

sender.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <sys/mman.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include "protocol.h"
8
9
10 int main()
11 {
12     int fd = shm_open(NAME, O_CREAT | O_EXCL | O_RDWR, 0600);
13     if (fd < 0) {
14         perror("shm_open()");
15         return EXIT_FAILURE;
16     }
17
18     int size = ARRAY_S + STR_L;
19
20     ftruncate(fd, size);
21
22     void *data = mmap(0, size, PROT_READ | PROT_WRITE,
23                       MAP_SHARED, fd, 0);
24
25     if(data == (void *)-1){
26         perror("mmap()");
27         return EXIT_FAILURE;
28     }
29
30     int *array = (int *)data;
31     char *msg = (char *) (data + ARRAY_S); // get address of message
32
33     printf("sender address: %p\n", data);
34
35     // write array in shared memory
36     for (int i = 0; i < NUM; i++) {
37         array[i] = i*i;
38     }
39
40     // write message in shared memory
41     strncat(msg, "Hi Shared Memory!", STR_L);
42
43     munmap(data, size);
44
45     close(fd);
46     return EXIT_SUCCESS;
47 }
```

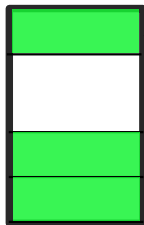
receiver.c

```
1  #include "protocol.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <fcntl.h>
5  #include <sys/mman.h>
6  #include <unistd.h>
7
8  int main()
9  {
10     int fd = shm_open(NAME, O_RDONLY, 0666);
11     if (fd < 0) {
12         perror("shm_open()");
13         return EXIT_FAILURE;
14     }
15
16     int size = ARRAY_S + STR_L;
17
18     void *data = mmap(0, size, PROT_READ, MAP_SHARED, fd, 0);
19     if (data == (void *)-1) {
20         perror("mmap()");
21         return EXIT_FAILURE;
22     }
23
24     int *array = (int *)data;
25     char *msg = (char *) (data + ARRAY_S);
26
27     printf("receiver address: %p\n", data);
28
29     for (int i = 0; i < NUM; i++) {
30         printf("num%d: %d\n", i, array[i]);
31     }
32
33     printf("msg: %s\n", msg);
34
35     munmap(data, size);
36
37     close(fd);
38
39     // delete file
40     shm_unlink(NAME);
41     return EXIT_SUCCESS;
42 }
```

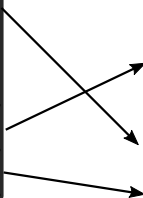
makefile

```
1 CFLAGS= -lrt -Wall -Wextra
2
3 all: receiver sender
4
5 receiver: receiver.c
6 | $(CC) receiver.c -o receiver $(CFLAGS)
7
8 sender: sender.c
9 | $(CC) sender.c -o sender $(CFLAGS)
10
11
12 clean:
13 | rm receiver sender
```

Sender addr space



Main Memory

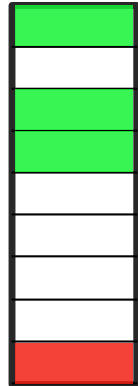


```
int fd = shm_open(NAME,  
O_CREAT | O_EXCL | O_RDWR, 0600);  
if (fd<0) {  
    perror("shm_open()");  
    return EXIT_FAILURE;  
}  
  
int size = ARRAY_S + STR_L;  
  
ftruncate(fd, size);
```

Sender addr space



Main Memory

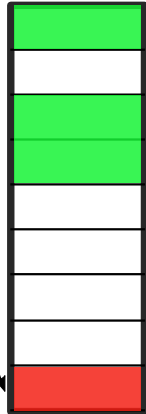


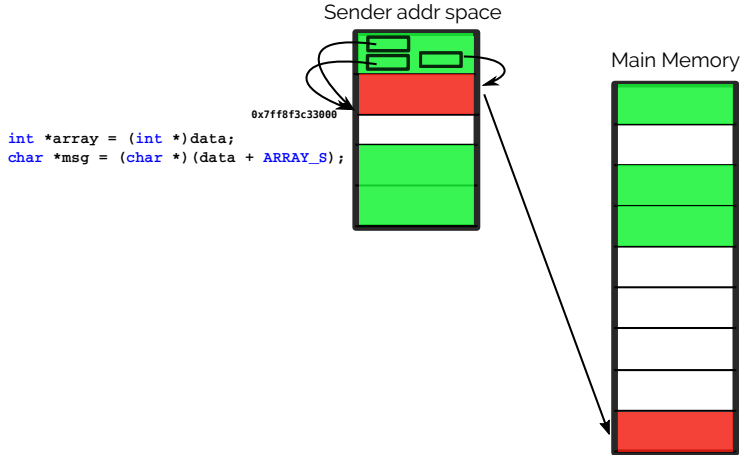
```
void *data = mmap(0, size,  
    PROT_READ | PROT_WRITE,  
    MAP_SHARED, fd, 0);
```

Sender addr space

0x7ff8f3c33000

Main Memory





```
printf("sender address: %p\n", data);
```

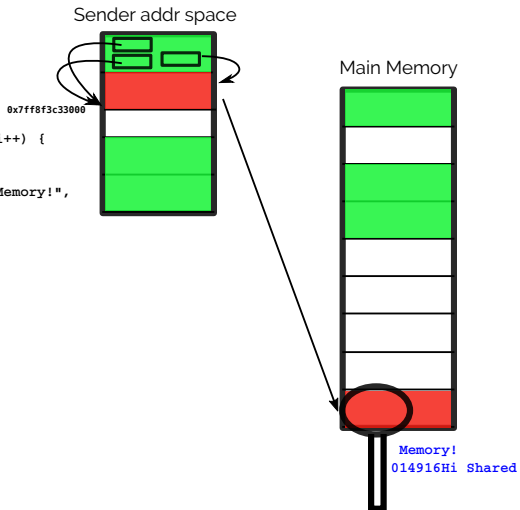
```
sender address: 0x7ff8f3c33000
```

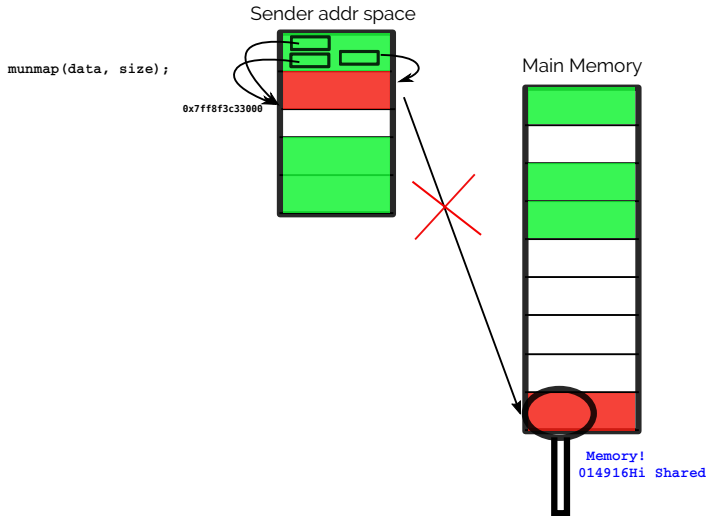


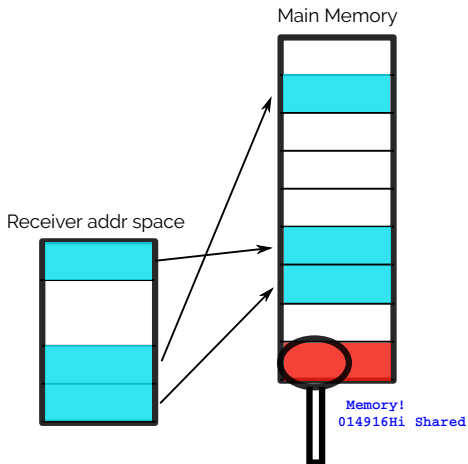
```

for (int i = 0; i < NUM; i++) {
    array[i] = i*i;
}
strncat(msg, "Hi Shared Memory!",
STR_L);

```







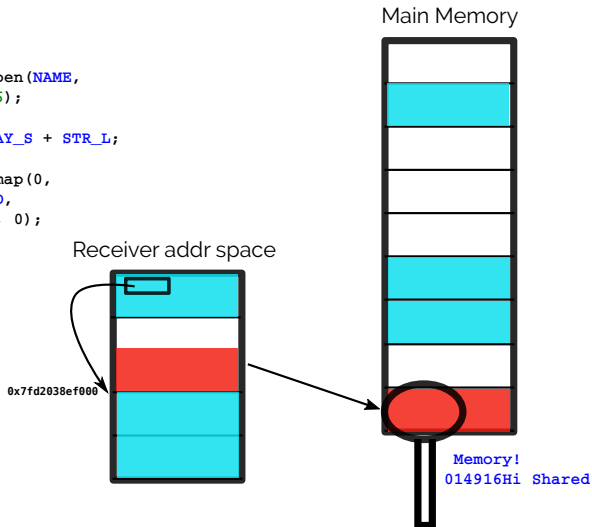
```

int fd = shm_open(NAME,
    O_RDONLY, 0666);

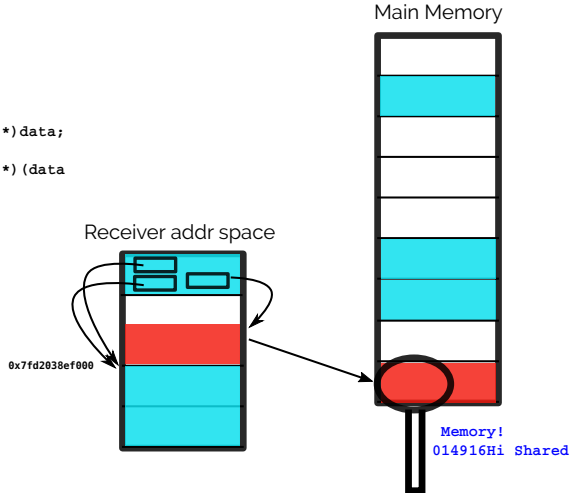
int size = ARRAY_S + STR_L;

void *data = mmap(0,
    size, PROT_READ,
    MAP_SHARED, fd, 0);

```



```
int *array = (int *)data;  
char *msg = (char *) (data  
+ ARRAY_S);
```



```
printf("receiver address: %p\n", data);
```

```
receiver address: 0x7fd2038ef000
```

```
for (int i = 0; i < NUM; i++)  
{  
    printf("num%d: %d\n", i,  
array[i]);  
}  
  
printf("msg: %s\n", msg);
```

```
num0: 0  
num1: 1  
num2: 4  
num3: 9  
num4: 16  
msg: Hi Shared Memory!
```

`munmap(data, size);`

0x7fd2038ef000

Receiver addr space

Main Memory

Memory!
014916Hi Shared

What happens if we try to reference the red region in the address space now?

- for example:

```
printf("msg: %s\n", msg);
```

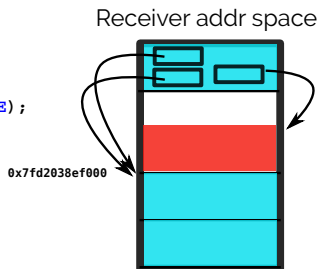
What happens if we try to reference the red region in the address space now?

- for example:

```
printf("msg: %s\n", msg);
```

```
[1] 18439 segmentation fault ./receiver
```

```
close(fd);  
shm_unlink(NAME);
```



Main Memory

