how to use shared memory to communicate between two processes

Join us for a one-day coding Tournament in Budapest.
Compete for $100k/yr job offers
www.crossover.com
CROSS|OVER
JOIN NOW

I am trying to communicate between two processes. I am trying to save data(like name, phone number, address) to shared memory in one process and trying to print that data through other process.

process1.c

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main ()
{
  int segment_id;
  char* shared_memory[3];
  int segment_size;
  key_t shm_key;
  int i=0;
  const int shared_segment_size = 0x6400;
  /* Allocate a shared memory segment. */
  segment_id = shmget (shm_key, shared_segment_size,
            IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  /* Attach the shared memory segment. */
  shared_memory[3] = (char*) shmat (segment_id, 0, 0);
  printf ("shared memory attached at address %p\n", shared_memory);
  /* Write a string to the shared memory segment. */
   sprintf(shared_memory[i], "maddy \n");
   sprintf(shared_memory[i+1], "73453916\n");
   sprintf(shared_memory[i+2], "america\n");

  /*calling the other process*/
  system("./process2");

  /* Detach the shared memory segment. */
  shmdt (shared_memory);
  /* Deallocate the shared memory segment.*/
  shmctl (segment_id, IPC_RMID, 0);

  return 0;
}
```

process2.c

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main ()
{
  int segment_id;
  char* shared_memory[3];
  int segment_size;
  int i=0;
  key_t shm_key;
  const int shared_segment_size = 0x6400;
  /* Allocate a shared memory segment. */
  segment_id = shmget (shm_key, shared_segment_size,
              S_IRUSR | S_IWUSR);
  /* Attach the shared memory segment. */
  shared_memory[3] = (char*) shmat (segment_id, 0, 0);
  printf ("shared memory22 attached at address %p\n", shared_memory);
   printf ("name=%s\n", shared_memory[i]);
   printf ("%s\n", shared_memory[i+1]);
   printf ("%s\n", shared_memory[i+2]);
  /* Detach the shared memory segment. */
  shmdt (shared_memory);
   return 0;
}
```

But I am not getting the desired output. the output which i got is:

```
shared memory attached at address 0x7fff0fd2d460
Segmentation fault
```

Anyone can please help me with this. Is this the correct way of initializing `shared_memory[3]` .

Thank you.

c ipc process shared-memory

edited Aug 30 '11 at 0:35
osgx
46.2k 25 181 340

asked Aug 30 '11 at 0:31
maddy
70 1 2 9

3 Answers

```
char* shared_memory[3];
...
shared_memory[3] = (char*) shmat (segment_id, 0, 0);
```

You declare `shared_memory` as an array capable of holding three pointers to char, but what you actually do with it is to write a pointer *one place behind the end of the array*. Since there is no telling what the memory there is otherwise used for, what happens next is generally unpredictable.

Things go conclusively bad afterwards when you try to make use of the pointers in `shared_memory[0]` through `shared_memory[2]` , because those pointers have never been initialized. They are filled with meaningless garbage from the stack -- thus the segmentation fault.

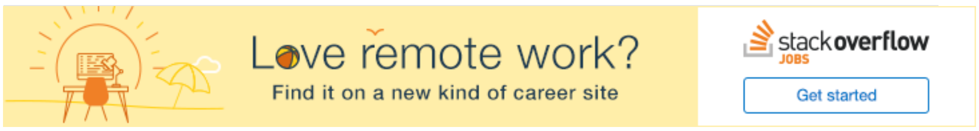
It seems, in general, that you're failing to distinguish between the array and its elements. You should go and make yourself *a lot* more comfortable with arrays and pointers in sequential code before you try your hand at shared-memory IPC.

Note that shared memory is one of the more easy-to-get-wrong ways of doing IPC out there. Unless you have rigid efficiency constraints and are going to exchange *a lot* of data, it's much easier to work with pipes, named pipes, or sockets.

edited Aug 30 '11 at 0:48

answered Aug 30 '11 at 0:36
Henning Makholm
16.9k 2 30 63

Thank you Henning. – maddy Aug 30 '11 at 1:01



The other two answers told you what's wrong, but I want to give you a runnable code. You can modify it to pass anything, the principle is you need to save the length of every element you passed to the other side.

//write.c

```
#include <stdio.h>
#include <string.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main ()
{
  key_t shm_key = 6166529;
  const int shm_size = 1024;

  int shm_id;
  char* shmaddr, *ptr;
  int next[2];

  printf ("writer started.\n");

  /* Allocate a shared memory segment. */
  shm_id = shmget (shm_key, shm_size, IPC_CREAT | S_IRUSR | S_IWUSR);

  /* Attach the shared memory segment. */
  shmaddr = (char*) shmat (shm_id, 0, 0);

  printf ("shared memory attached at address %p\n", shmaddr);

  /* Start to write data. */
  ptr = shmaddr + sizeof (next);
  next[0] = sprintf (ptr, "mandy") + 1;
  ptr += next[0];
  next[1] = sprintf (ptr, "73453916") + 1;
  ptr += next[1];
```

```
    sprintf (ptr, "amarica");
    memcpy(shmaddr, &next, sizeof (next));
    printf ("writer ended.\n");

    /*calling the other process*/
    system("./read");

    /* Detach the shared memory segment. */
    shmdt (shmaddr);
    /* Deallocate the shared memory segment.*/
    shmctl (shm_id, IPC_RMID, 0);

    return 0;
}
```

//read.c

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main ()
{
    key_t shm_key = 6166529;
    const int shm_size = 1024;

    int shm_id;
    char* shmaddr, *ptr;
    char* shared_memory[3];
    int *p;

    /* Allocate a shared memory segment. */
    shm_id = shmget (shm_key, shm_size, IPC_CREAT | S_IRUSR | S_IWUSR);

    /* Attach the shared memory segment. */
    shmaddr = (char*) shmat (shm_id, 0, 0);

    printf ("shared memory attached at address %p\n", shmaddr);

    /* Start to read data. */
    p = (int *)shmaddr;
    ptr = shmaddr + sizeof (int) * 2;
    shared_memory[0] = ptr;
    ptr += *p++;
    shared_memory[1] = ptr;
    ptr += *p;
    shared_memory[2] = ptr;
    printf ("0=%s\n", shared_memory[0]);
    printf ("1=%s\n", shared_memory[1]);
    printf ("2=%s\n", shared_memory[2]);

    /* Detach the shared memory segment. */
    shmdt (shmaddr);
    return 0;
}
```

//Result of run:

```
> [lex:shm]$ ./write
> writer started.
> shared memory attached at address 0x7fa20103b000
> writer ended.
> shared memory attached at address0x7fd85e2eb000
> 0=mandy
> 1=73453916
> 2=amarica
```

answered Jan 29 '15 at 7:39
Lex Pro
138 1 6

You should be reserving enough shared memory to exchange the *data*. Processes aren't supposed to access each others memory, even if using shared pointers. Keep in mind only the raw data you write during runtime is shared, there's no type checking or any other metadata passed. You can use a common structure, if your data allows it using fixed-size arrays, to access the data more easily. Otherwise, you'll have to manually marshal the data between the processes.

answered Aug 30 '11 at 11:41
Ioan
1,906 13 30