ELE3021 [Operating System]

Project01 Wiki

Abegunde Adebowale Nathaniel: 2022055741

Table of Content:

1. **Introduction**
   This is a general overview of the project, talking about the tasks this project aims to achieve.
2. **Design and Implementation**
   This part covers the design process, how each step is implemented to yield the necessary result to make this project successful. It also covers how various codebases are edited and how these edits affect the implementation of the program.
3. **Result**
4. **Troubleshooting**
   Covers various problems encountered during the implementation of this project and my approach to solving them.

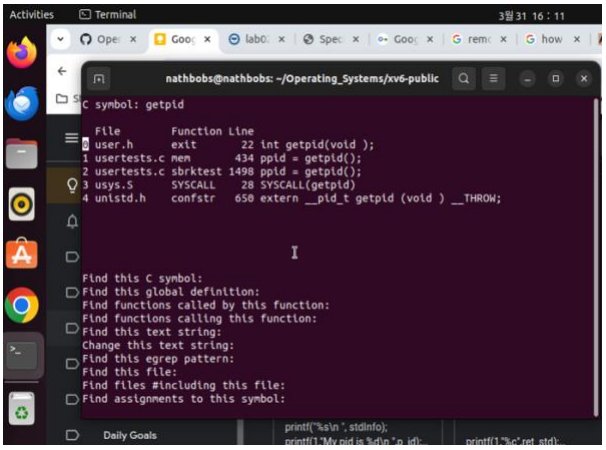
1. **Introduction:**

The user program is supposed to output the student number of the person who performed the task, the id of the process and the id of the grandparent process. This project is supposed to test my knowledge on process IDs and how to find them. Process IDs are responsible for executing tasks and managing resources in operating systems, making them very important. Process IDs are unique identifiers assigned to each processes and they are important because they;

1) Help separate one process from another, making the operating system to track and manage processes effectively
2) Since process IDs also have parent processes, there is a process tree where the parent process ID (PPID) is at the top, followed by the child processes. Sometimes, there is a grandparent process ID too as executed in this project.
3) When a process terminates, the process ID is also released helping the operating system to save on resources.

2. **Design and Implementation:**

To begin with, cscope was used to analyse **getpid()** system call in the xv6-public kernel.

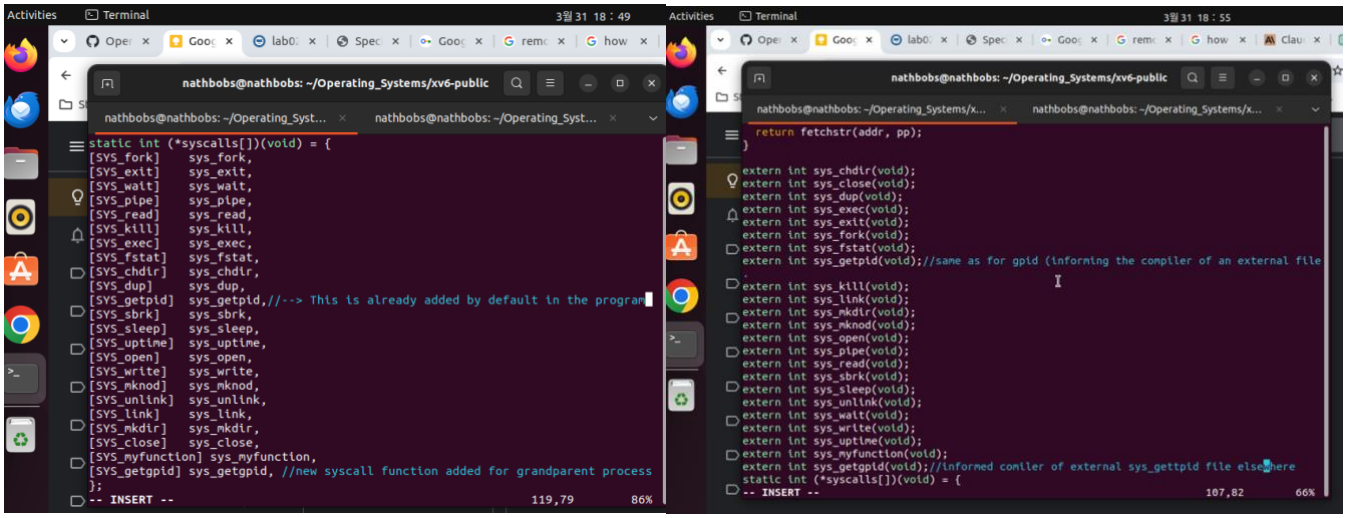


The following steps were further taken;

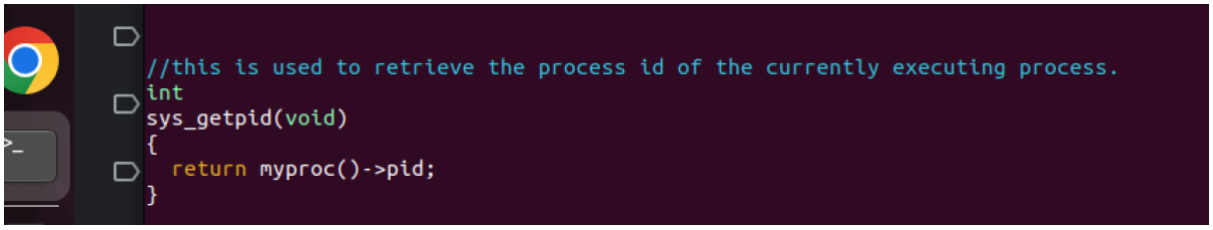
a) Modify the system call table (syscall.c) by adding and entry for getgpid and getpid system call into the array.
b) Implement the sys_getpid() function in the sysproc.c file which is supposed to retrieve the process ID of the current process and return it. Implement the sys_getgpid() function in the sysproc.c file which is supposed to retrieve the Parent process ID of the Parent process ID (grandparent process ID) of the current process and return it.
c) Modify User level library in usys.S by adding an entry for getpid() and getgpid() system call which allows the user program to be able to execute getpid and getgpid() whenever there's a system call.

d) Declare the getpid() and getgpid() in the user.h file and implement the user-level wrapper function for getpid() and getgpid() in the usys.S file.
e) Define SYS_getpid() and SYS_getgpid() in the syscall.h file to ensure that the system call is properly operating and linking.
f) Test the implementation by creating project01.c and writing the user program to test all of our implementations above.
g) Check the Makefile for modifications.

2a) From the file syscall.c in the xv6 codebase, I added the following; [SYS_getgpid] sys_getgpid and external int sys_getgpid(void); , to the already long list of system calls present in this file. This is used to define the system call handler for the getgpid system call. “external int sys_getgpid(void) on the other hand is used to inform the compiler that I have defined the sys_getgpid() function elsewhere and I will like to use it in the current file. This process is also repeated for the process ID in the form of [SYS_getpid] sys_getpid and external int sys_getpid(void);.



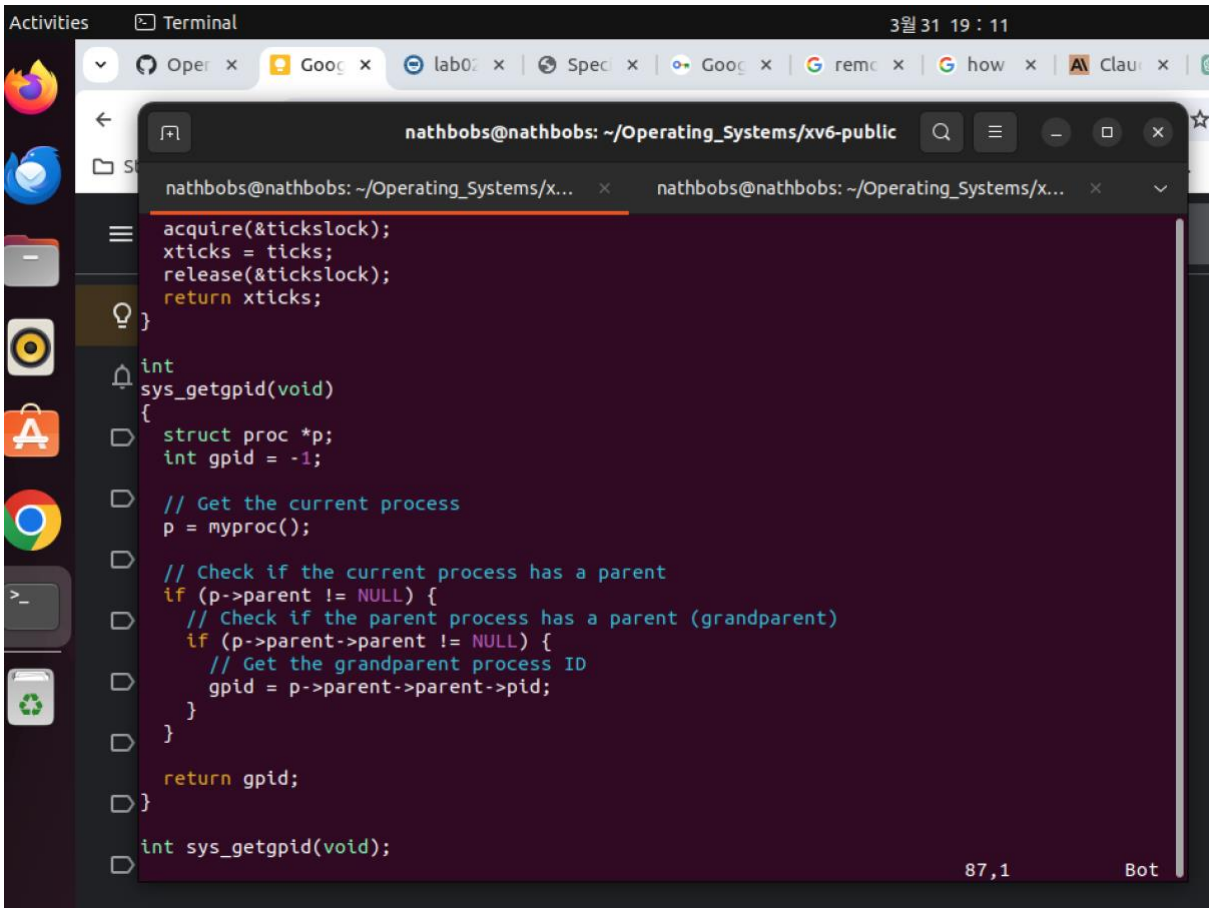
2b) In the sysproc.c file, I implemented various commands that is supposed to retrieve the current process id and grandparent of the current process id and return it. For getting the current process ID, return myproc()->pid; is used. This accesses the pid field of the proc structure that is returned by myproc(). Therefore, the value myproc()-> is returned by the sys_getpid() function.



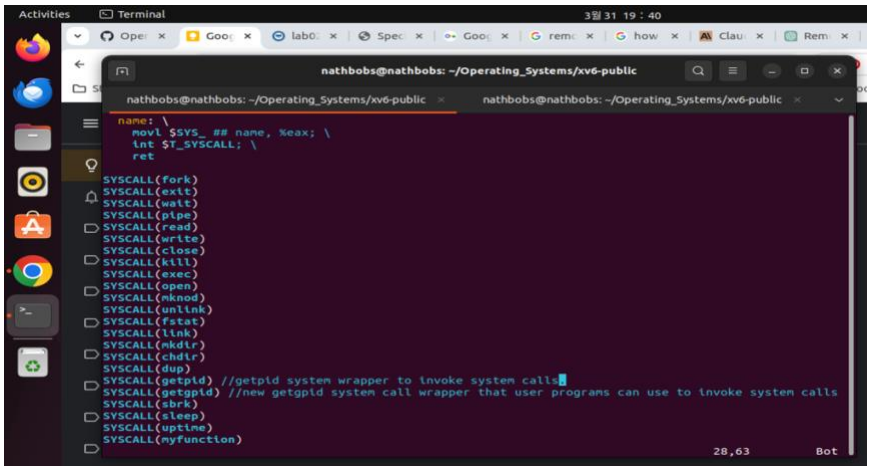
```
//this is used to retrieve the process id of the currently executing process.
int
sys_getpid(void)
{
  return myproc()->pid;
}
```

On the other hand, the sys_getgpid(void) is a little more complex than the getpid() process. First, a pointer to the current process (p) is obtained using myproc(). Then if the current

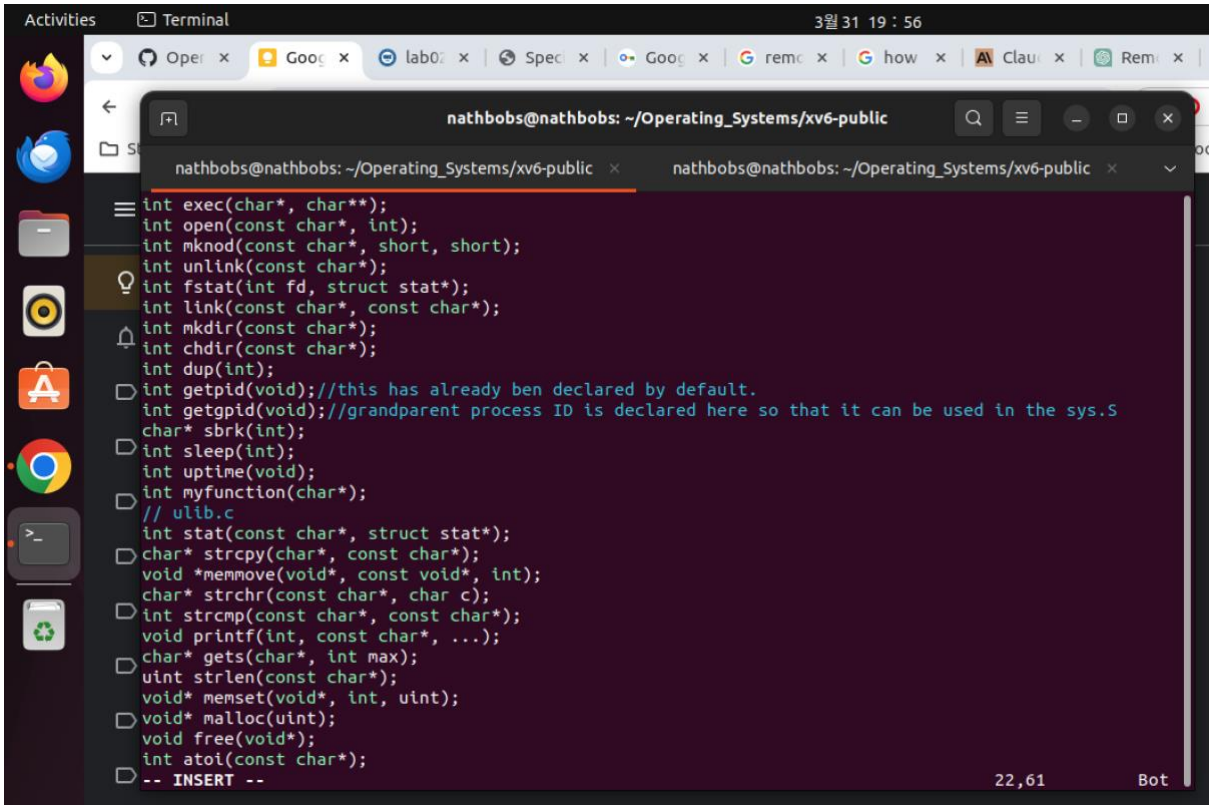
process has a parent, it does not return NULL (p->parent !=NULL). However, if it does not have a parent, it returns -1 which indicates that there is an error. In the next step, if the current process has a parent, then the parent process of that parent process is checked (grandparent process) (p->parent->parent !=NULL) This returns -1 if the parent does not have a parent. Next, if the grandparent process finally exists, the grandparent process id (p->parent->parent->pid) is stored in the gpid variable, and then the function returns the gpid value.



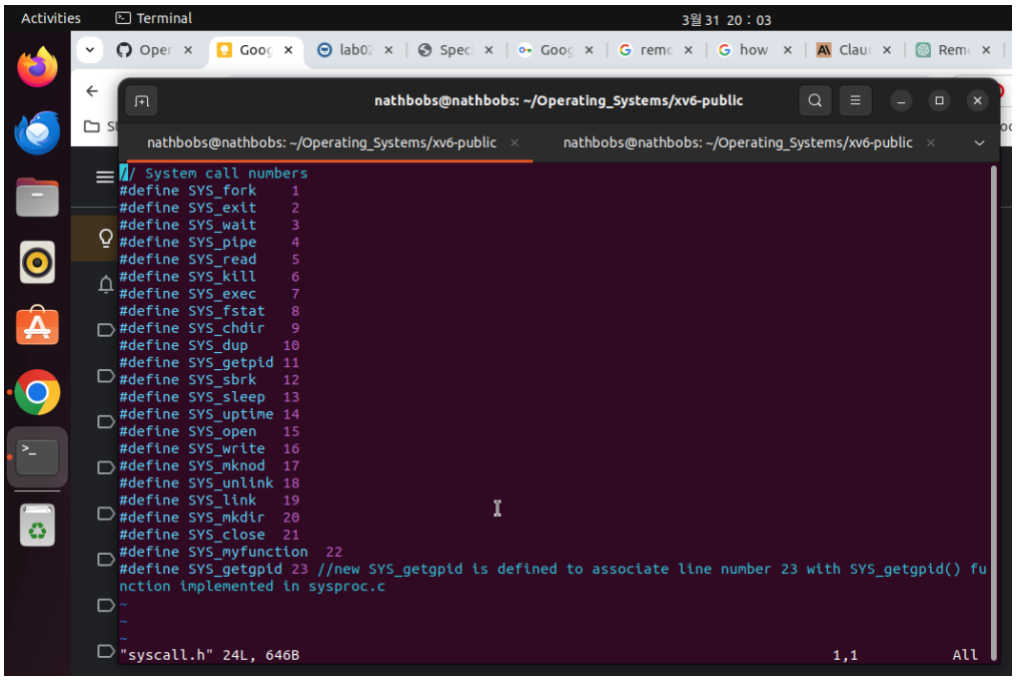
2c) For this step, an entry for the getgpid system call is added to the usys.S file. There was no need to add for getpid since it's already added there by default. Since this file contains system cal wrappers that user programs can use to invoke system calls, it can also be used to invoke system calls for the current process ID and the grandparent process ID.



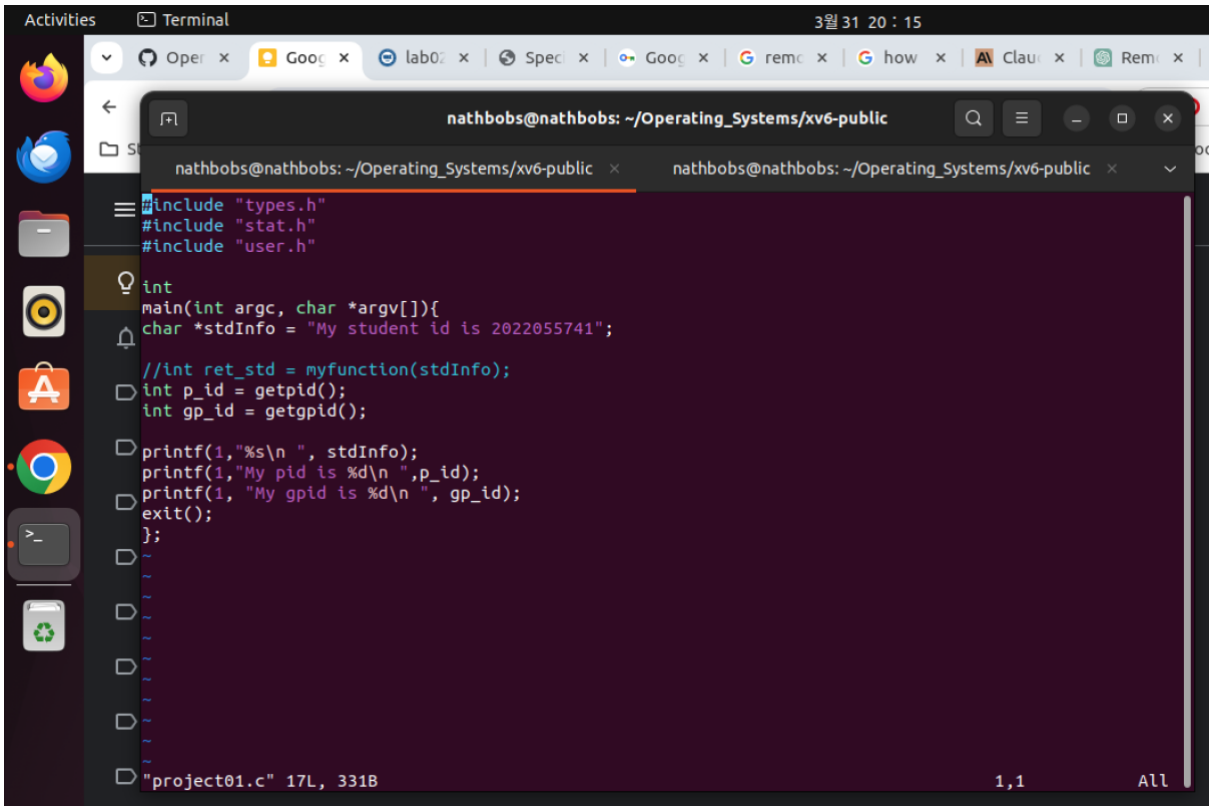
2d) getpid() and getgpid() is declared in the user.h file and implement the user-level wrapper function for getpid() and getgpid() in the usys.S file as in step 2c.



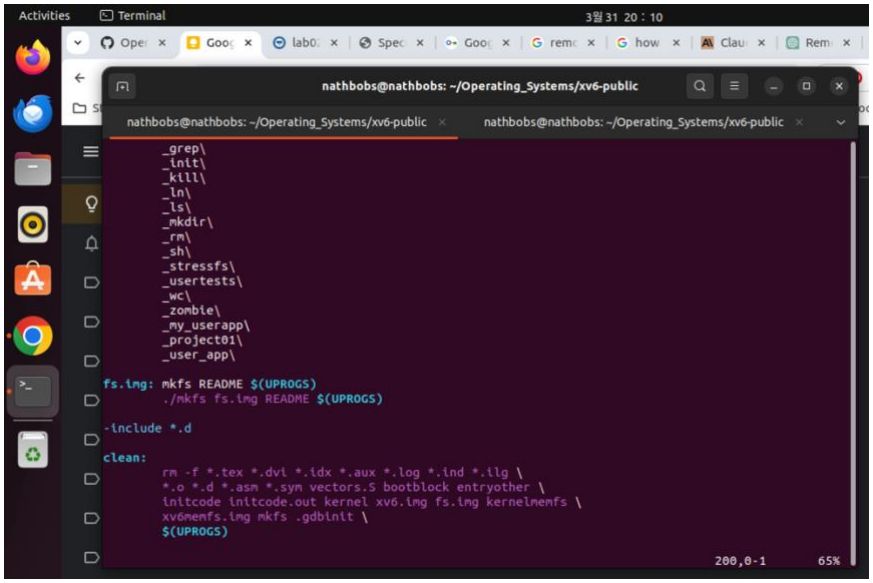
2e) SYS_getpid() and SYS_getgpid() is associated with the line number 23 and defined in the syscall.h file to ensure that the system call is properly operating and linking. Failure to do this will result to a non linkage between sysproc.c file and the user program hence resulting to errors.



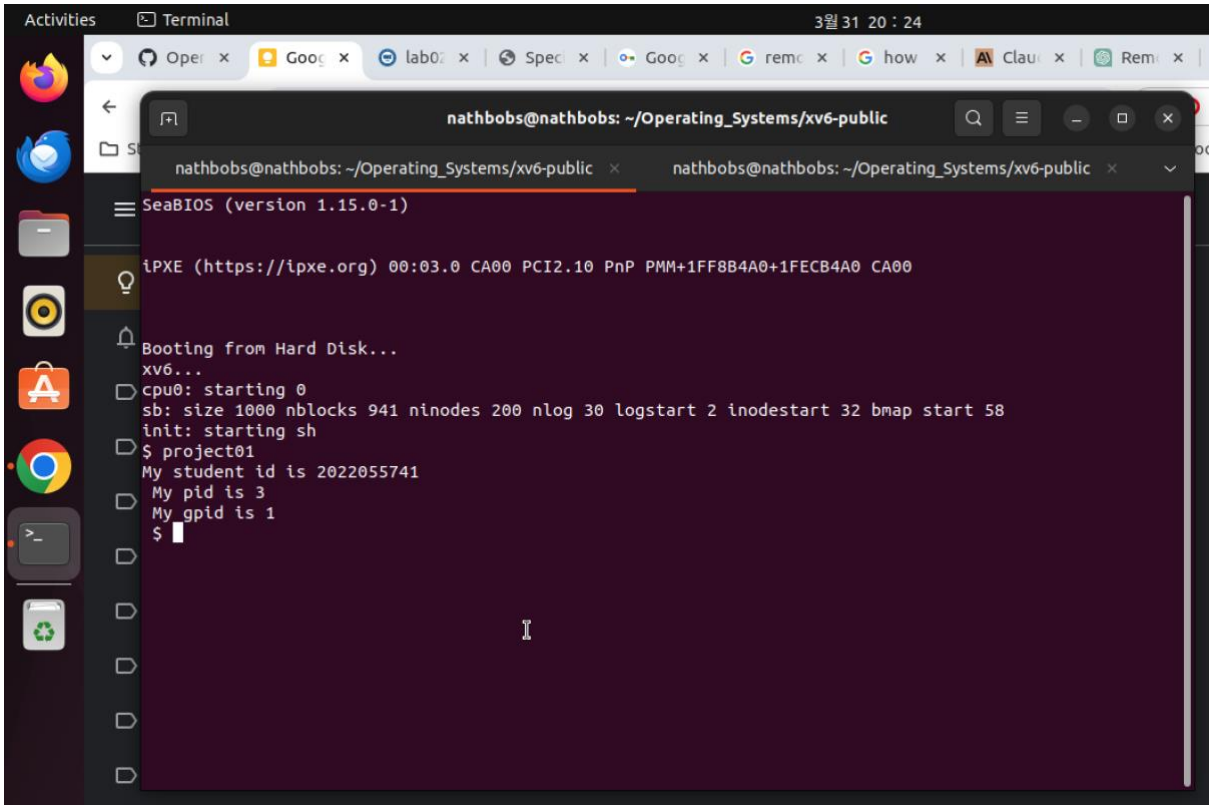
2f) After all the steps above, I created my user program named project01.c and wrote a program to display my student number, process ID and grandparent process ID. In the main function, stdInfo is a character pointer that holds my student ID. This is followed by an integer p_id which stores the process ID obtained by calling getpid() and then gp_id integer that stores the grandparent process ID obtained by calling getgpid(). The following string statements prints my student info, process ID and grandparent process ID after which the program exits().



2g) Modify the Makefile by adding _project01\ to enable the project to run.



3. Result



4. Troubleshooting

During the project, I encountered several problems but one major problem of linkage bothered me for 2 days before I was finally able to discover my mistake and correct it. The error I encountered was due to an undefined reference to the _cat function in the usys.S file “/home/nathbobs/Operating_Systems/xv6-public/usys.S 29: undefined reference to “‘make: *** [Makefile:150: _cat] Error 1” . This error happened because the linker in the program could not find the implementation of the _cat function that is being used in the usys.S file. I updated the UPROGS section of the Makefile, added what was missing in the usys.S file and the program was able to run successfully.