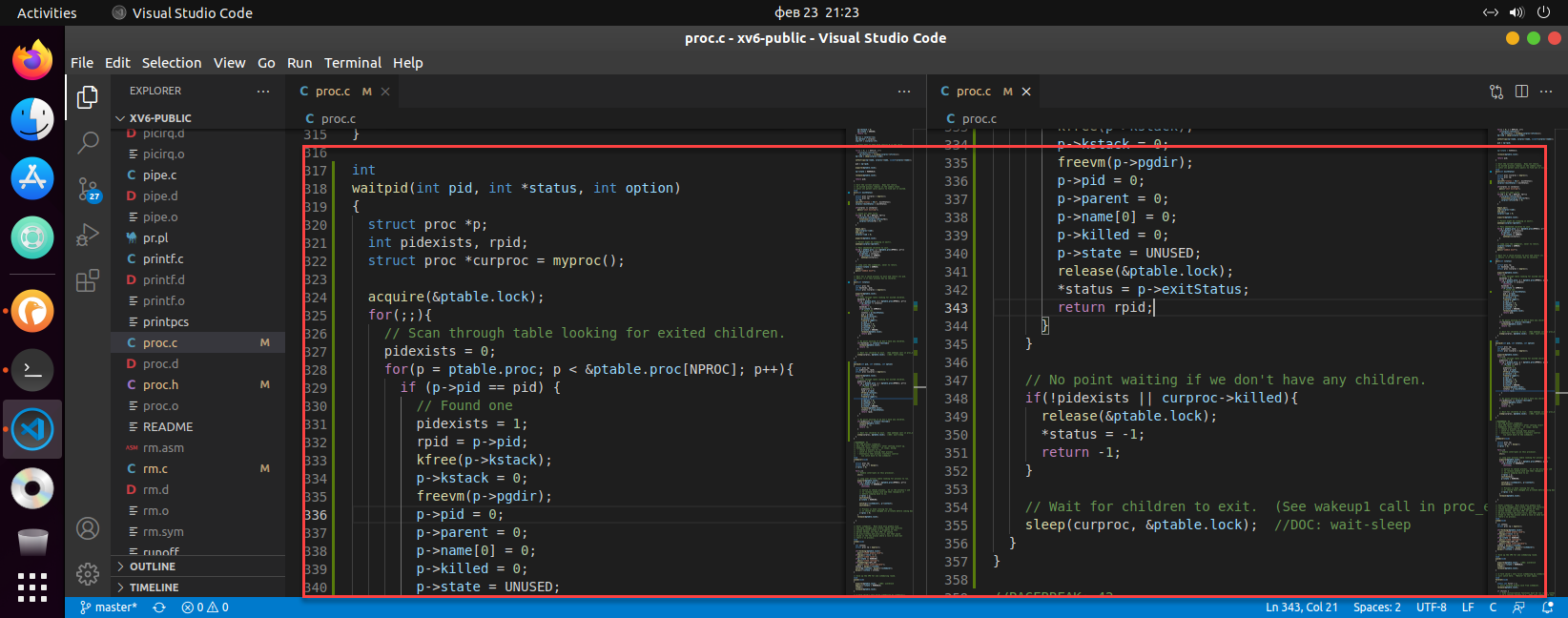
user.h



defs.h



proc.c

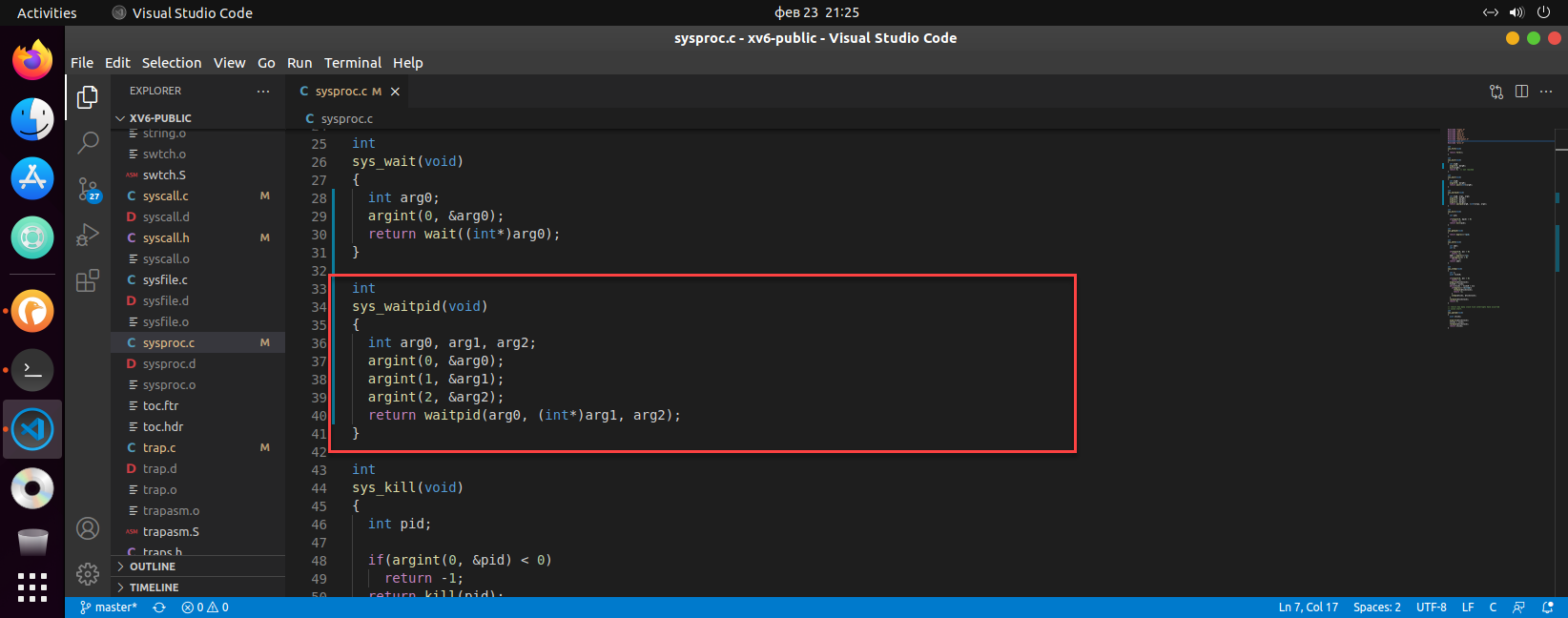


Add new waitpid system call

For proc.c file, compaire p->pid with pid parameter to select correct process.

If there is no matched one(when pidexists is 0), return -1.

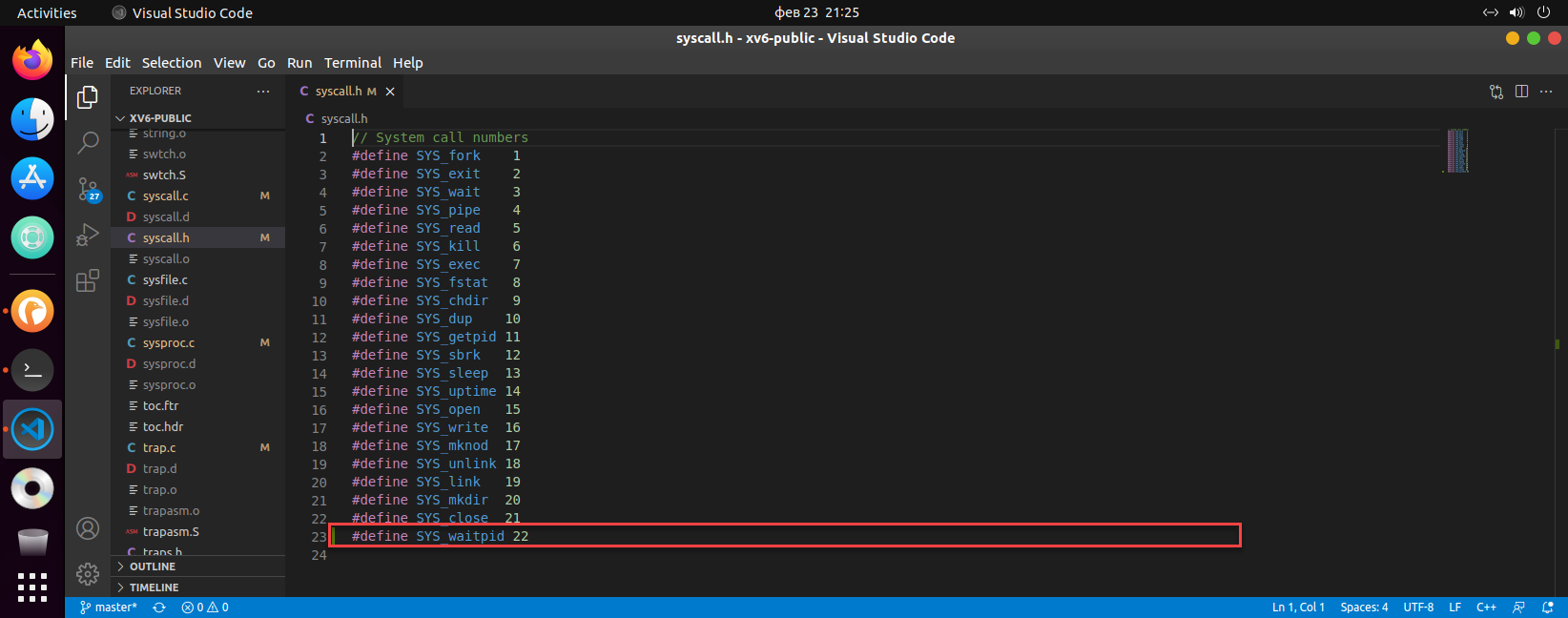
sysproc.c



There are 3 arguments for waitpid, int, int\*, and int.

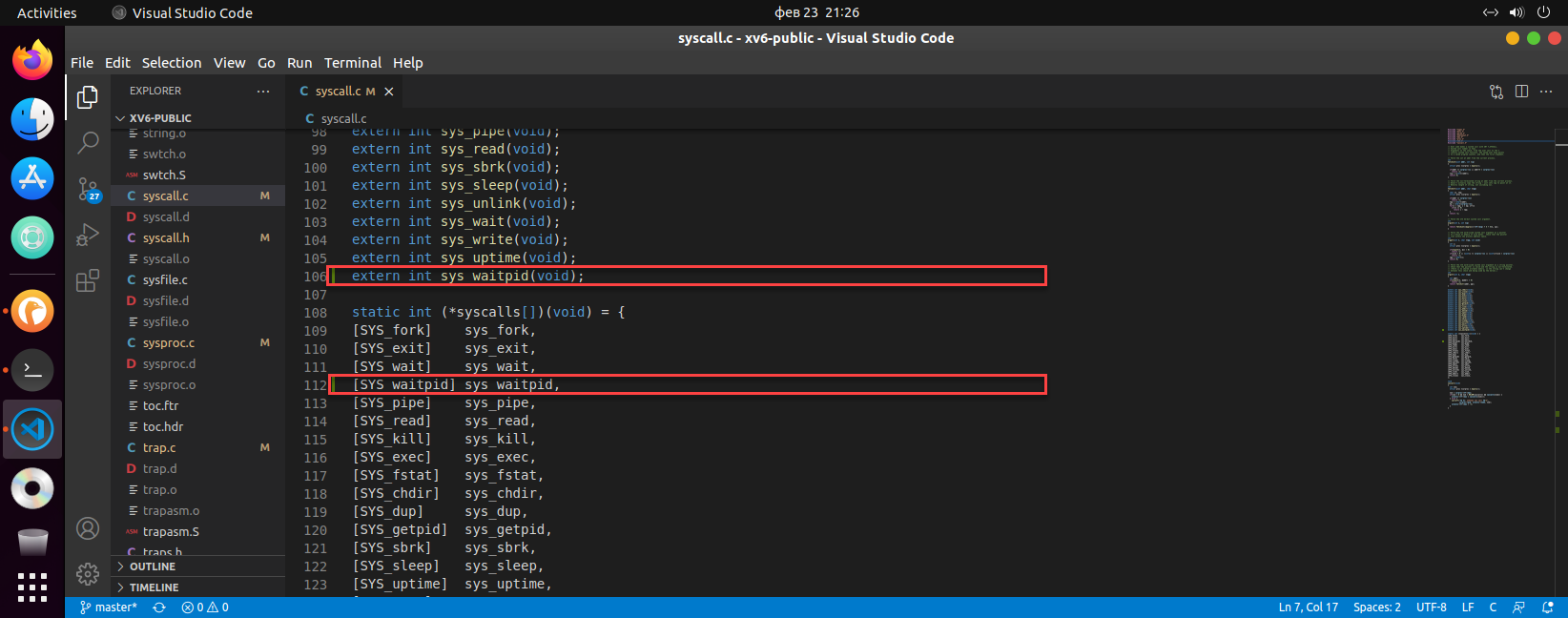
Retrieve these arguments by using argint method.

syscall.h



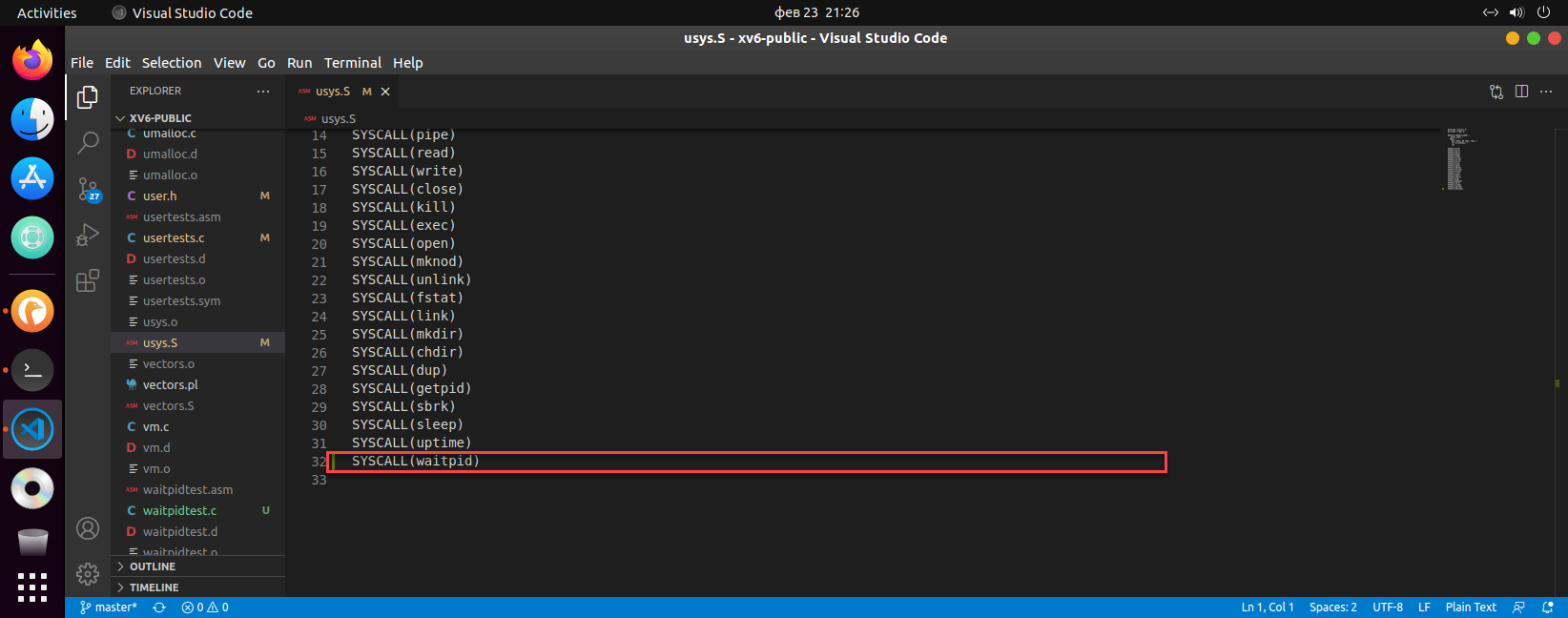
Add new constant – SYS\_waitpid

syscall.c



Add new syscall – sys\_waitpid

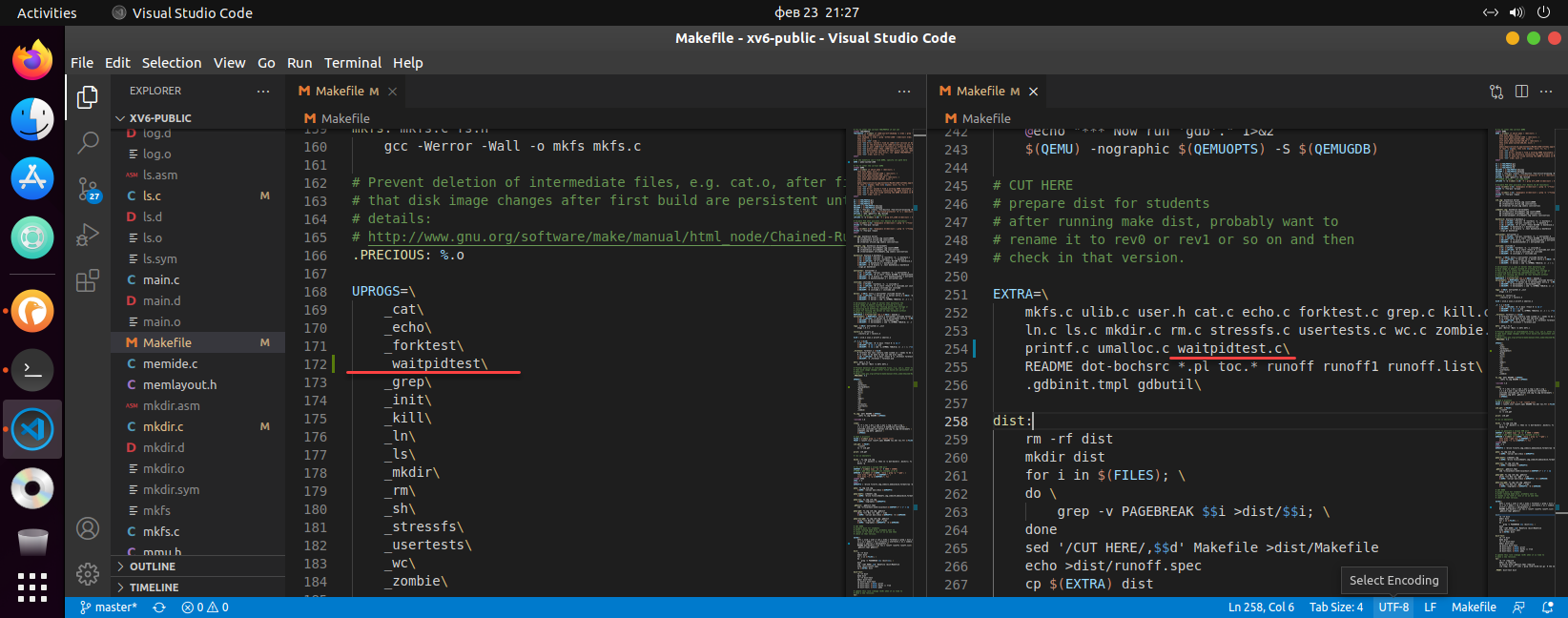
usys.S



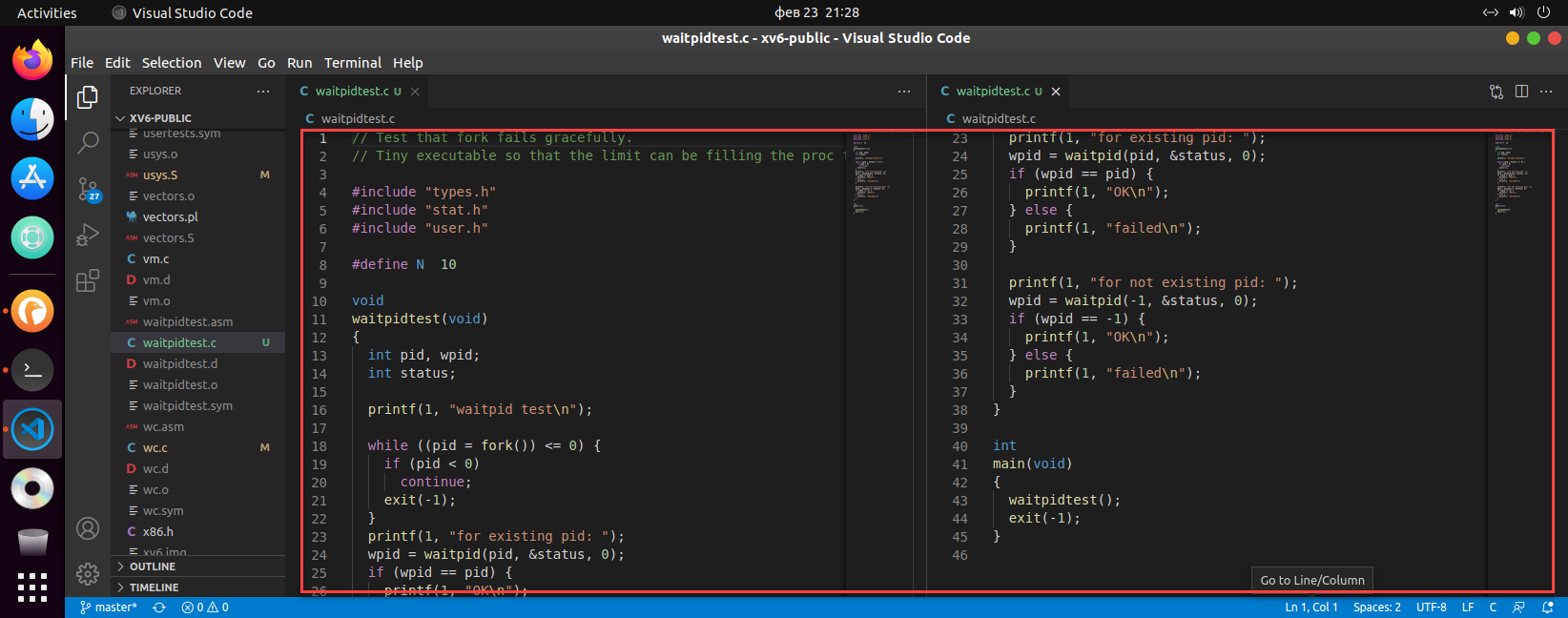
Add new call – waitpid

# Task 4: Write an example program to illustrate that your waitpid works

Makefile



waitpidtest.c



terminal

