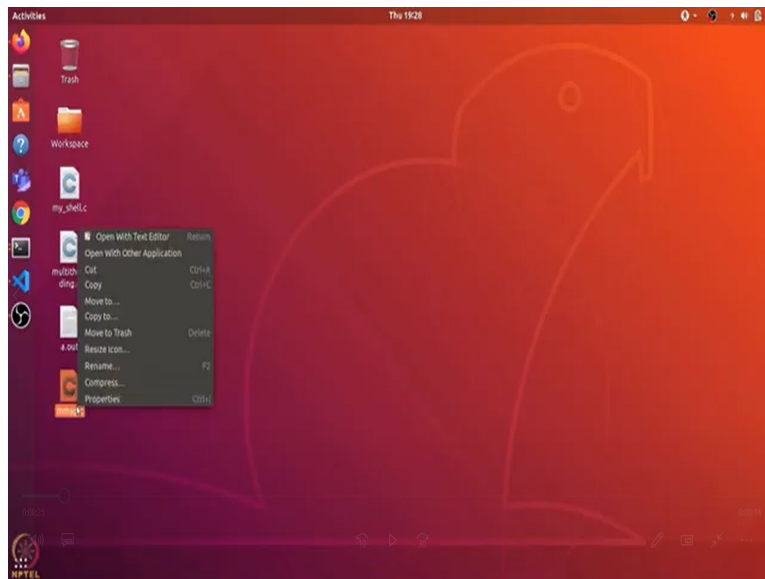


Design and Engineering of Computer Systems
Professor. Mythili Vutukuru
Computer Science and Engineering
Indian Institute of Technology, Bombay
Week 3, Tutorial 3
Memory allocation using mmap

Hi everyone, in this video we will learn how we can use the mmap function to allocate memory in a C program.

(Refer Slide Time: 00:25)

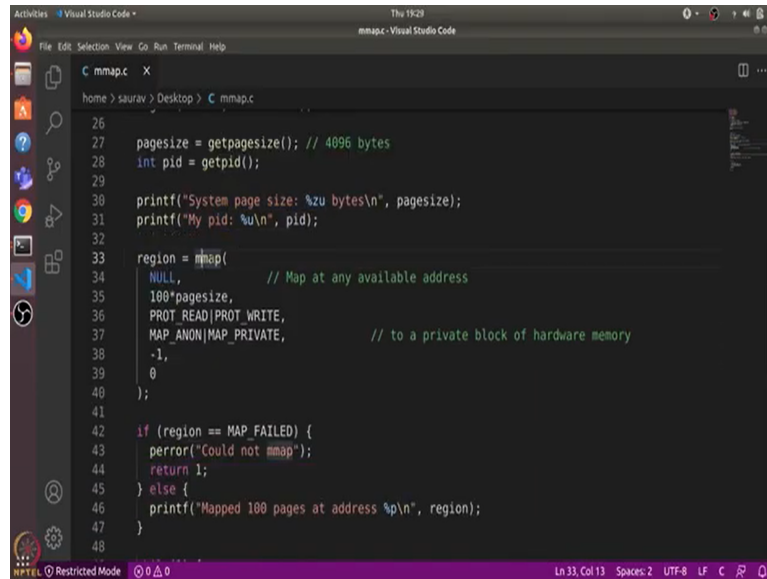
A screenshot of a Visual Studio Code editor window titled 'mmap.c - Visual Studio Code'. The editor displays the following C code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <sys/mman.h>
5 #include <unistd.h>
6 #include <signal.h>
7
8 char *region;
9 size_t pagesize;
10
11 void intHandler(int dummy) {
12
13     int unmap_result = munmap(region, 100*pagesize);
14     if (unmap_result != 0) {
15         perror("Could not munmap\n");
16         exit(1);
17     }
18     printf("Successfully unmapped\n");
19     exit(0);
20 }
21
22
23 int main(void) {
```

The status bar at the bottom indicates 'Ln 35, Col 27', 'Spaces: 2', 'UTF-8', 'LF', 'C', and 'R'.

```
Activities Visual Studio Code Thu 19:28
mmap.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
C mmap.c x
home > saurav > Desktop > C mmap.c
20
21 }
22
23 int main(void) {
24
25     signal(SIGINT, intHandler);
26
27     pagesize = getpagesize();
28     int pid = getpid();
29
30     printf("System page size: %zu bytes\n", pagesize);
31     printf("My pid: %u\n", pid);
32
33     region = mmap(
34         NULL, // Map at any available address
35         100*pagesize,
36         PROT_READ|PROT_WRITE,
37         MAP_ANON|MAP_PRIVATE, // to a private block of hardware memory
38         -1,
39         0
40     );
41
42     if (region == MAP_FAILED) {
43         perror("could not mmap");
44     }
45 }
```

```
Activities Visual Studio Code Thu 19:28
mmap.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
C mmap.c
home > saurav > Desktop > C mmap.c
18 printf("Successfully unmapped\n");
19 exit(0);
20
21 }
22
23 int main(void) {
24
25     signal(SIGINT, intHandler);
26
27     pagesize = getpagesize(); // 4096 bytes
28     int pid = getpid();
29
30     printf("System page size: %zu bytes\n", pagesize);
31     printf("My pid: %u\n", pid);
32
33     region = mmap(
34         NULL, // Map at any available address
35         100*pagesize,
36         PROT_READ|PROT_WRITE,
37         MAP_ANON|MAP_PRIVATE, // to a private block of hardware memory
38         -1,
39         0
40     );
41 }
```

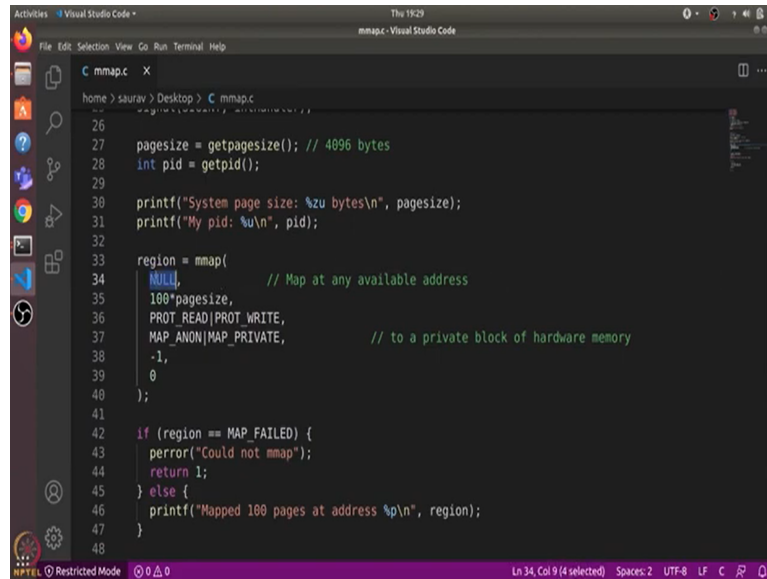


```
26
27 pagesize = getpagesize(); // 4096 bytes
28 int pid = getpid();
29
30 printf("System page size: %zu bytes\n", pagesize);
31 printf("My pid: %u\n", pid);
32
33 region = mmap(
34     NULL,           // Map at any available address
35     100*pagesize,
36     PROT_READ|PROT_WRITE,
37     MAP_ANON|MAP_PRIVATE, // to a private block of hardware memory
38     -1,
39     0
40 );
41
42 if (region == MAP_FAILED) {
43     perror("Could not mmap");
44     return 1;
45 } else {
46     printf("Mapped 100 pages at address %p\n", region);
47 }
48
```

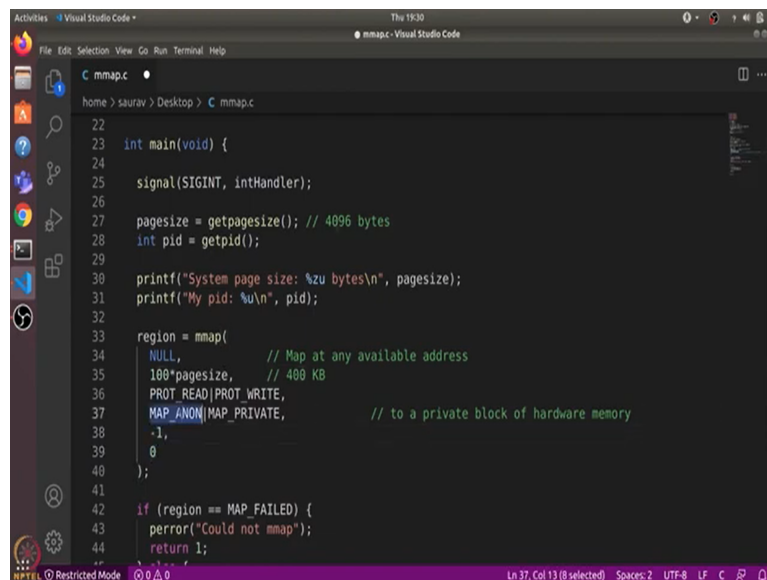
So, I have written this mmap dot c program here. Let us open this with visual code. So, this is the mmap dot c program. I will go over the code and then we will compile and run this. So, here is the main function I defined a signal handler I will come to this later.

So, first we get the page size of the system using the get page size function. So the page size is generally 4 kb which is 4096 bytes. Then we define the pid using the get pid function and this will return the pid of the process which is running this code. Then we use printf to print out the page size and the pid of this program and this is where we are using the mmap function to allocate memory.

(Refer Slide Time: 01:18)



```
26
27 pagesize = getpagesize(); // 4096 bytes
28 int pid = getpid();
29
30 printf("System page size: %zu bytes\n", pagesize);
31 printf("My pid: %u\n", pid);
32
33 region = mmap(
34     NULL,           // Map at any available address
35     100*pagesize,
36     PROT_READ|PROT_WRITE,
37     MAP_ANON|MAP_PRIVATE, // to a private block of hardware memory
38     -1,
39     0
40 );
41
42 if (region == MAP_FAILED) {
43     perror("Could not mmap");
44     return 1;
45 } else {
46     printf("Mapped 100 pages at address %p\n", region);
47 }
48
```



```
22
23 int main(void) {
24
25     signal(SIGINT, intHandler);
26
27     pagesize = getpagesize(); // 4096 bytes
28     int pid = getpid();
29
30     printf("System page size: %zu bytes\n", pagesize);
31     printf("My pid: %u\n", pid);
32
33     region = mmap(
34         NULL,           // Map at any available address
35         100*pagesize,   // 400 KB
36         PROT_READ|PROT_WRITE,
37         MAP_ANON|MAP_PRIVATE, // to a private block of hardware memory
38         -1,
39         0
40     );
41
42     if (region == MAP_FAILED) {
43         perror("Could not mmap");
44         return 1;
45     }
46 }
```

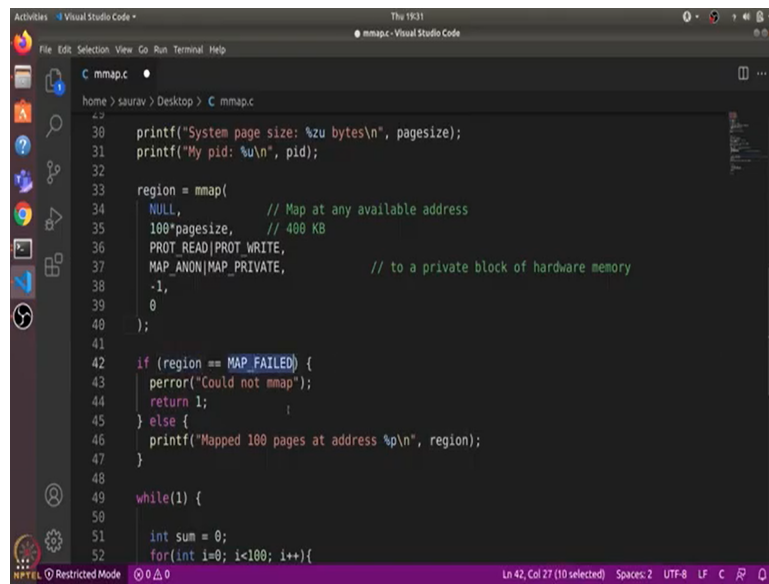
So, here we have defined this `char * region` which is a pointer to the memory and this `mmap` function takes first argument as the address at which we want to allocate memory and the system will try to assign us memory at that particular address. But we have given `null` here so it will just give us the memory at any available address. Then the second argument defines the size of the memory that you want to allocate. So here, we have given 100 times the page size which means it will be around 400 kilobytes of memory.

So, we are allocating about 400 kilobytes of memory. Then this defines what are the permissions on that memory. So, we have read and write permissions on that memory region. Then this

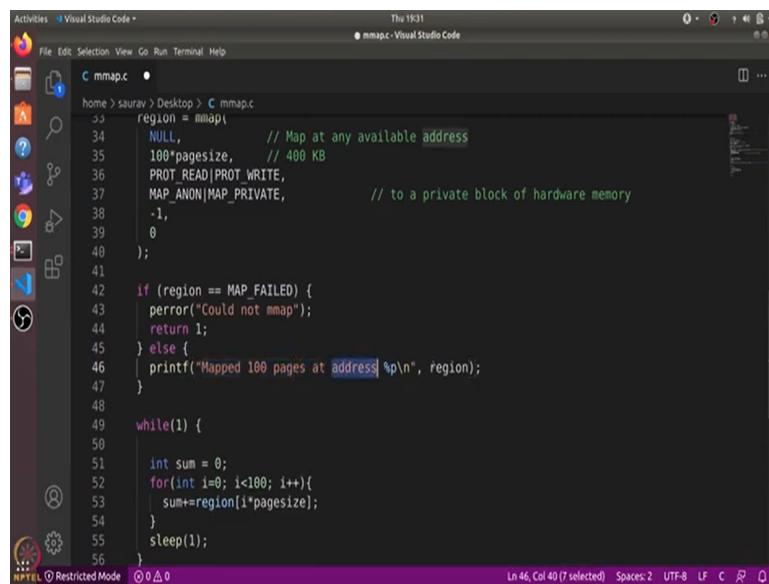
indicates whether this memory is backed by some file or not, so we have used `MAP_ANONYMOUS` which means that it is not backed by any file it is just a chunk of memory.

And it is `MAP_PRIVATE` which means that it is not shared with other processes and because this memory is not backed by any file, so we use `-1` for file descriptor because there is no file with which we want to back this memory and we use `0` as offset. So, these two arguments are used when the memory region is actually backed by some file.

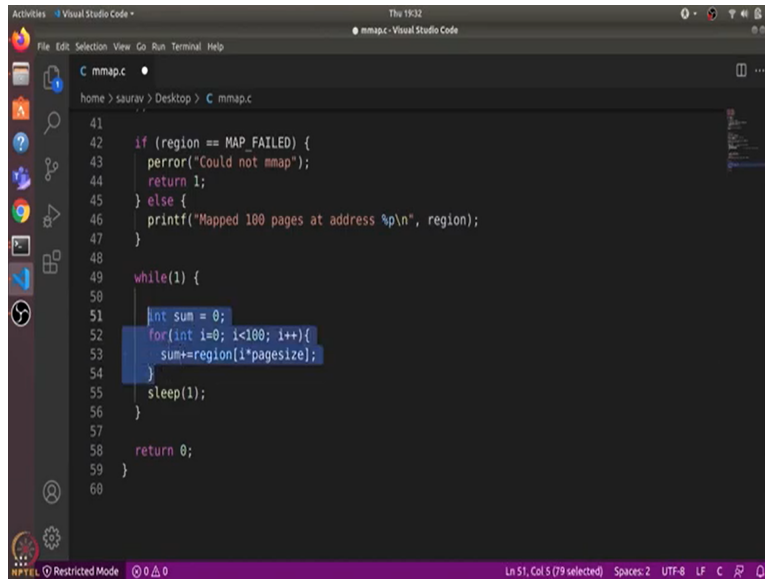
(Refer Slide Time: 02:42)



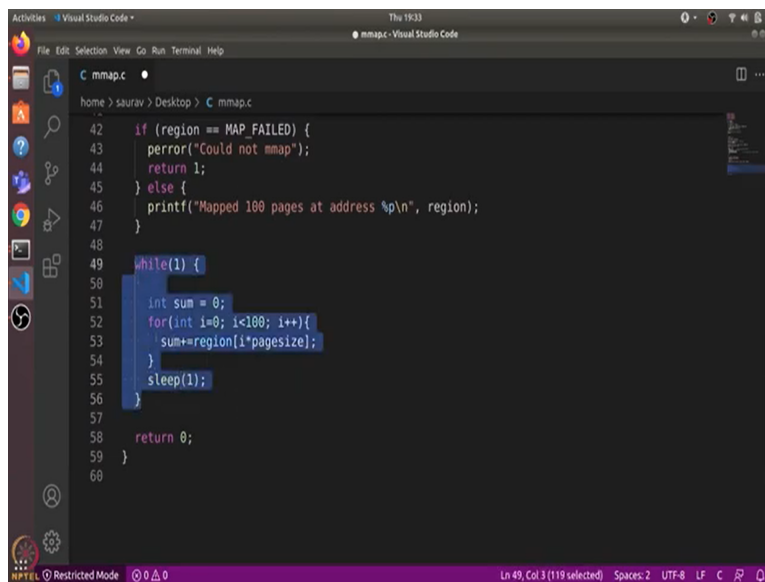
```
30 printf("System page size: %zu bytes\n", pagesize);
31 printf("My pid: %u\n", pid);
32
33 region = mmap(
34     NULL,           // Map at any available address
35     100*pagesize,   // 400 KB
36     PROT_READ|PROT_WRITE,
37     MAP_ANON|MAP_PRIVATE, // to a private block of hardware memory
38     -1,
39     0
40 );
41
42 if (region == MAP_FAILED) {
43     perror("Could not mmap");
44     return 1;
45 } else {
46     printf("Mapped 100 pages at address %p\n", region);
47 }
48
49 while(1) {
50
51     int sum = 0;
52     for(int i=0; i<100; i++){
```



```
53         sum+=region[i*pagesize];
54     }
55     sleep(1);
56 }
```



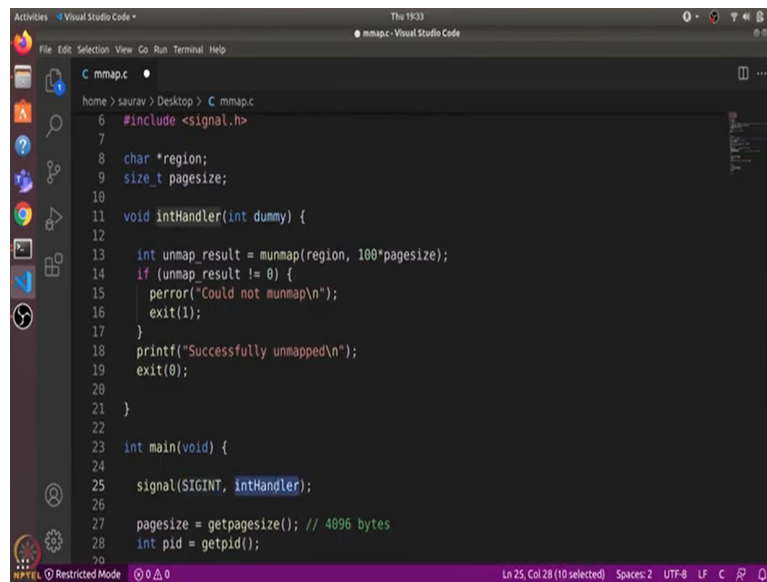
```
41
42 if (region == MAP_FAILED) {
43     perror("Could not mmap");
44     return 1;
45 } else {
46     printf("Mapped 100 pages at address %p\n", region);
47 }
48
49 while(1) {
50
51     int sum = 0;
52     for(int i=0; i<100; i++){
53         sum+=region[i*pagesize];
54     }
55     sleep(1);
56 }
57
58 return 0;
59 }
60
```



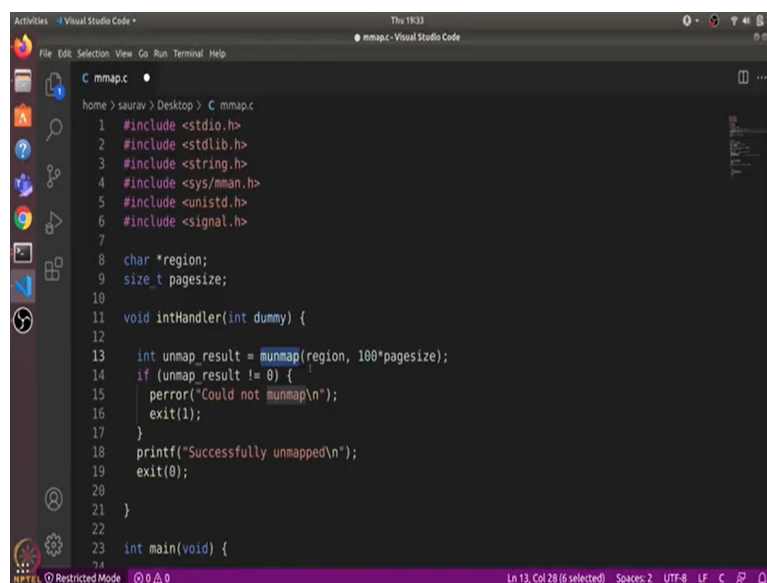
```
42 if (region == MAP_FAILED) {
43     perror("Could not mmap");
44     return 1;
45 } else {
46     printf("Mapped 100 pages at address %p\n", region);
47 }
48
49 while(1) {
50
51     int sum = 0;
52     for(int i=0; i<100; i++){
53         sum+=region[i*pagesize];
54     }
55     sleep(1);
56 }
57
58 return 0;
59 }
60
```

Now we check if the return value is map failed then we print out an error could not map and return otherwise we print out that we have mapped hundred pages at this particular address. And then we have this while loop what this while loop does is it iterates over 0 to 100 and accesses the first byte of every allocated page and adds it to sum. So, this is just to make sure that we are accessing every page and then it sleeps for one second and it repeats it again and again.

(Refer Slide Time: 03:14)



```
6 #include <signal.h>
7
8 char *region;
9 size_t pagesize;
10
11 void intHandler(int dummy) {
12
13     int unmap_result = munmap(region, 100*pagesize);
14     if (unmap_result != 0) {
15         perror("Could not munmap\n");
16         exit(1);
17     }
18     printf("Successfully unmapped\n");
19     exit(0);
20 }
21
22
23 int main(void) {
24
25     signal(SIGINT, intHandler);
26
27     pagesize = getpagesize(); // 4096 bytes
28     int pid = getpid();
```



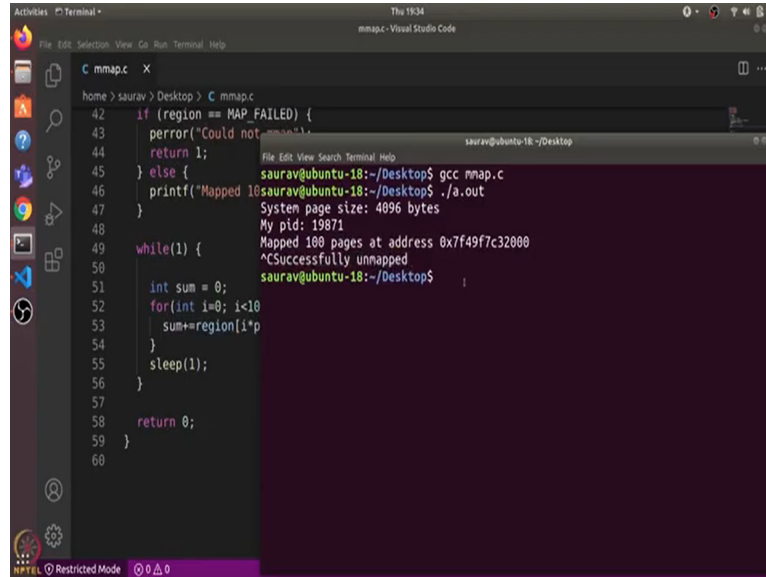
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <sys/mman.h>
5 #include <unistd.h>
6 #include <signal.h>
7
8 char *region;
9 size_t pagesize;
10
11 void intHandler(int dummy) {
12
13     int unmap_result = munmap(region, 100*pagesize);
14     if (unmap_result != 0) {
15         perror("Could not munmap\n");
16         exit(1);
17     }
18     printf("Successfully unmapped\n");
19     exit(0);
20 }
21
22
23 int main(void) {
```

So, the program will be stuck here so that we can analyze certain aspects of the program. But what if we want to exit the program then we can use control plus c and because we have mapped some memory and we need to unmap that before exiting the program. So, we have defined this signal handler which handles the SIGINT which is control plus c and what this handler does is it unmaps the memory region using unmap call.

And it takes the address of the region and the size of the memory and if the unmap result is not equal to 0, then we print out the error that could not unmap otherwise the printout are

successfully unmapped and then we exit the program. So, that is the complete mmap dot c code. So, let us compile it and execute it.

(Refer Slide Time: 04:08)



The screenshot shows the Visual Studio Code editor with a file named `mmap.c` open. The code in the editor is as follows:

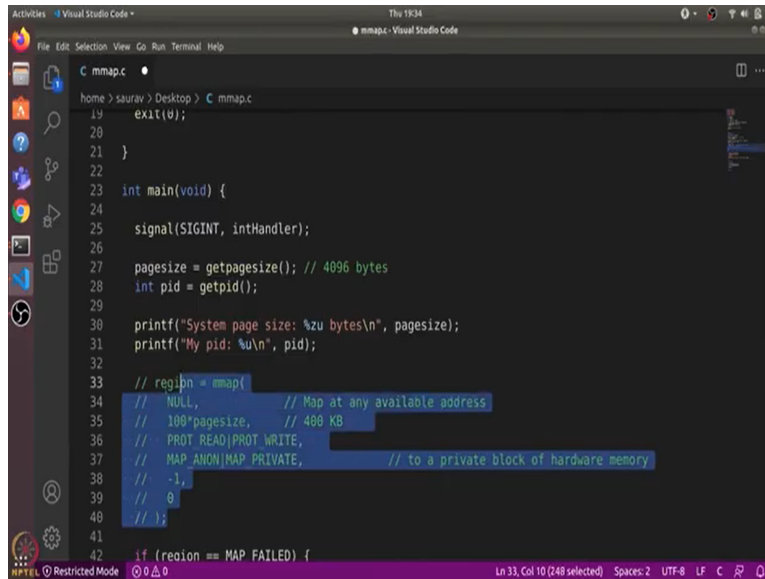
```
42 if (region == MAP_FAILED) {
43     perror("Could not mmap");
44     return 1;
45 } else {
46     printf("Mapped 100 pages at address %p\n", region);
47 }
48
49 while(1) {
50     int sum = 0;
51     for(int i=0; i<10; i++)
52         sum+=region[i]*p;
53     sleep(1);
54 }
55
56 return 0;
57 }
58 }
```

Below the editor, a terminal window is open, showing the execution of the program. The terminal output is:

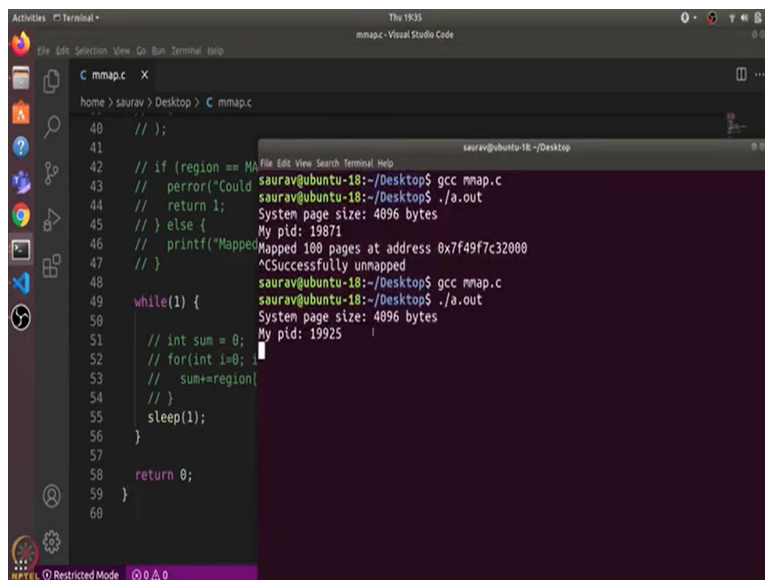
```
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19871
Mapped 100 pages at address 0x7f49f7c32000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$
```

So, I will open the terminal and compile it using `gcc mmap dot c` and then I will run the `a dot out`. So, it shows that system page size is 4096 bytes pid of this process is 19871. And then we have mapped some 100 pages at this particular virtual address and we can use `ctrl c` to exit the program and it prints successfully unmapped. So that is the `mmap dot c` program. Now let us try to analyze this program further.

(Refer Slide Time: 04:40)



```
19  exit(0);
20
21 }
22
23 int main(void) {
24
25     signal(SIGINT, intHandler);
26
27     pagesize = getpagesize(); // 4096 bytes
28     int pid = getpid();
29
30     printf("System page size: %zu bytes\n", pagesize);
31     printf("My pid: %u\n", pid);
32
33     // region = mmap(
34     //     NULL, // Map at any available address
35     //     100*pagesize, // 400 KB
36     //     PROT_READ|PROT_WRITE,
37     //     MAP_ANON|MAP_PRIVATE, // to a private block of hardware memory
38     //     -1,
39     //     0
40     // );
41
42     if (region == MAP_FAILED) {
```

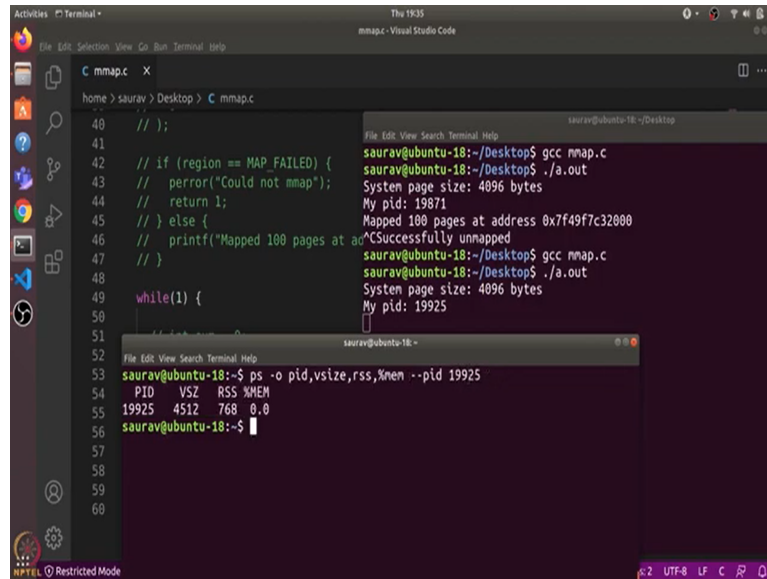


```
48 // };
49
50 // if (region == MAP_FAILED) {
51 //     perror("Could not map region");
52 //     return 1;
53 // } else {
54 //     printf("Mapped %zu pages at address %p\n",
55 //           region/pagesize, region);
56 // }
57
58 while(1) {
59     // int sum = 0;
60     // for(int i=0; i<region; i++) {
61     //     sum+=region[i];
62     // }
63     sleep(1);
64 }
65
66 return 0;
67 }
```

```
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19871
Mapped 100 pages at address 0x7f49f7c32000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19925
```

So, what I will do is I will comment out this mmap part so that it does not allocate any memory and I will comment out this part as well and this part as well. So now, the program just prints out the page size and pid and just keeps running this while loop. And I will also comment out the signal handler. So now, let us compile this program and run this. So, it shows 1995 as the pid.

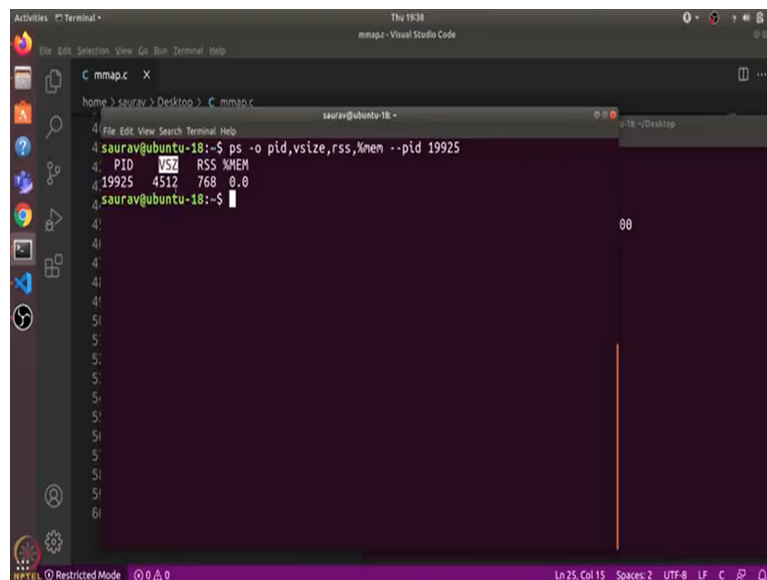
(Refer Slide Time: 05:17)



```
40 // );
41
42 // if (region == MAP_FAILED) {
43 //     perror("Could not mmap");
44 //     return 1;
45 // } else {
46 //     printf("Mapped 100 pages at address %x\n", (int) region);
47 // }
48
49 while(1) {
50     // if (region == MAP_FAILED) {
51     //     perror("Could not mmap");
52     //     return 1;
53     // } else {
54     //     printf("Mapped 100 pages at address %x\n", (int) region);
55     // }
56 }
57
58
59
60
```

```
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19871
Mapped 100 pages at address 0x7f49f7c32000
Successfully unmapped
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19925
```

```
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,%mem --pid 19925
  PID  VSZ  RSS  %MEM
19925 4512  768   0.0
saurav@ubuntu-18:~$
```

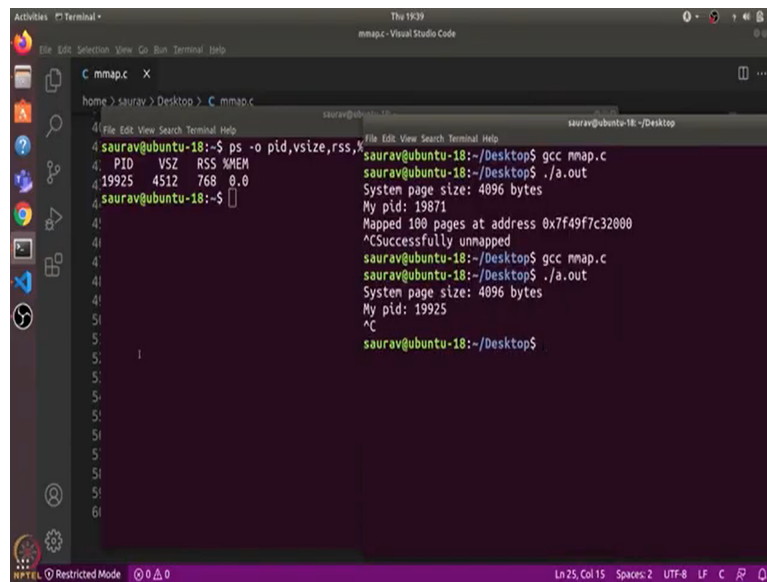


```
4 saurav@ubuntu-18:~$ ps -o pid,vsize,rss,%mem --pid 19925
  PID  VSZ  RSS  %MEM
19925 4512  768   0.0
saurav@ubuntu-18:~$
```

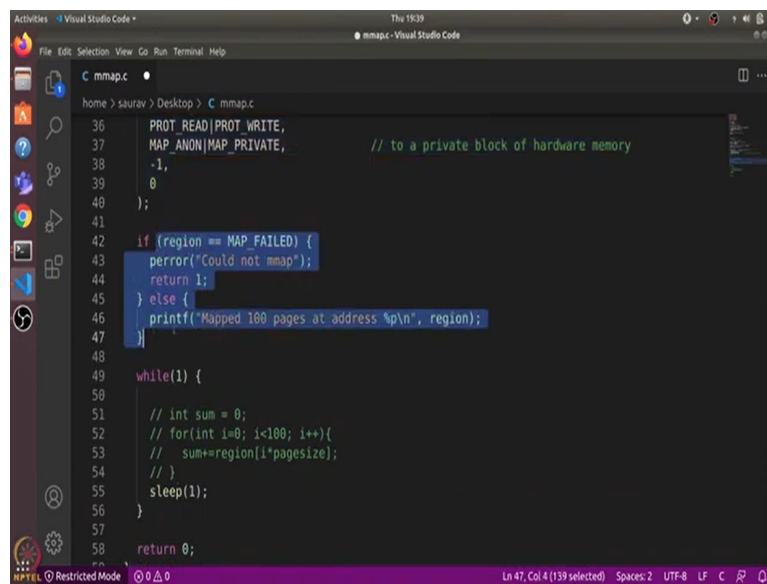
Now I will open another terminal and I have written this command here so this is just the ps command and this hyphen o is used to define the output. So, I want pid the virtual memory size which is the vss and RSS which is the resident set size and the percentage of memory used in the output of this ps command. And I want all of this information for this particular pid.

So, I will change this pid with 19925. So, this shows me VSZ which is the virtual memory size this is the amount of memory which a program can access which is 4512 kilobytes. And then, when a program actually uses some memory that is when it adds up in RSS which is the resident set size. So, RSS is 768 kilobytes which means that program can access 4512 kilobytes of memory but it is actually using just 768 kilobytes of the memory.

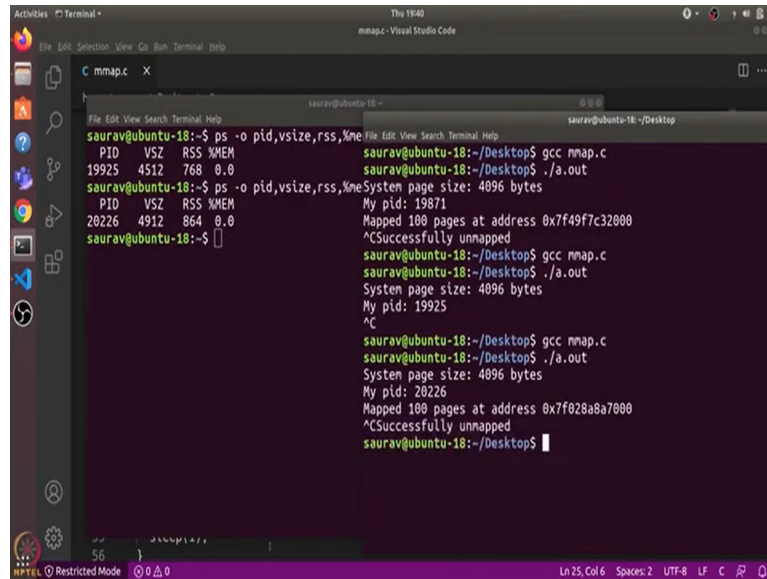
(Refer Slide Time: 06:17)



```
home > saurav > Desktop > C mmap.c
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,%
PID    VSZ    RSS %MEM
19925  4512   768  0.0
saurav@ubuntu-18:~$
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19871
Mapped 100 pages at address 0x7f49f7c32000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19925
^C
saurav@ubuntu-18:~/Desktop$
```



```
36  PROT_READ|PROT_WRITE,
37  MAP_ANON|MAP_PRIVATE,           // to a private block of hardware memory
38  -1,
39  0
40  );
41
42  if (region == MAP_FAILED) {
43      perror("Could not mmap");
44      return 1;
45  } else {
46      printf("Mapped 100 pages at address %p\n", region);
47  }
48
49  while(1) {
50
51      // int sum = 0;
52      // for(int i=0; i<100; i++){
53      //     sum+=region[i*pagesize];
54      // }
55      sleep(1);
56  }
57
58  return 0;
```

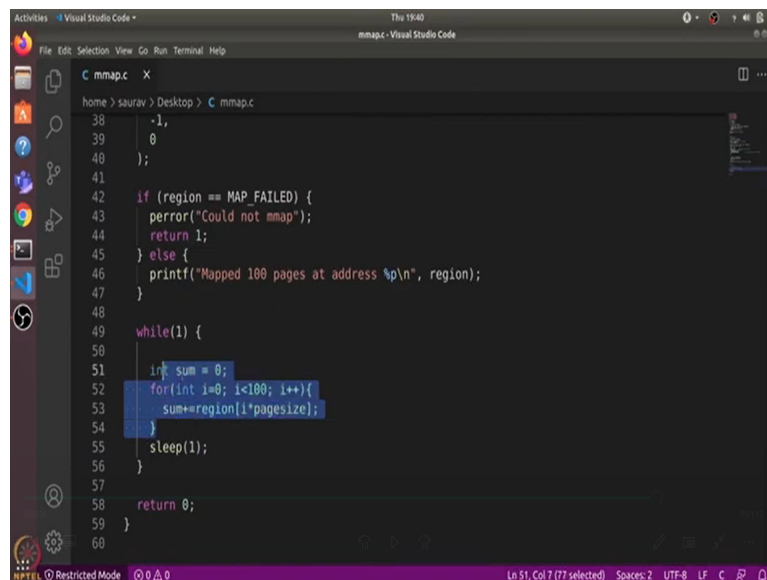



```
Activities Terminal • Thu 19:40
mmap.c - Visual Studio Code

C mmap.c x
File Edit View Search Terminal Help

saaurav@ubuntu-18:~$ ps -o pid,vsize,rss,%me
PID    VSZ    RSS %MEM
19925   4512   768  0.0
saaurav@ubuntu-18:~$ ps -o pid,vsize,rss,%me
PID    VSZ    RSS %MEM
20226   4912   864  0.0
saaurav@ubuntu-18:~$

saaurav@ubuntu-18:~/Desktop$ gcc mmap.c
saaurav@ubuntu-18:~/Desktop$ ./a.out
My pid: 19871
Mapped 100 pages at address 0x7f49f7c32000
^CSuccessfully unmapped
saaurav@ubuntu-18:~/Desktop$ gcc mmap.c
saaurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19925
^C
saaurav@ubuntu-18:~/Desktop$ gcc mmap.c
saaurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 20226
Mapped 100 pages at address 0x7f028a8a7000
^CSuccessfully unmapped
saaurav@ubuntu-18:~/Desktop$
```



```
Activities Visual Studio Code • Thu 19:40
mmap.c - Visual Studio Code

C mmap.c x
home > saaurav > Desktop > C mmap.c
38     -1,
39     0
40 );
41
42 if (region == MAP_FAILED) {
43     perror("Could not mmap");
44     return 1;
45 } else {
46     printf("Mapped 100 pages at address %p\n", region);
47 }
48
49 while(1) {
50
51     int sum = 0;
52     for(int i=0; i<100; i++){
53         sum+=region[i*pagesize];
54     }
55     sleep(1);
56 }
57
58 return 0;
59 }
60
```

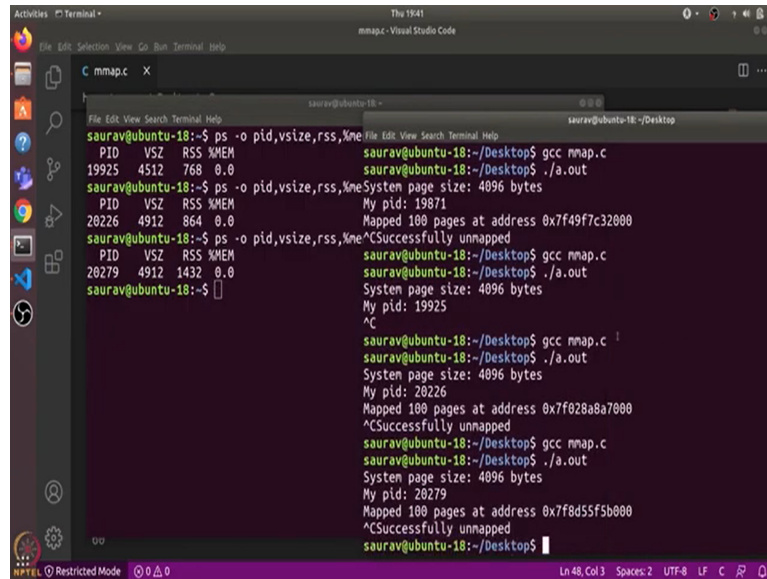
So, here we see that VSZ did increase by 400 kilobytes which is the 400 kilobytes extra memory that we allocated using the map call and RSS just increased by approximately 100 which is due to uncommenting the other parts of the code. And it does not include the 400 kilobytes of memory which was allocated using mmap. Now let us stop this as well and let us uncomment the remaining part now we are allocating memory and we are also accessing the memory in a while loop.

(Refer Slide Time: 07:34)

```
File Edit View Search Terminal Help
C mmap.c x
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,Mem
PID VSZ RSS %MEM
19925 4512 768 0.0
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,Mem
PID VSZ RSS %MEM
20226 4912 864 0.0
saurav@ubuntu-18:~$

saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19871
Mapped 100 pages at address 0x7f49f7c32000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19925
^C
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 20226
Mapped 100 pages at address 0x7f028a8a7000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 20279
Mapped 100 pages at address 0x7f8d55f5b000
```

```
File Edit View Search Terminal Help
C mmap.c x
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,Mem --pid 19925
PID VSZ RSS %MEM
19925 4512 768 0.0
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,Mem --pid 20226
PID VSZ RSS %MEM
20226 4912 864 0.0
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,Mem --pid 20279
PID VSZ RSS %MEM
20279 4912 1432 0.0
saurav@ubuntu-18:~$
```



```
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,vmem
PID    VSZ    RSS    %MEM
19925   4512   768    0.0
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,vmem
PID    VSZ    RSS    %MEM
20226   4912   864    0.0
saurav@ubuntu-18:~$ ps -o pid,vsize,rss,vmem
PID    VSZ    RSS    %MEM
20279   4912   1432   0.0
saurav@ubuntu-18:~$
```

```
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19871
Mapped 100 pages at address 0x7f49f7c32000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 19925
^C
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 20226
Mapped 100 pages at address 0x7f028a8a7000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$ gcc mmap.c
saurav@ubuntu-18:~/Desktop$ ./a.out
System page size: 4096 bytes
My pid: 20279
Mapped 100 pages at address 0x7f8d55f5b000
^CSuccessfully unmapped
saurav@ubuntu-18:~/Desktop$
```

So now, let us compile and use this 20279 as the pid here. So now, you can see that there is a significant increase in RSS because now it includes the 400 kilobytes of memory that is being accessed by the program so we see a significant increase in both VSZ and RSS. So, that is it for this video thanks and have a nice day.