How to use shared memory with Linux in C

I have a bit of an issue with one of my projects.

I have been trying to find a well documented example of using shared memory with fork() but to no success.

Basically the scenario is that when the user starts the program, I need to store two values in shared memory: current_path which is a char* and a file_name which is also char*.

Depending on the command arguments, a new process is kicked off with fork() and that process needs to read and modify the current_path variable stored in shared memory while the file_name variable is read only.

Is there a good tutorial on shared memory with example code (if possible) that you can direct me to?

Thanks, bleepzter

c linux fork shared-memory

asked Apr 13 '11 at 22:41
bleepzter
3,978 7 27 55

You may consider using threads instead of processes. Then the whole memory is shared with no further tricks. – elomage Feb 4 '14 at 12:22

6 Answers

The man page for `shmget` has pretty much everything you need to get started.

If you'd rather have a quickstart, here you have an explanation with examples.

The `shmget` approach is simpler, but kinda outdated. Using `mmap` is the more modern approach. `mmap` is more versatile, though not very intuitive. Here you have a good guide with examples.

edited Dec 20 '16 at 23:03 answered Apr 13 '11 at 22:43
slezica
32.2k 11 60 114

11 This is why Linux is so frustrating for inexperienced devs. The man page doesn't explain how to actually use it, and there is no sample code. :(– bleepzter Apr 13 '11 at 22:46

11 Haha I know what you mean, but it's actually because we're not used to reading manpages. When I learned to read them and got used to them, they became even more useful than lousy tutorials with particular demonstrations. I remember I got a 10/10 in my Operating Systems course using nothing but manpages for reference during the exam. – slezica Apr 13 '11 at 22:51

13 `shmget` is a really old-fashioned, and some would say deprecated, way to do shared memory... Better to use `mmap` and `shm_open`, plain files, or simply `MAP_ANONYMOUS`. – R.. Apr 13 '11 at 23:29

SysV shared memory (shmget) is extremely crufty and unnecessary, as mmap() works properly on Linux. – MarkR Apr 14 '11 at 5:39

3 @Mark @R You guys are right, I'll point that out in the answer for future reference. – slezica Apr 14 '11 at 21:16

Here is an example for shared memory. This might help :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1024  /* make it a 1K shared memory segment */

int main(int argc, char *argv[])
{
    key_t key;
    int shmid;
    char *data;
    int mode;

    if (argc > 2) {
        fprintf(stderr, "usage: shmdemo [data_to_write]\n");
        exit(1);
    }

    /* make the key: */
    if ((key = ftok("hello.txt", 'R')) == -1) /*Here the file must exist */
{
        perror("ftok");
        exit(1);
    }

    /*  create the segment: */
    if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
        perror("shmget");
        exit(1);
    }

    /* attach to the segment to get a pointer to it: */
    data = shmat(shmid, (void *)0, 0);
    if (data == (char *)(-1)) {
        perror("shmat");
        exit(1);
    }

    /* read or modify the segment, based on the command line: */
    if (argc == 2) {
        printf("writing to segment: \"%s\"\n", argv[1]);
        strncpy(data, argv[1], SHM_SIZE);
    } else
        printf("segment contains: \"%s\"\n", data);

    /* detach from the segment: */
    if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
    }

    return 0;
}
```

Steps : 1- Use ftok to convert a pathname and a project identifier to a System V IPC key

2- Use shmget which allocates a shared memory segment

3- Use shmat to attache the shared memory segment identified by shmid to the address space of the calling process

4- Do the operations on the memory area

5- Detach using shmdt

edited Oct 6 '14 at 9:36 answered Mar 21 '14 at 11:28
Mayank
985 7 11

Why are you casting 0 into a void* instead of using NULL ? – Clément Péau Mar 28 at 15:45

Chapter 5 of the book "Advanced Linux Programming" has a nice introduction to IPC with Linux (entire book as pdf)

edited Feb 12 '16 at 3:00
Janus Troelsen
11k 4 79 125

answered Apr 13 '11 at 22:47
sl0815
492 1 8 21

A good summary, but it lacks the newer shared stuff mentioned in the other answer. – Matt Joiner Feb 28 '12 at 12:43

2 The link is dead. – Aleksandr Kovalev Nov 10 '15 at 20:26

1 @AleksandrKovalev The link has been fixed. – Janus Troelsen Feb 12 '16 at 3:07

These are includes for using shared memory

```
#include<sys/ipc.h>
#include<sys/shm.h>

int shmid;
int shmkey = 12222;//u can choose it as your choice

int main()
{
  //now your main starting
  shmid = shmget(shmkey,1024,IPC_CREAT);
  // 1024 = your preferred size for share memory
  // IPC_CREAT  its a flag to create shared memory

  //now attach a memory to this share memory
  char *shmpointer = shmat(shmid,NULL);

  //do your work with the shared memory
  //read -write will be done with the *shmppointer
  //after your work is done deattach the pointer

  shmdt(&shmpointer, NULL);
```

edited Feb 4 '14 at 12:21 — Mat **144k** 24 255 292

answered Feb 4 '14 at 12:16 — Bharat **65** 1 2

try this code sample, I tested it, source: http://www.makelinux.net/alp/035

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

int main ()
{
  int segment_id;
  char* shared_memory;
  struct shmid_ds shmbuffer;
  int segment_size;
  const int shared_segment_size = 0x6400;

  /* Allocate a shared memory segment.  */
  segment_id = shmget (IPC_PRIVATE, shared_segment_size,
                 IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  /* Attach the shared memory segment.  */
  shared_memory = (char*) shmat (segment_id, 0, 0);
  printf ("shared memory attached at address %p\n", shared_memory);
  /* Determine the segment's size. */
  shmctl (segment_id, IPC_STAT, &shmbuffer);
  segment_size  =               shmbuffer.shm_segsz;
  printf ("segment size: %d\n", segment_size);
  /* Write a string to the shared memory segment.  */
  sprintf (shared_memory, "Hello, world.");
  /* Detach the shared memory segment.  */
  shmdt (shared_memory);

  /* Reattach the shared memory segment, at a different address.  */
  shared_memory = (char*) shmat (segment_id, (void*) 0x5000000, 0);
  printf ("shared memory reattached at address %p\n", shared_memory);
  /* Print out the string from shared memory.  */
  printf ("%s\n", shared_memory);
  /* Detach the shared memory segment.  */
  shmdt (shared_memory);

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

  return 0;
}
```

answered Mar 27 '16 at 6:01 — shakram02 **458** 6 13

Here's a mmap example:

```
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

/*
 * pvtmMmapAlloc - creates a memory mapped file area.
 * The return value is a page-aligned memory value, or NULL if there is a failure.
 * Here's the list of arguments:
 * @mmapFileName - the name of the memory mapped file
 * @size - the size of the memory mapped file (should be a multiple of the system page for
best performance)
 * @create - determines whether or not the area should be created.
 */
void* pvtmMmapAlloc (char * mmapFileName, size_t size, char create)
{
  void * retv = NULL;
  if (create)
  {
    mode_t origMask = umask(0);
    int mmapFd = open(mmapFileName, O_CREAT|O_RDWR, 00666);
    umask(origMask);
    if (mmapFd < 0)
    {
      perror("open mmapFd failed");
      return NULL;
    }
    if ((ftruncate(mmapFd, size) == 0))
    {
      int result = lseek(mmapFd, size - 1, SEEK_SET);
      if (result == -1)
      {
        perror("lseek mmapFd failed");
        close(mmapFd);
        return NULL;
      }

      /* Something needs to be written at the end of the file to
       * have the file actually have the new size.
       * Just writing an empty string at the current file position will do.
       * Note:
       *  - The current position in the file is at the end of the stretched
       *    file due to the call to lseek().
              *  - The current position in the file is at the end of the stretched
       *    file due to the call to lseek().
       *  - An empty string is actually a single '\0' character, so a zero-byte
       *    will be written at the last byte of the file.
       */
      result = write(mmapFd, "", 1);
      if (result != 1)
      {
        perror("write mmapFd failed");
        close(mmapFd);
        return NULL;
      }
      retv  =  mmap(NULL, size,
                  PROT_READ | PROT_WRITE, MAP_SHARED, mmapFd, 0);

      if (retv == MAP_FAILED || retv == NULL)
      {
        perror("mmap");
        close(mmapFd);
        return NULL;
      }
    }
  }
  else
  {
    int mmapFd = open(mmapFileName, O_RDWR, 00666);
    if (mmapFd < 0)
    {
      return NULL;
    }
    int result = lseek(mmapFd, 0, SEEK_END);
    if (result == -1)
    {
      perror("lseek mmapFd failed");
      close(mmapFd);
      return NULL;
    }
    if (result == 0)
    {
      perror("The file has 0 bytes");
      close(mmapFd);
      return NULL;
    }
    retv  =  mmap(NULL, size,
                PROT_READ | PROT_WRITE, MAP_SHARED, mmapFd, 0);

    if (retv == MAP_FAILED || retv == NULL)
    {
      perror("mmap");
      close(mmapFd);
      return NULL;
    }

    close(mmapFd);

  }
  return retv;
}
```

answered May 16 '16 at 13:58

Leo
483 5 18