

ELE3021 [Operating System]

Project01 Wiki

Abegunde Adebawale 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.

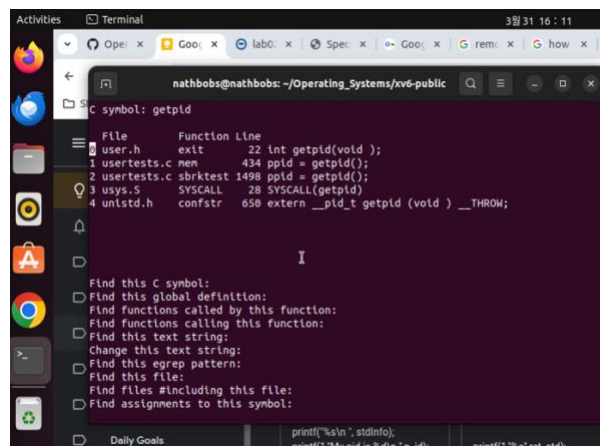
1. Introduction:

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 an entry for getgid 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_getgid() 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 getgid() system call which allows the user program to be able to execute getpid and getgid() whenever there's a system call.

- d) Declare the getpid() and getgid() in the user.h file and implement the user-level wrapper function for getpid() and getgid() in the usys.S file.
- e) Define SYS_getpid() and SYS_getgid() 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_getgid] sys_getgid and external int sys_getgid(void); , to the already long list of system calls present in this file. This is used to define the system call handler for the getgid system call. "external int sys_getgid(void) on the other hand is used to inform the compiler that I have defined the sys_getgid() 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);.

```

static int (*syscalls[])(void) = {
  [SYS_fork] sys_fork,
  [SYS_exit] sys_exit,
  [SYS_wait] sys_wait,
  [SYS_pipe] sys_pipe,
  [SYS_read] sys_read,
  [SYS_kill] sys_kill,
  [SYS_exec] sys_exec,
  [SYS_fstat] sys_fstat,
  [SYS_chdir] sys_chdir,
  [SYS_dup] sys_dup,
  [SYS_getpid] sys_getpid, //--> This is already added by default in the program
  [SYS_sbrk] sys_sbrk,
  [SYS_sleep] sys_sleep,
  [SYS_uptime] sys_uptime,
  [SYS_open] sys_open,
  [SYS_write] sys_write,
  [SYS_mknod] sys_mknod,
  [SYS_unlink] sys_unlink,
  [SYS_link] sys_link,
  [SYS_mkdir] sys_mkdir,
  [SYS_close] sys_close,
  [SYS_myfunction] sys_myfunction,
  [SYS_getgid] sys_getgid, //new syscall function added for grandparent process
};
-- INSERT --

```

```

extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void); //same as for gid (informing the compiler of an external file)
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_myfunction(void);
extern int sys_getgid(void); //Informing compiler of external sys_getgid file elsewhere
static int (*syscalls[])(void) = {
-- INSERT --

```

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.

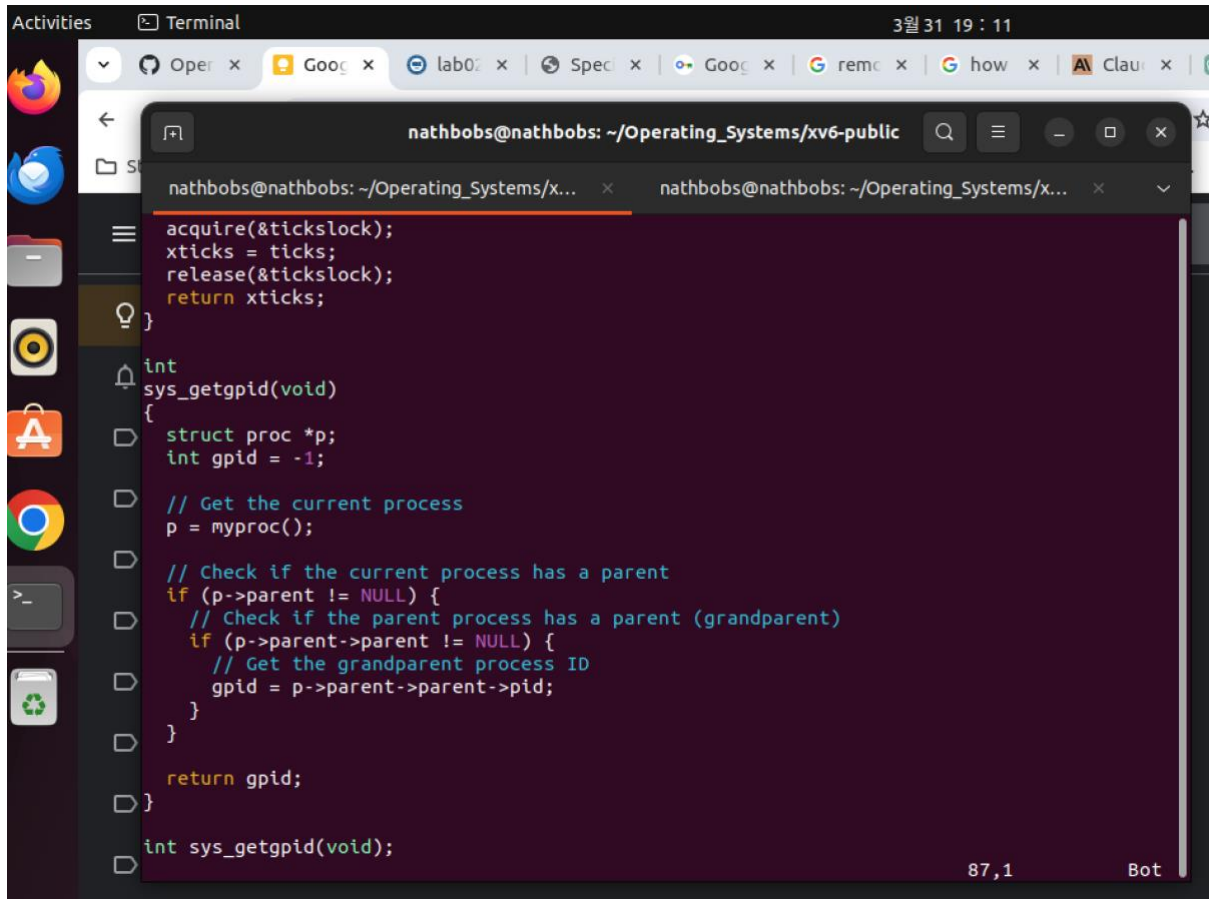
```

//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_getgid(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 ($p \rightarrow \text{parent} \neq \text{NULL}$). 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) ($p \rightarrow \text{parent} \rightarrow \text{parent} \neq \text{NULL}$). This returns -1 if the parent does not have a parent. Next, if the grandparent process finally exists, the grandparent process id ($p \rightarrow \text{parent} \rightarrow \text{parent} \rightarrow \text{pid}$) is stored in the gpid variable, and then the function returns the gpid value.



```

nathbobs@nathbobs: ~/Operating_Systems/xv6-public
nathbobs@nathbobs: ~/Operating_Systems/xv6-public
acquire(&tickslock);
xticks = ticks;
release(&tickslock);
return xticks;
}

int
sys_getpid(void)
{
    struct proc *p;
    int gpid = -1;

    // Get the current process
    p = myproc();

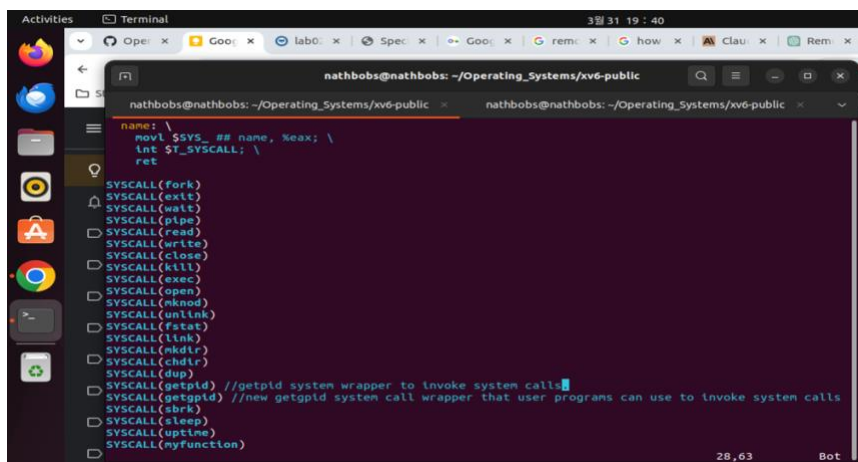
    // Check if the current process has a parent
    if (p->parent != NULL) {
        // Check if the parent process has a parent (grandparent)
        if (p->parent->parent != NULL) {
            // Get the grandparent process ID
            gpid = p->parent->parent->pid;
        }
    }

    return gpid;
}

int sys_getpid(void);
87,1 Bot

```

2c) For this step, an entry for the getpid 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 call 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.



```

nathbobs@nathbobs: ~/Operating_Systems/xv6-public
nathbobs@nathbobs: ~/Operating_Systems/xv6-public
name: \
movl $SYS_# name, %eax; \
int $T_SYSCALL; \
ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(rstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid) //getpid system wrapper to invoke system calls
SYSCALL(getgid) //new getpid system call wrapper that user programs can use to invoke system calls
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(myfunction)
28,63 Bot

```

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

```

nathbobs@nathbobs: ~/Operating_Systems/xv6-public
nathbobs@nathbobs: ~/Operating_Systems/xv6-public
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void); //this has already ben declared by default.
int getgid(void); //grandparent process ID is declared here so that it can be used in the sys.S
char* sbrk(int);
int sleep(int);
int uptime(void);
int myfunction(char*);
// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void printf(int, const char*, ...);
char* gets(char*, int max);
uint strlen(const char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
-- INSERT --
22,61 Bot

```

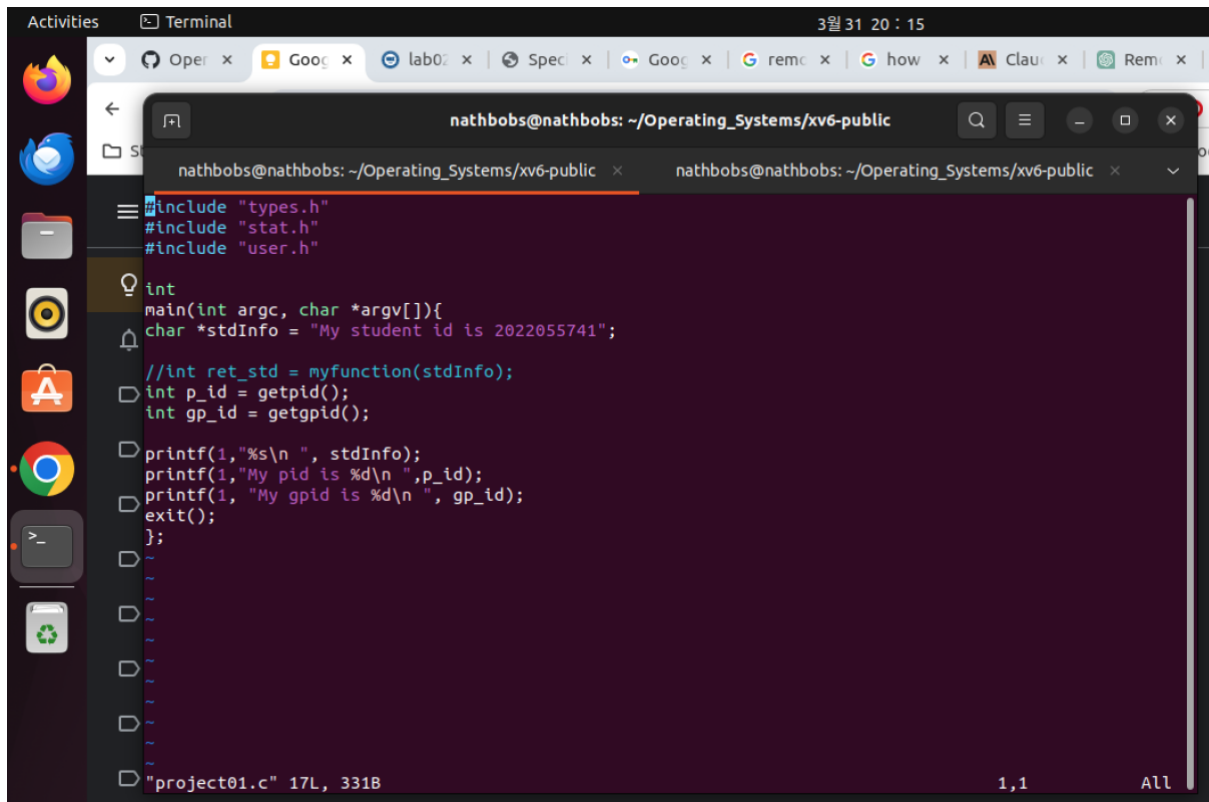
2e) SYS_getpid() and SYS_getgid() is associated with the line number 23 and defined in the syscall.h file to ensure that the system call is properly operating and linking. Failure to do this will result to a non linkage between sysproc.c file and the user program hence resulting to errors.

```

nathbobs@nathbobs: ~/Operating_Systems/xv6-public
nathbobs@nathbobs: ~/Operating_Systems/xv6-public
// System call numbers
#define SYS_fork 1
#define SYS_exit 2
#define SYS_wait 3
#define SYS_pipe 4
#define SYS_read 5
#define SYS_kill 6
#define SYS_exec 7
#define SYS_fstat 8
#define SYS_chdir 9
#define SYS_dup 10
#define SYS_getpid 11
#define SYS_sbrk 12
#define SYS_sleep 13
#define SYS_uptime 14
#define SYS_open 15
#define SYS_write 16
#define SYS_mknod 17
#define SYS_unlink 18
#define SYS_link 19
#define SYS_mkdir 20
#define SYS_close 21
#define SYS_myfunction 22
#define SYS_getpid 23 //new SYS_getgid is defined to associate line number 23 with SYS_getgid() fu
nction implemented in sysproc.c
--
"syscall.h" 24L, 646B
1,1 All

```

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 getppid(). The following string statements prints my student info, process ID and grandparent process ID after which the program exits().



```

nathbobs@nathbobs: ~/Operating_Systems/xv6-public
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[]){
    char *stdInfo = "My student id is 2022055741";

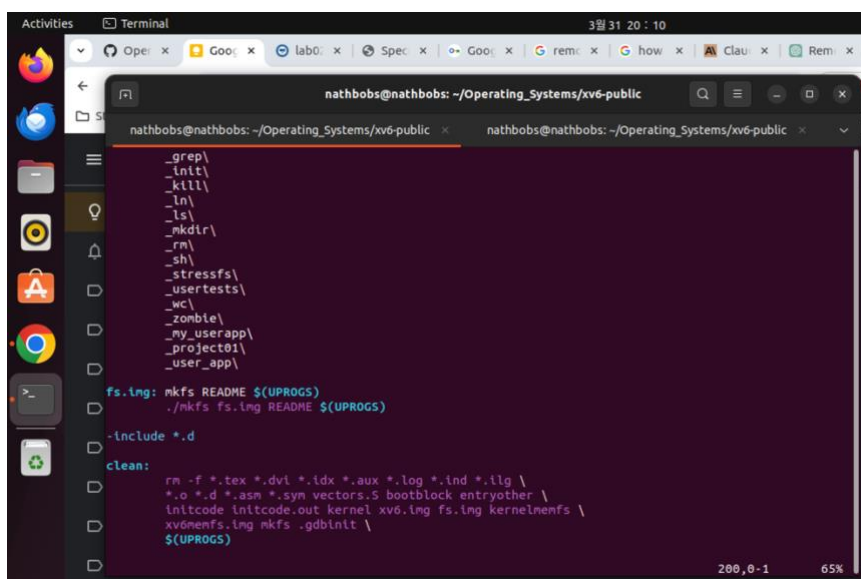
    //int ret_std = myfunction(stdInfo);
    int p_id = getpid();
    int gp_id = getppid();

    printf(1,"%s\n ", stdInfo);
    printf(1,"My pid is %d\n ",p_id);
    printf(1, "My gpid is %d\n ", gp_id);
    exit();
};

"project01.c" 17L, 331B
1,1 All

```

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



```

nathbobs@nathbobs: ~/Operating_Systems/xv6-public
_grep\
_init\
_kill\
_ln\
_ls\
_mkdir\
_rm\
_sh\
_stressfs\
_usertests\
_wc\
_zombie\
_my_userapp\
_project01\
_user_app\

fs.img: mkfs README $(UPROGS)
./mkfs fs.img README $(UPROGS)

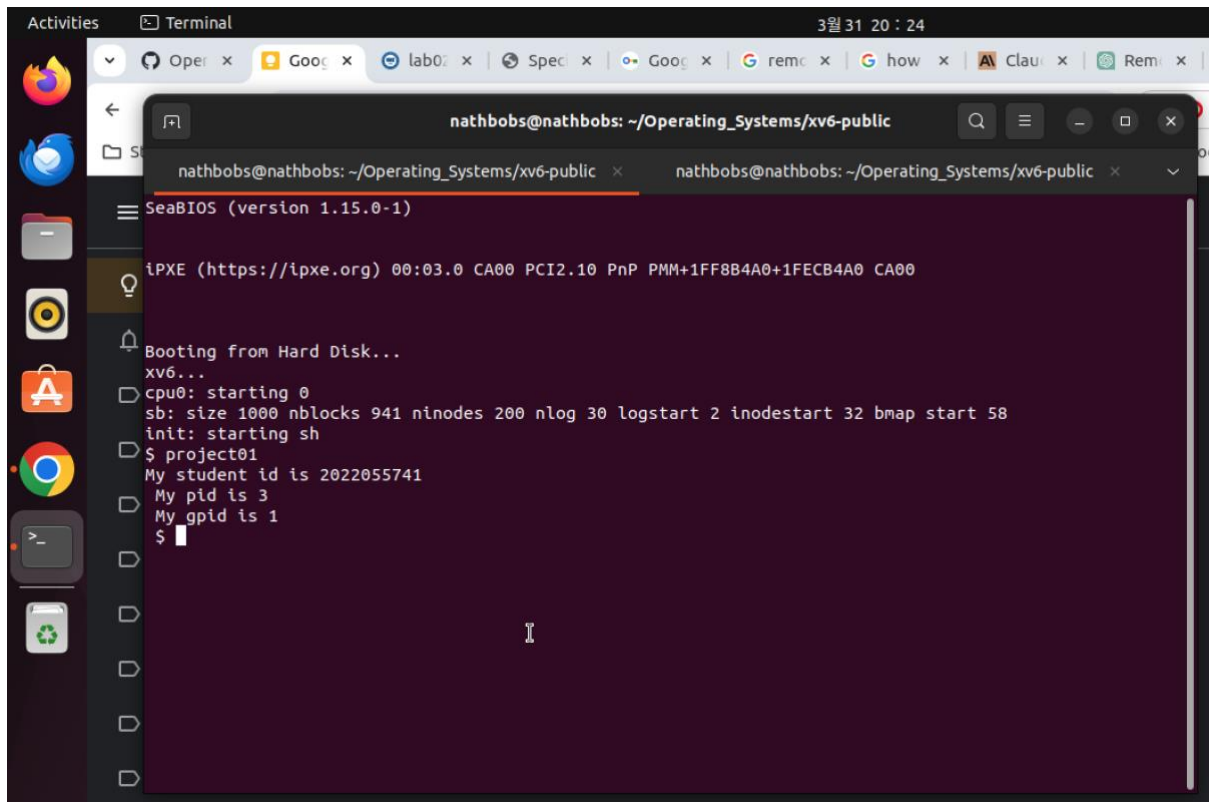
-include *.d

clean:
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
$(UPROGS)

200,0-1 65%

```


3. Result



```
nathbobs@nathbobs: ~/Operating_Systems/xv6-public
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00

Booting from Hard Disk...
xv6...
  cpu0: starting 0
  sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
  init: starting sh
  $ project01
    My student id is 2022055741
    My pid is 3
    My_gpid is 1
  $
```

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.