Operating System

Graded lab 2

Submitted by
Jerold Kingston GNANASEKARAN
5 Jan 2023

1. Building the Rootfs.

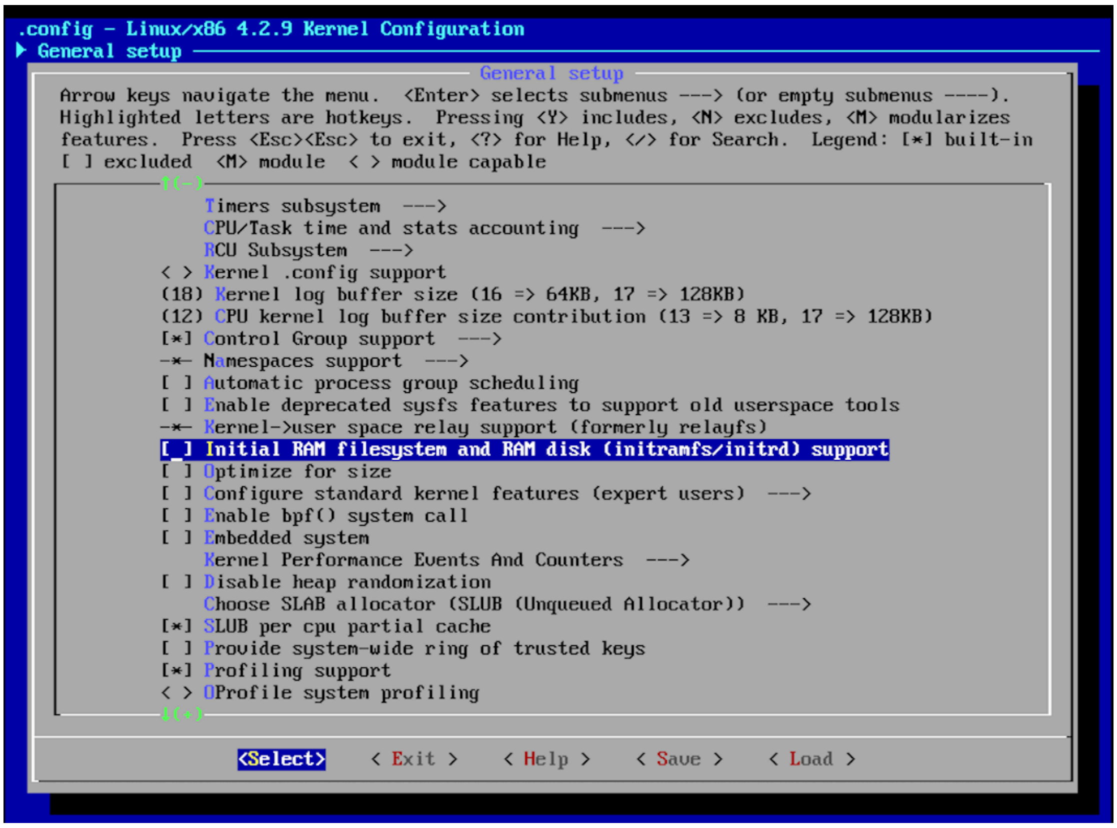
To store the history commands, I modified the .profile file in the path /usr/src/initrd/root/.profile.

export HISTFILE = “/usr/src/histfile”

To store the previous commands in the histfile.

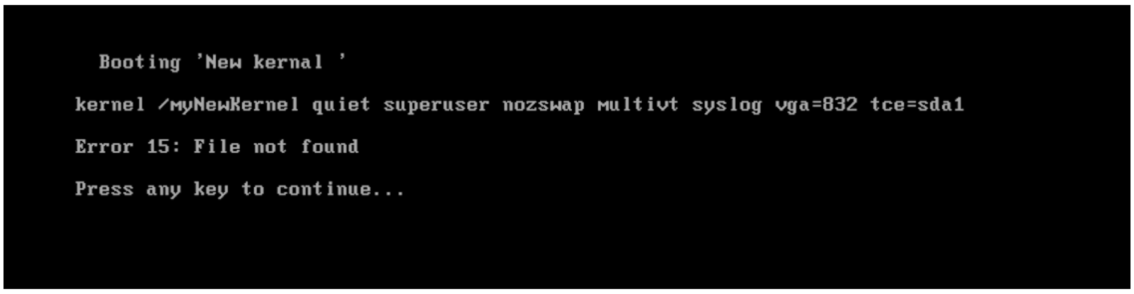
2. Compiling the linux commands.

To build a new kernel image that should not cause the kernel to boot correctly, I unselected the “initial RAM filesystem and RAM disk” in the configuration menu.



```
.config - Linux/x86 4.2.9 Kernel Configuration
▸ General setup ─────────────────────────────────────────────
                        General setup
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
 [ ] excluded  <M> module  < > module capable
          ↑(-)
          Timers subsystem  --->
          CPU/Task time and stats accounting  --->
          RCU Subsystem  --->
      < > Kernel .config support
      (18) Kernel log buffer size (16 => 64KB, 17 => 128KB)
      (12) CPU kernel log buffer size contribution (13 => 8 KB, 17 => 128KB)
      [*] Control Group support  --->
      -*- Namespaces support  --->
      [ ] Automatic process group scheduling
      [ ] Enable deprecated sysfs features to support old userspace tools
      -*- Kernel->user space relay support (formerly relayfs)
      [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
      [ ] Optimize for size
      [ ] Configure standard kernel features (expert users)  --->
      [ ] Enable bpf() system call
      [ ] Embedded system
          Kernel Performance Events And Counters  --->
      [ ] Disable heap randomization
          Choose SLAB allocator (SLUB (Unqueued Allocator))  --->
      [*] SLUB per cpu partial cache
      [ ] Provide system-wide ring of trusted keys
      [*] Profiling support
      < > OProfile system profiling
          ↓(+)

            <Select>    < Exit >    < Help >    < Save >    < Load >
```

After building and compiling. The newly creating kernel wasn’t booting correctly because of the missing of initial RAM file system and RAM disk.



```
  Booting 'New kernal '

kernel /myNewKernel quiet superuser nozswap multivt syslog vga=832 tce=sda1

Error 15: File not found

Press any key to continue...
```

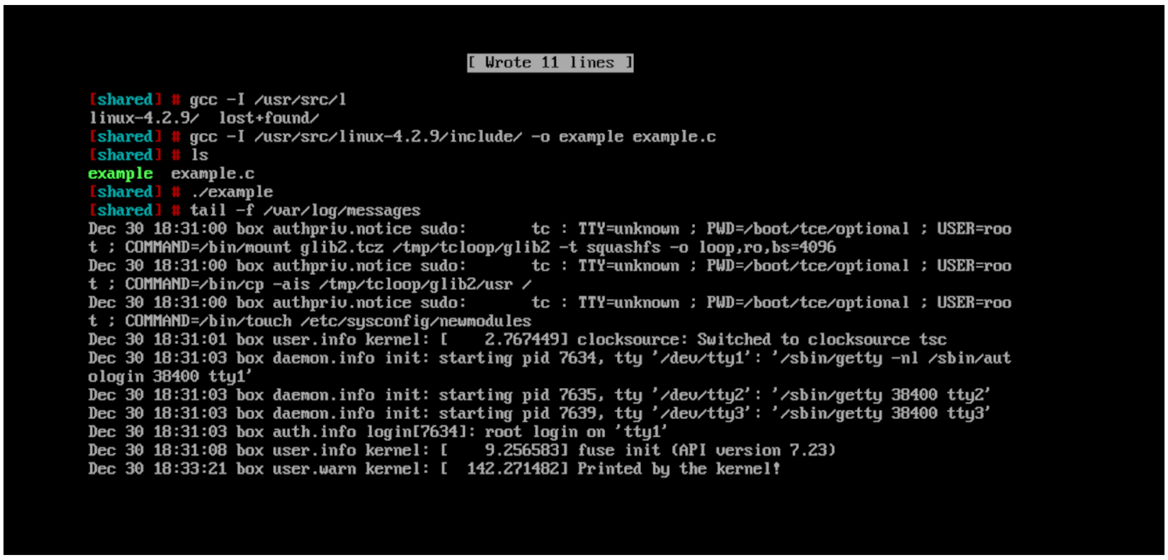
This is because the initial Ram disk (intird) and the Initial Ram file system (initrfs) both would be used in the linux kernel start up process to load the kernel from the file system. The bootable image is stored in that root file system and it is bounded in the boot procedure. Therefore, without that initial file system. The system wouldn't boot anything. The Grub is the boat loader that brings up or load the kernel from the file system location after the BIOS setup.

3.Adding a Basic system call to Linux "kernel print"

Generally, a system calls are the variety of services we are requesting to do it on behalf. For instance, we can request for the services like authentication, server connection and shutdown in our user application, we can do it in the same application or can be achieved by inheriting the header file.

I can recognise many sys calls in that..

Syscalls like chmod(), exit(), kill(), terminate() are normally used in our linux commands.



```
                              [ Wrote 11 lines ]

[shared] # gcc -I /usr/src/l
linux-4.2.9/  lost+found/
[shared] # gcc -I /usr/src/linux-4.2.9/include/ -o example example.c
[shared] # ls
example  example.c
[shared] # ./example
[shared] # tail -f /var/log/messages
Dec 30 18:31:00 box authpriv.notice sudo:       tc : TTY=unknown ; PWD=/boot/tce/optional ; USER=roo
t ; COMMAND=/bin/mount glib2.tcz /tmp/tcloop/glib2 -t squashfs -o loop,ro,bs=4096
Dec 30 18:31:00 box authpriv.notice sudo:       tc : TTY=unknown ; PWD=/boot/tce/optional ; USER=roo
t ; COMMAND=/bin/cp -ais /tmp/tcloop/glib2/usr /
Dec 30 18:31:00 box authpriv.notice sudo:       tc : TTY=unknown ; PWD=/boot/tce/optional ; USER=roo
t ; COMMAND=/bin/touch /etc/sysconfig/newmodules
Dec 30 18:31:01 box user.info kernel: [    2.767449] clocksource: Switched to clocksource tsc
Dec 30 18:31:03 box daemon.info init: starting pid 7634, tty '/dev/tty1': '/sbin/getty -nl /sbin/aut
ologin 38400 tty1'
Dec 30 18:31:03 box daemon.info init: starting pid 7635, tty '/dev/tty2': '/sbin/getty 38400 tty2'
Dec 30 18:31:03 box daemon.info init: starting pid 7639, tty '/dev/tty3': '/sbin/getty 38400 tty3'
Dec 30 18:31:03 box auth.info login[7634]: root login on 'tty1'
Dec 30 18:31:08 box user.info kernel: [    9.256583] fuse init (API version 7.23)
Dec 30 18:33:21 box user.warn kernel: [  142.271482] Printed by the kernel!
```

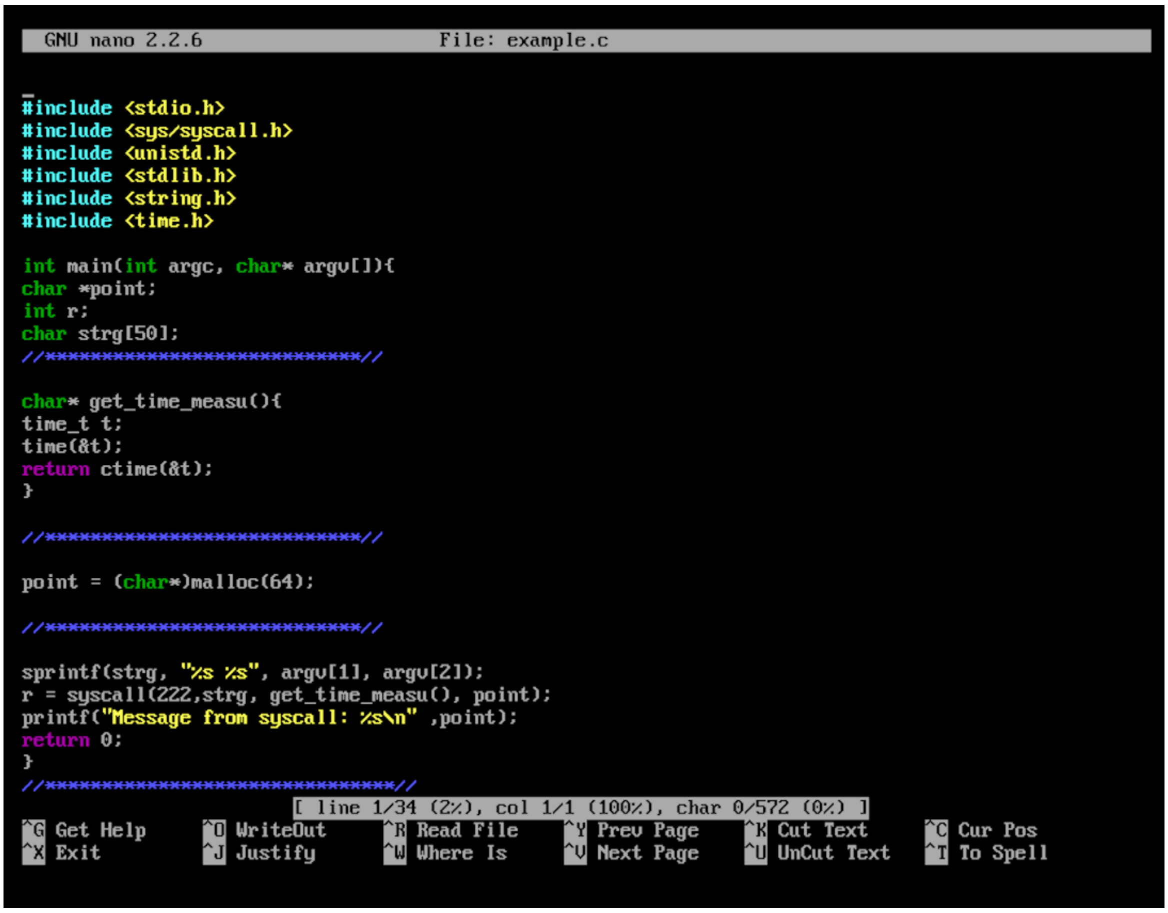
After calling the system_call's identifier number, in my case it was 359. The log message shows the kernel_print function's output.

4. Enhancing your system call.

For enhancing the system call, I used the function name called "getString" with that I passed three arguments "(String data, time data, pointer)" from the user space to the kernel space.

- String data to pass the "Addlog and Getlog" string with additional string.
- Time data to pass the current measurement from the user space to kernel space.
- I send the empty pointer as a third argument to the kernel space and returns a string as a current time to the user space.

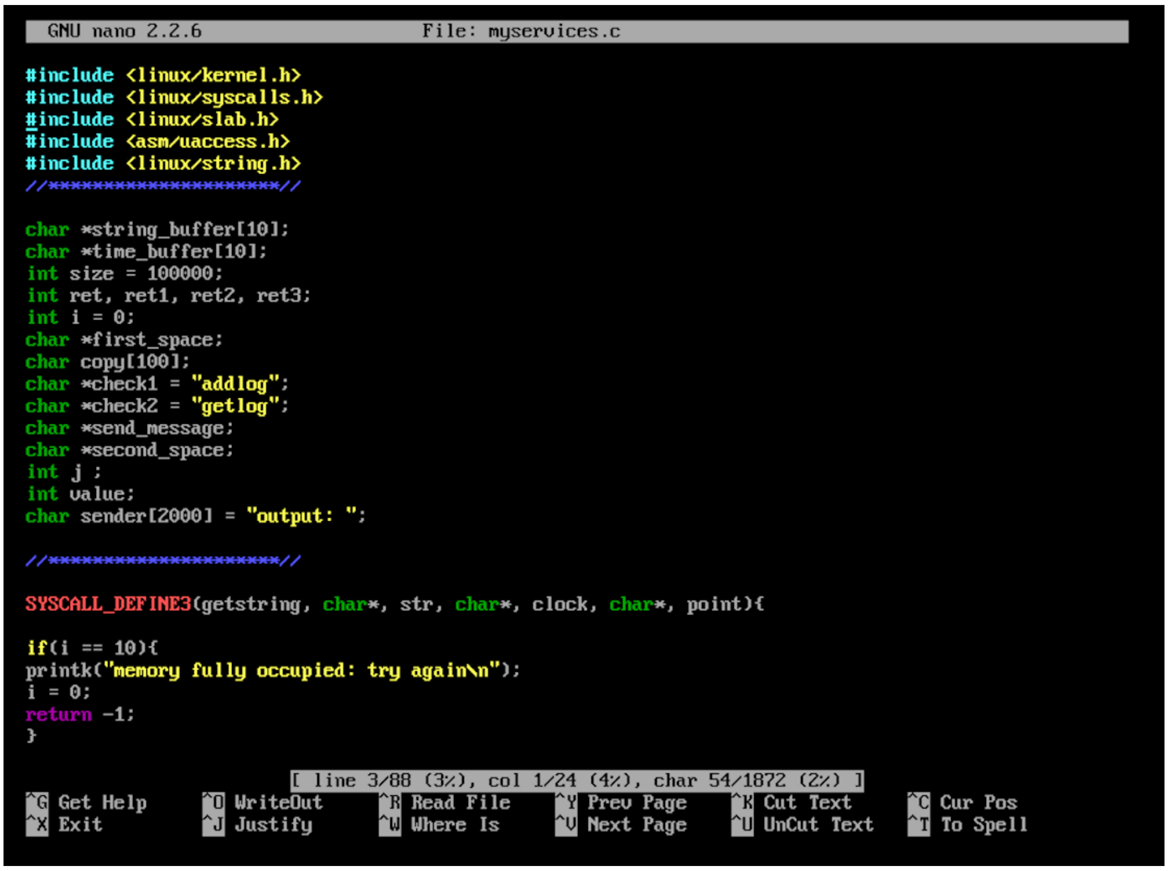
Logic Implementation in the User space:



```
  GNU nano 2.2.6                    File: example.c

#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int main(int argc, char* argv[]){
char *point;
int r;
char strg[50];
//******************************//

char* get_time_measu(){
time_t t;
time(&t);
return ctime(&t);
}

//******************************//

point = (char*)malloc(64);

//******************************//

sprintf(strg, "%s %s", argv[1], argv[2]);
r = syscall(222,strg, get_time_measu(), point);
printf("Message from syscall: %s\n" ,point);
return 0;
}
//*********************************//
                        [ line 1/34 (2%), col 1/1 (100%), char 0/572 (0%) ]
^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

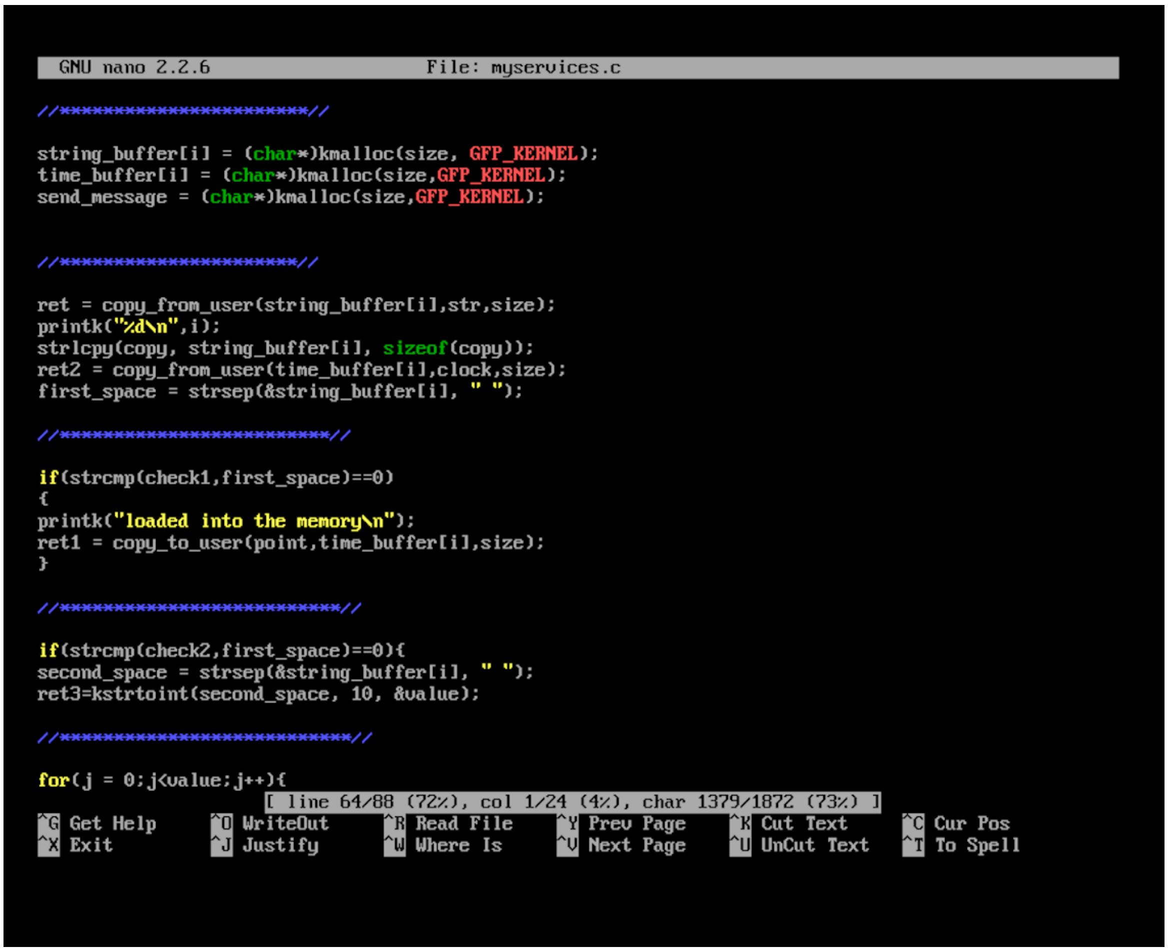
Firstly, I passed the "Addlog or Getlog" string as an argument from outside the user space with an executable file. Inside the code, I first bind the two strings from the argv[] using the Sprintf and I sent it as a first argument in the sys call function, then, I am calculated the time stamp and sent it to the kernel as a second argument. And finally as empty pointer to carry a value from the kernel space. And, Finally print function to print the value in the user space.

<u>Logic implementation in the Kernel Space</u>



```
  GNU nano 2.2.6                 File: myservices.c

#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
#include <linux/string.h>
//*********************//

char *string_buffer[10];
char *time_buffer[10];
int size = 100000;
int ret, ret1, ret2, ret3;
int i = 0;
char *first_space;
char copy[100];
char *check1 = "addlog";
char *check2 = "getlog";
char *send_message;
char *second_space;
int j ;
int value;
char sender[2000] = "output: ";

//*********************//

SYSCALL_DEFINE3(getstring, char*, str, char*, clock, char*, point){

if(i == 10){
printk("memory fully occupied: try again\n");
i = 0;
return -1;
}
                          [ line 3/88 (3%), col 1/24 (4%), char 54/1872 (2%) ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

In the above code, after declaring all the datatypes, I defined the sys call function with three argument. Below the code, I used the if condition for a reason to ensure the storage space within the limit of 10. Once, if the "i" value reached the value 10, I again reusing the memory by reinitialising again it to the memory from zero. I thought it would be the best way to save the kernel memory.



```
  GNU nano 2.2.6                    File: myservices.c

//***********************//

string_buffer[i] = (char*)kmalloc(size, GFP_KERNEL);
time_buffer[i] = (char*)kmalloc(size,GFP_KERNEL);
send_message = (char*)kmalloc(size,GFP_KERNEL);


//**********************//

ret = copy_from_user(string_buffer[i],str,size);
printk("%d\n",i);
strlcpy(copy, string_buffer[i], sizeof(copy));
ret2 = copy_from_user(time_buffer[i],clock,size);
first_space = strsep(&string_buffer[i], " ");

//**************************//

if(strcmp(check1,first_space)==0)
{
printk("loaded into the memory\n");
ret1 = copy_to_user(point,time_buffer[i],size);
}

//***************************//

if(strcmp(check2,first_space)==0){
second_space = strsep(&string_buffer[i], " ");
ret3=kstrtoint(second_space, 10, &value);

//****************************//

for(j = 0;j<value;j++){
                      [ line 64/88 (72%), col 1/24 (4%), char 1379/1872 (73%) ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

In the above code, I totally created three memory to store the value coming from the user space using kmalloc.

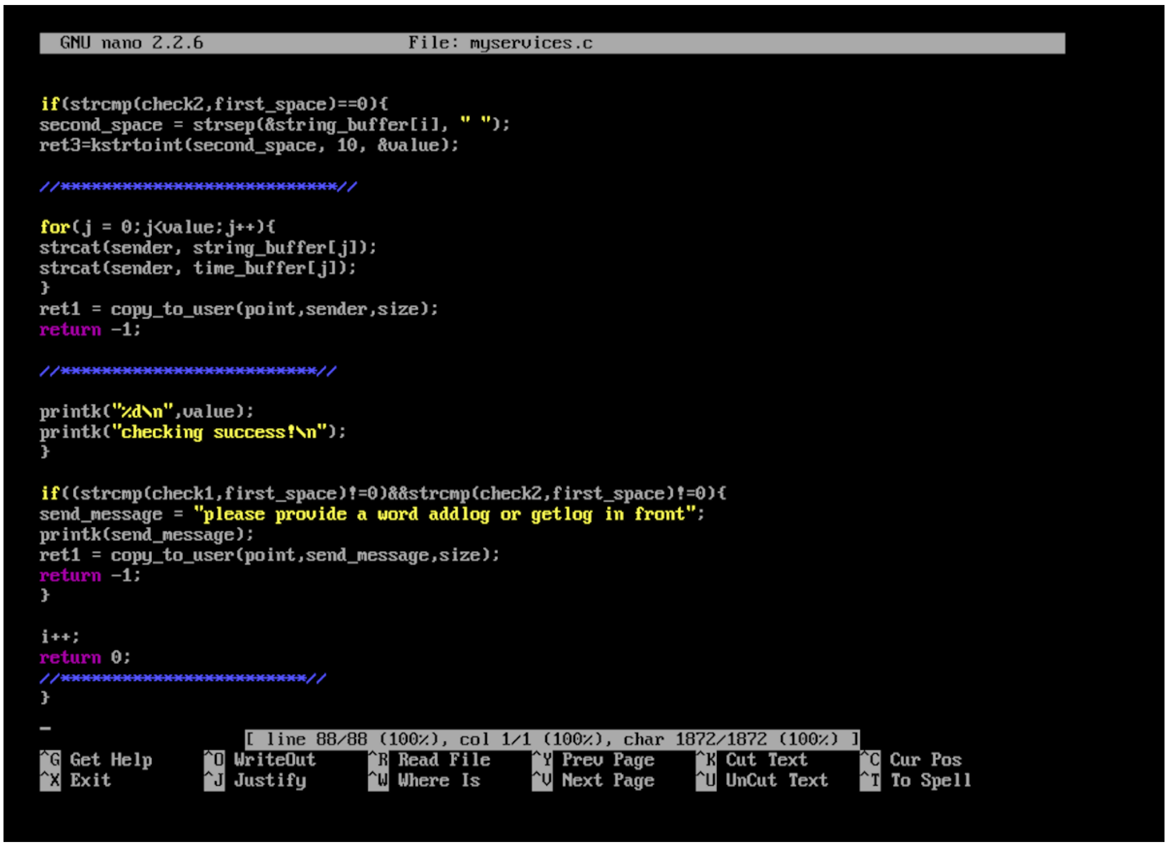
I'm using the copy_from_user function to copy the string value to the two pointer assigned in the kernel space, One pointer to store the time stamp and other to store the string. Once, its done I copy the string for the later use, because the function next I am going to use will modify the entire string in the end, Therefore, nothing will store in the memory.

After creating copy of string I extracted the first word using the strsep function to check whether its a addlog or a getlog word I received from the user space,

If the first space matches with the addlog then I send the remaining word to the user space by using the copy_to_user function. And the way I did it for the getlog is the way different then the addlog.

Once the getlog matches with the getlog check pointer. I again extracted the second value to get the n number of values to send back to the user space.

The second values its in the format of string so I converted it into integer using a strtoint function.

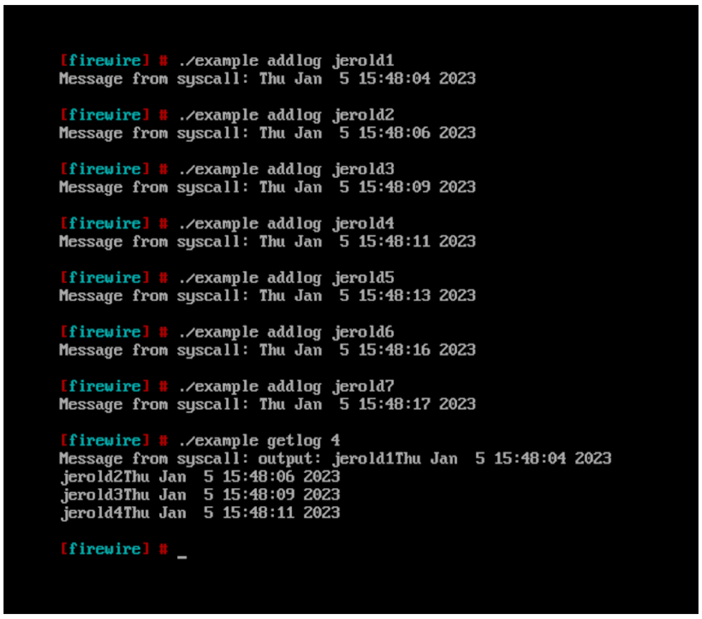


```
  GNU nano 2.2.6                 File: myservices.c


if(strcmp(check2,first_space)==0){
second_space = strsep(&string_buffer[i], " ");
ret3=kstrtoint(second_space, 10, &value);

//****************************//

for(j = 0;j<value;j++){
strcat(sender, string_buffer[j]);
strcat(sender, time_buffer[j]);
}
ret1 = copy_to_user(point,sender,size);
return -1;

//**************************//

printk("%d\n",value);
printk("checking success!\n");
}

if((strcmp(check1,first_space)!=0)&&strcmp(check2,first_space)!=0){
send_message = "please provide a word addlog or getlog in front";
printk(send_message);
ret1 = copy_to_user(point,send_message,size);
return -1;
}

i++;
return 0;
//************************//
}
_
                    [ line 88/88 (100%), col 1/1 (100%), char 1872/1872 (100%) ]
^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

I used the for loop to send the n number of values stored in the memory. In kernel we cannot append the values in the way that we do it user space. So I again used the “strcat” function to append the two string from the memory and I send it to the user space using the copy_to_user function.

If the first word of the string not belongs to any of the recognisable word the “please provide a valid” message will send to the user space.

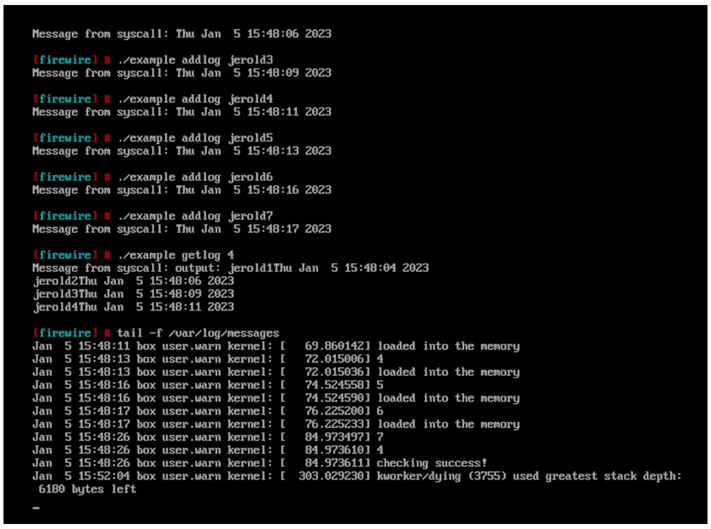
Output image:



```
[firewire] # ./example addlog jerold1
Message from syscall: Thu Jan  5 15:48:04 2023

[firewire] # ./example addlog jerold2
Message from syscall: Thu Jan  5 15:48:06 2023

[firewire] # ./example addlog jerold3
Message from syscall: Thu Jan  5 15:48:09 2023

[firewire] # ./example addlog jerold4
Message from syscall: Thu Jan  5 15:48:11 2023

[firewire] # ./example addlog jerold5
Message from syscall: Thu Jan  5 15:48:13 2023

[firewire] # ./example addlog jerold6
Message from syscall: Thu Jan  5 15:48:16 2023

[firewire] # ./example addlog jerold7
Message from syscall: Thu Jan  5 15:48:17 2023

[firewire] # ./example getlog 4
Message from syscall: output: jerold1Thu Jan  5 15:48:04 2023
jerold2Thu Jan  5 15:48:06 2023
jerold3Thu Jan  5 15:48:09 2023
jerold4Thu Jan  5 15:48:11 2023

[firewire] # _
```

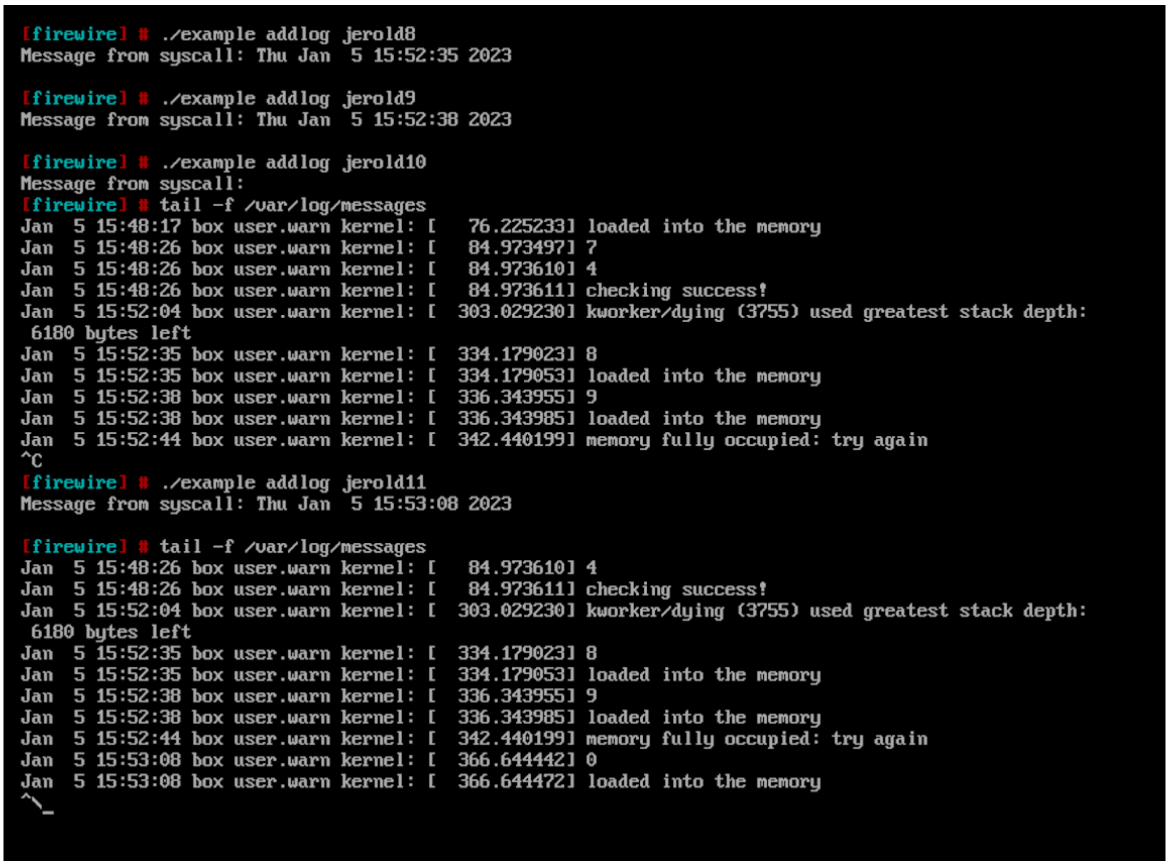
I passed the addlog string as an argument with the executable and the kernel returns the time stamp and I print it in the user space.

For the getlog, it returns the n value stored in the memory, in this case. The kernel returned 4 values.



```
Message from syscall: Thu Jan  5 15:48:06 2023

[firewire] # ./example addlog jerold3
Message from syscall: Thu Jan  5 15:48:09 2023

[firewire] # ./example addlog jerold4
Message from syscall: Thu Jan  5 15:48:11 2023

[firewire] # ./example addlog jerold5
Message from syscall: Thu Jan  5 15:48:13 2023

[firewire] # ./example addlog jerold6
Message from syscall: Thu Jan  5 15:48:16 2023

[firewire] # ./example addlog jerold7
Message from syscall: Thu Jan  5 15:48:17 2023

[firewire] # ./example getlog 4
Message from syscall: output: jerold1Thu Jan  5 15:48:04 2023
jerold2Thu Jan  5 15:48:06 2023
jerold3Thu Jan  5 15:48:09 2023
jerold4Thu Jan  5 15:48:11 2023

[firewire] # tail -f /var/log/messages
Jan  5 15:48:11 box user.warn kernel: [   69.860142] loaded into the memory
Jan  5 15:48:13 box user.warn kernel: [   72.015006] 4
Jan  5 15:48:13 box user.warn kernel: [   72.015036] loaded into the memory
Jan  5 15:48:16 box user.warn kernel: [   74.524558] 5
Jan  5 15:48:16 box user.warn kernel: [   74.524590] loaded into the memory
Jan  5 15:48:17 box user.warn kernel: [   76.225200] 6
Jan  5 15:48:17 box user.warn kernel: [   76.225233] loaded into the memory
Jan  5 15:48:26 box user.warn kernel: [   84.973497] 7
Jan  5 15:48:26 box user.warn kernel: [   84.973610] 4
Jan  5 15:48:26 box user.warn kernel: [   84.973611] checking success!
Jan  5 15:52:04 box user.warn kernel: [  303.029230] kworker/dying (3755) used greatest stack depth:
 6180 bytes left
_
```

here is the message in the kernel space about the value loaded in to the memory.



```
[firewire] # ./example addlog jerold8
Message from syscall: Thu Jan  5 15:52:35 2023

[firewire] # ./example addlog jerold9
Message from syscall: Thu Jan  5 15:52:38 2023

[firewire] # ./example addlog jerold10
Message from syscall:
[firewire] # tail -f /var/log/messages
Jan  5 15:48:17 box user.warn kernel: [   76.225233] loaded into the memory
Jan  5 15:48:26 box user.warn kernel: [   84.973497] 7
Jan  5 15:48:26 box user.warn kernel: [   84.973610] 4
Jan  5 15:48:26 box user.warn kernel: [   84.973611] checking success!
Jan  5 15:52:04 box user.warn kernel: [  303.029230] kworker/dying (3755) used greatest stack depth:
 6180 bytes left
Jan  5 15:52:35 box user.warn kernel: [  334.179023] 8
Jan  5 15:52:35 box user.warn kernel: [  334.179053] loaded into the memory
Jan  5 15:52:38 box user.warn kernel: [  336.343955] 9
Jan  5 15:52:38 box user.warn kernel: [  336.343985] loaded into the memory
Jan  5 15:52:44 box user.warn kernel: [  342.440199] memory fully occupied: try again
^C
[firewire] # ./example addlog jerold11
Message from syscall: Thu Jan  5 15:53:08 2023

[firewire] # tail -f /var/log/messages
Jan  5 15:48:26 box user.warn kernel: [   84.973610] 4
Jan  5 15:48:26 box user.warn kernel: [   84.973611] checking success!
Jan  5 15:52:04 box user.warn kernel: [  303.029230] kworker/dying (3755) used greatest stack depth:
 6180 bytes left
Jan  5 15:52:35 box user.warn kernel: [  334.179023] 8
Jan  5 15:52:35 box user.warn kernel: [  334.179053] loaded into the memory
Jan  5 15:52:38 box user.warn kernel: [  336.343955] 9
Jan  5 15:52:38 box user.warn kernel: [  336.343985] loaded into the memory
Jan  5 15:52:44 box user.warn kernel: [  342.440199] memory fully occupied: try again
Jan  5 15:53:08 box user.warn kernel: [  366.644442] 0
Jan  5 15:53:08 box user.warn kernel: [  366.644472] loaded into the memory
^\_
```

Once when the i value reaches a maximum limit, the memory fully occupied message pop up and again when we loaded the memory it starts with i = 0 to reuse the memory.