OPERATING SYSTEMS

Interprocess Communication

Operating Systems

School of Data & Computer Science

Sun Yat-sen University

Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

中山大學 SUN YAT-SEN UNIVERSITY

Contents

Independent Processes and Cooperating Processes

- Processes executing concurrently in the operating system may be either independent processes (独立进程) or cooperating processes (合作进程).
  - A process is *independent* if it cannot affect or be affected by other processes executing in the system.
    - Any process that does not share data with any other process is independent.
  - A process is *cooperating* if it can affect or be affected by other processes executing in the system.
    - Any process that shares data with other processes is a cooperating process.

Independent Processes and Cooperating Processes

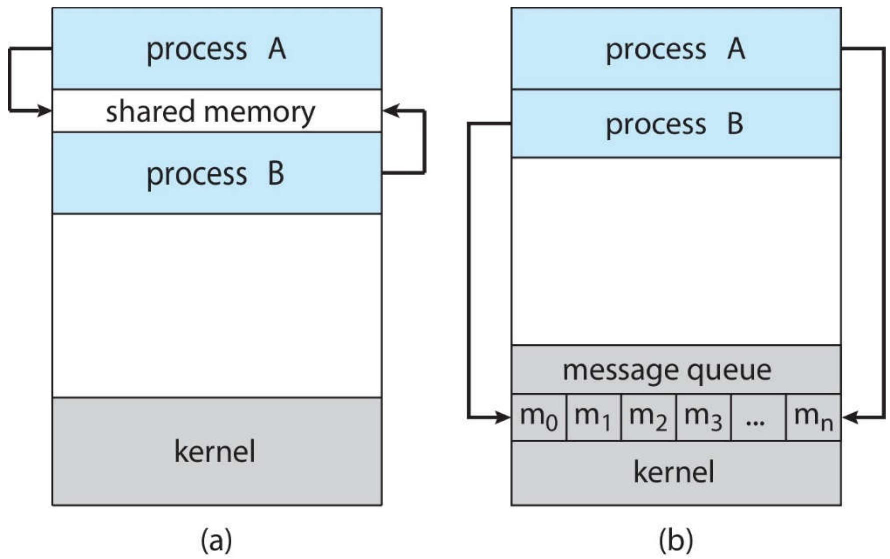
- Reasons for providing process cooperation:
  - Information sharing
    - concurrent access to information by several applications
  - Computation speedup
    - For a computer with multiple processing cores, breaking a particular task into subtasks and executing in parallel may speed up the computation.
  - Modularity
    - construct the system in a modular fashion, dividing the system functions into separate processes or threads
  - Convenience
    - Even an individual user may work on many tasks at the same time. For instance, a user may be editing, listening to music, and compiling in parallel.
- We will discuss cooperating processes and their synchronization in detail later (Lecture14 - Lecture19).

Interprocess Communication

- Cooperating processes require an *Interprocess Communication* (IPC) mechanism that will allow them to exchange data and information.
  - If two processes P and Q wish to communicate, they need to:
    - establish *communication link* between them
    - exchange *messages* via send/receive
  - Implementation of communication link:
    - physical link (e.g., shared memory, hardware bus)
    - logical link (e.g., logical properties)
  - Implementation questions:
    - How are links established?
    - Can a link be associated with more than two processes?
    - How many links can there be between every pair of communicating processes?
    - What is the capacity of a link?
    - Is the size of a message that the link can accommodate fixed or variable?
    - Is a link unidirectional or bi-directional?

Interprocess Communication

- There are two fundamental models of IPC.
  - *Shared Memory* / *Memory Sharing*
    - A region of memory, shared by cooperating processes, is established. Processes can then exchange information by reading and writing data to the shared region.
    - System calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required.



Interprocess Communication

- There are two fundamental models of IPC
  - *Message Passing*
    - Communication takes place by means of messages exchanged between cooperating processes.
    - useful for exchanging smaller amounts of data
    - easier to implement in a distributed system than shared memory
    - typically implemented using system calls and thus require the more time-consuming task of kernel intervention
    - better performance on multicore systems
      - the prefer mechanism for IPC on such systems

Shared-memory Systems

- Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment.
- Other processes that wish to communicate using this shared-memory segment must attach it to their address space and then exchange information by reading and writing data in the shared areas.
- The location and the form of the data are determined by these processes and are not under the operating system’s control.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.
  - They most keep *mutual exclusion* (Lecture14 - Lecture19).

Producer-Consumer Problem with Shared-memory

- The producer–consumer problem is a common paradigm for cooperating processes.
  - A *producer* process produces information that is consumed by a *consumer* process.
  - A FIFO *buffer* shared by these two processes is designed to be filled by the producer and emptied by the consumer.
- The producer and consumer are running concurrently and must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.
- Two types of buffers can be used.
  - unbounded buffer
    - with no practical limit on the size of the buffer
    - The consumer has to wait if the buffer is empty; the producer can always produce new items.
  - bounded buffer
    - with a fixed buffer size
    - The consumer must wait if the buffer is empty; the producer must wait if the buffer is full.

Producer-Consumer Problem with Shared-memory

- Shared data

```
#define BUFFER_SIZE 10

typedef struct {
    … …     /* item structure */
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- The shared buffer is implemented as a *circular array* with two logical pointers: in and out.
  - The variable in points to the next free position in the buffer; out points to the first full position in the buffer.
  - The buffer is empty when in is equal to out;
  - The buffer is full when ((in + 1) % BUFFER_SIZE) is equal to out.
  - This scheme allows at most BUFFER SIZE - 1 items in the buffer at the same time.

Producer-Consumer Problem with Shared-memory

- Producer:

```
item next_produced;
while (true) {
        … …   /* produce an item saved in next_produced */
    while (((in + 1) % BUFFER_SIZE) == out)
        ;     /* buffer full, do nothing */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
}
```

- Consumer:

```
item next_consumed;
while (true) {
    while (in == out)
        ;     /* buffer empty, do nothing */
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
              /* consume the item in next_consumed */
}
```

Linux: Shared Memory

- Linux IPCs Limits
  - The kernel level limits can be redefined in /etc/sysctl.conf

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ipcs -l

------ Messages Limits --------
max queues system wide = 32000
max size of message (bytes) = 8192
default max size of queue (bytes) = 16384

------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 18014398509465599
max total shared memory (kbytes) = 18014398509481980
min seg size (bytes) = 1

------ Semaphore Limits --------
max number of arrays = 32000
max semaphores per array = 32000
max semaphores system wide = 1024000000
max ops per semop call = 500
semaphore max value = 32767

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

2^{64}

Linux: Shared Memory

- Key ID

```
#include <sys/shm.h>
key_t ftok(const char *pathname, int id);
/* key_t is of type int. ftok() convert a pathname and a project
identifier to an IPC key */

key_t key = ftok("/home/myshm", 0x27);
if((key == -1) {
    perror("ftok()");
} else
    printf("key = 0x%x\n", key);
```

- Create

```
int shmget(key_t key, int size, int shmflg);
/* shmget() allocates a shared memory segment */
/* upper bound of size: 1.9G */

int shmid = shmget(IPC_PRIVATE, 4096, IPC_CREATE|IPC_EXCL|0660);
if(shmid == -1) {
    perror("shmget()");
}
```

Linux: Shared Memory

- Attach

```
void *shmat(int shmid, const void *shmaddr, int shmflg);

void *shmptr = shmat(shmid, 0, 0);
    /* shmaddr=0: attaching address is decided by kernel */
if(shmptr == (void *)(-1))
    perror("shmat()");
```

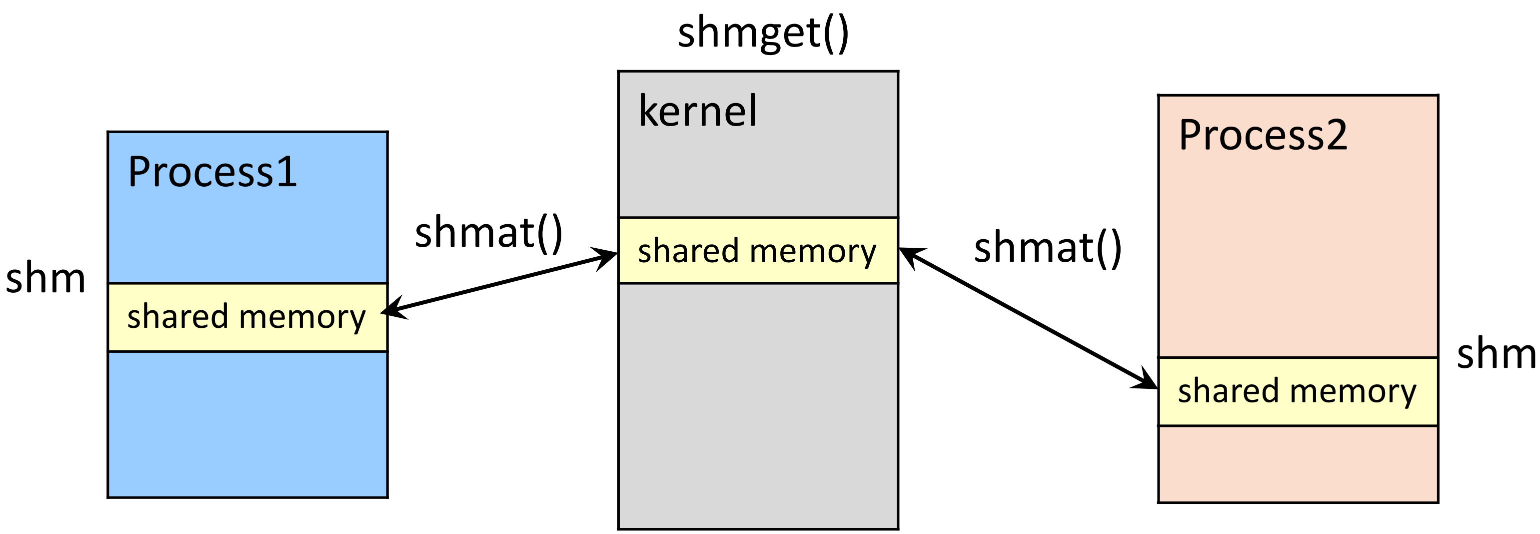
- Detach

```
int shmdt(const void *shmaddr);

if(shmdt(shmptr) == -1)
    perror("shmdt()");
```

Linux: Shared Memory

- Release

```
int shmctl(int shmid,int cmd,struct shmid_ds *buf);

if (shmctl(shmid, IPC_RMID, 0) == -1)
    perror("shmctl()");
```



Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-0: shmdata.h

```
#define TEXT_SIZE 4*1024  /* = PAGE_SIZE, size of each message */
#define TEXT_NUM 1      /* maximal number of mesages */
    /* total size can not exceed current shmmax,
       or an 'invalid argument' error occurs when shmget */

#define PERM S_IRUSR|S_IWUSR|IPC_CREAT

#define ERR_EXIT(m) \
    do { \
        perror(m); \
        exit(EXIT_FAILURE); \
    } while(0)

/* a demo structure, modified as needed */
struct shared_struct {
    int written; /* flag = 0: buffer writable; others: readable */
    char mtext[TEXT_SIZE]; /* buffer for message reading and writing */
};
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-1: shmcon.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <fcntl.h>
#include “alg.8-0-shmdata.h"
```

```
int main(int argc, char *argv[])
{
    struct stat fileattr;
    key_t key; /* of type int */
    int shmid; /* shared memory ID */
    void *shmptr;
    struct shared_struct *shared; /* structured shm */
    pid_t childpid1, childpid2;
    char pathname[80], key_str[10], cmd_str[80];
    int shmsize, ret;

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    printf("max record number = %d, shm size = %d\n", TEXT_NUM, shmsize);

    if(argc <2) {
        printf("Usage: ./a.out pathname\n");
        return EXIT_FAILURE;
    }
    strcpy(pathname, argv[1]);
    if(stat(pathname, &fileattr) == -1) {
        ret = creat(pathname, O_RDWR);
        if (ret == -1) {
            ERR_EXIT("creat()");
        }
        printf("shared file object created\n");
    }
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-1: shmcon.c (2)

```
    key = ftok(pathname, 0x27); /* 0x27 a pro_id 0x0001 - 0xffff, 8 least bits used */
    if(key == -1) {
        ERR_EXIT("shmcon: ftok()");
    }
    printf("key generated: IPC key = %x\n", key); /* can set any key>0 without ftok()*/

    shmid = shmget((key_t)key, shmsize, 0666|PERM);
    if(shmid == -1) {
        ERR_EXIT("shmcon: shmget()");
    }
    printf("shmcon: shmid = %d\n", shmid);

    shmptr = shmat(shmid, 0, 0); /* returns the virtual base address mapping to the
shared memory, *shmaddr=0 decided by kernel */

    if(shmptr == (void *)-1) {
        ERR_EXIT("shmcon: shmat()");
    }
    printf("shmcon: shared Memory attached at %p\n", shmptr);

    shared = (struct shared_struct *)shmptr;
    shared->written = 0;

    sprintf(cmd_str, "ipcs -m | grep '%d'\n", shmid);
    printf("\n------ Shared Memory Segments ------\n");
    system(cmd_str);
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-1: shmcon.c (3)

```
if(shmdt(shmptr) == -1) {
    ERR_EXIT("shmcon: shmdt()");
}

printf("\n------ Shared Memory Segments ------\n");
system(cmd_str);

sprintf(key_str, "%x", key);
char *argv1[] = {" ", key_str, 0};

childpid1 = vfork();
if(childpid1 < 0) {
    ERR_EXIT("shmcon: 1st vfork()");
}
else if(childpid1 == 0) {
    execv("./alg.8-2-shmread.o", argv1); /* call shm_read with IPC key */
}
else {
    childpid2 = vfork();
    if(childpid2 < 0) {
        ERR_EXIT("shmcon: 2nd vfork()");
    }
    else if (childpid2 == 0) {
        execv("./alg.8-3-shmwrite.o", argv1); /* call shmwrite with IPC key */
    }
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-1: shmcon.c (4)

```
            else {
                wait(&childpid1);
                wait(&childpid2);
                     /* shmid can be removed by any process known the IPC key */
                if (shmctl(shmid, IPC_RMID, 0) == -1) {
                    ERR_EXIT("shmcon: shmctl(IPC_RMID)");
                }
                else {
                    printf("shmcon: shmid = %d removed \n", shmid);
                    printf("\n------ Shared Memory Segments ------\n");
                    system(cmd_str);
                    printf("nothing found ...\n");
                    return EXIT_SUCCESS;
                }
            }
        }
    }
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-2: shmread.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/shm.h>
#include “alg.8-0-shmdata.h"
```

```
int main(int argc, char *argv[])
{
    void *shmptr = NULL;
    struct shared_struct *shared;
    int shmid;
    key_t key;

    sscanf(argv[1], "%x", &key);
    printf("%*sshmread: IPC key = %x\n", 30, " ", key);

    shmid = shmget((key_t)key, TEXT_NUM*sizeof(struct shared_struct), 0666|PERM);
    if (shmid == -1) {
        ERR_EXIT("shread: shmget()");
    }

    shmptr = shmat(shmid, 0, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("shread: shmat()");
    }
    printf("%*sshmread: shmid = %d\n", 30, " ", shmid);
    printf("%*sshmread: shared memory attached at %p\n", 30, " ", shmptr);
    printf("%*sshmread process ready ...\n", 30, " ");

    shared = (struct shared_struct *)shmptr;
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-2: shmread.c (2)

```
    while (1) {
        while (shared->written == 0) {
            sleep(1); /* message not ready, waiting ... */
        }
        printf("%*sYou wrote: %s\n", 30, " ", shared->mtext);
        shared->written = 0;
        if (strncmp(shared->mtext, "end", 3) == 0) {
            break;
        }
    } /* it is not reliable to use shared->written for process synchronization */

   if (shmdt(shmptr) == -1) {
        ERR_EXIT("shmread: shmdt()");
   }

    sleep(1);
    exit(EXIT_SUCCESS);
}
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-3: shmwrite.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/shm.h>
#include “alg.8-0-shmdata.h"
```

```
int main(int argc, char *argv[])
{
    void *shmptr = NULL;
    struct shared_struct *shared = NULL;
    int shmid;
    key_t key;

    char buffer[BUFSIZ + 1]; /* 8192bytes, saved from stdin */

    sscanf(argv[1], "%x", &key);
    printf("shmwrite: IPC key = %x\n", key);

    shmid = shmget((key_t)key, TEXT_NUM*sizeof(struct shared_struct), 0666|PERM);
    if (shmid == -1) {
        ERR_EXIT("shmwite: shmget()");
    }

    shmptr = shmat(shmid, 0, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("shmwrite: shmat()");
    }
    printf("shmwrite: shmid = %d\n", shmid);
    printf("shmwrite: shared memory attached at %p\n", shmptr);
    printf("shmwrite precess ready ...\n");

    shared = (struct shared_struct *)shmptr;
```

Linux: Shared Memory

- Single-writer-single-reader problem illustrating shared-memory
  - Algorithm 8-3: shmwrite.c (2)

```
while (1) {
    while (shared->written == 1) {
        sleep(1); /* message not read yet, waiting ... */
    }

    printf("Enter some text: ");
    fgets(buffer, BUFSIZ, stdin);
    strncpy(shared->mtext, buffer, TEXT_SIZE);
    printf("shared buffer: %s\n",shared->mtext);
    shared->written = 1;  /* message prepared */

    if(strncmp(buffer, "end", 3) == 0) {
        break;
    }
}
   /* detach the shared memory */
if(shmdt(shmptr) == -1) {
    ERR_EXIT("shmwrite: shmdt()");
}

sleep(1);
exit(EXIT_SUCCESS);
}
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-2-shmread.o alg.8-2-shmread.c
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out
max record number = 1, shm size = 4100
key generated: IPC key = 27011c6c
shmcon: shmid = 32768
shmcon: shared Memory attached at 0x7fdf086c6000

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       1

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       0
                              shmread: IPC key = 27011c6c
                              shmread: shmid = 32768
                              shmread: shared memory attached at 0x7f4f1dfa3000
                              shmread process ready ...
shmwrite: IPC key = 27011c6c
shmwrite: shmid = 32768
shmwrite: shared memory attached at 0x7fcaca8ff000
shmwrite precess ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmread: shmid = 32768 removed

------ Shared Memory Segments ------
nothing found ...
isscgy@ubuntu:/mnt/os-2020$
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-2-shmread.o alg.8-2-shmread.c
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out
max record number = 1, shm size = 4100
key generated: IPC key = 27011c6c
shmcon: shmid = 32768
shmcon: shared Memory attached at 0x7fdf086c6000

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       1

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       0
                              shmread: IPC key = 27011c6c
                              shmread: shmid = 32768
                              shmread: shared memory attached at 0x7f4f1dfa3000
                              shmread process ready ...
shmwrite: IPC key = 27011c6c
shmwrite: shmid = 32768
shmwrite: shared memory attached at 0x7fcaca8ff000
shmwrite precess ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmread: shmid = 32768 removed

------ Shared Memory Segments ------
nothing found ...
isscgy@ubuntu:/mnt/os-2020$ 
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-2-shmread.o alg.8-2-shmread.c
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out
max record number = 1, shm size = 4100
key generated: IPC key = 27011c6c
shmcon: shmid = 32768
shmcon: shared Memory attached at 0x7fdf086c6000

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       1

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       0
                              shmread: IPC key = 27011c6c
                              shmread: shmid = 32768
                              shmread: shared memory attached at 0x7f4f1dfa3000
                              shmread process ready ...
shmwrite: IPC key = 27011c6c
shmwrite: shmid = 32768
shmwrite: shared memory attached at 0x7fcaca8ff000
shmwrite precess ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmread: shmid = 32768 removed

------ Shared Memory Segments ------
nothing found ...
isscgy@ubuntu:/mnt/os-2020$
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-2-shmread.o alg.8-2-shmread.c
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out
max record number = 1, shm size = 4100
key generated: IPC key = 27011c6c
shmcon: shmid = 32768
shmcon: shared Memory attached at 0x7fdf086c6000

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       1

------ Shared Memory Segments ------
0x27011c6c 32768      isscgy     666        4100       0
                              shmread: IPC key = 27011c6c
                              shmread: shmid = 32768
                              shmread: shared memory attached at 0x7f4f1dfa3000
                              shmread process ready ...
shmwrite: IPC key = 27011c6c
shmwrite: shmid = 32768
shmwrite: shared memory attached at 0x7fcaca8ff000
shmwrite precess ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmread: shmid = 32768 removed

------ Shared Memory Segments ------
nothing found ...
isscgy@ubuntu:/mnt/os-2020$
```

POSIX Shared Memory

- Several IPC mechanisms are available for POSIX systems, including shared memory and message passing.
- POSIX shared memory is organized using *memory-mapped files*, which associate the region of shared memory with a file in /dev/shm/.
- For memory sharing, a process must first create a shared-memory object using the shm_open() system call:

```
int shm_open(const char *path, int flags, mode_t mode);
```

  - Example.

```
fd = shm_open(name, O_CREAT|O_RDWR, 0666);
```

  - *path*: the name of the shared-memory object. Processes that wish to access this shared memory must refer to the object by this name.
  - *flags*: the shared-memory object is to be created if it does not yet exist (O_CREAT) and that the object is open for reading and writing (O_RDWR).
  - *mode*: the file-access permissions of the shared-memory object.
  - A successful call to shm_open() returns an integer file descriptor for the shared-memory object.

POSIX Shared Memory

- Once the object is established, the ftruncate() function is used to configure the size of the object in bytes.

```
int ftruncate(int fd, off_t length);
```

  - E.g., the following call sets the size of the object to 4,096 bytes.

```
ftruncate(fd, 4096);
```

- Finally, the mmap() function establishes a memory-mapped file containing the shared-memory object. It also returns a pointer to the memory-mapped file that is used for accessing the shared-memory object.

```
void *mmap(void *addr, size_t len, int prot, int flags, int
fd, off_t offset);
```

- The API is supported by Linux 2.4 and later, FreeBSD, …
- Compiling

```
gcc -lrt filename.c
```

POSIX Shared Memory

- Producer-Consumer problem illustrating POSIX shared-memory API.
  - Algorithm 8-4: shmpthreadcon.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "alg.8-0-shmdata.h"
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
    char pathname[80], cmd_str[80];
    struct stat fileattr;
    int fd, shmsize, ret;
    pid_t childpid1, childpid2;

    if(argc < 2) {
        printf("Usage: ./a.out filename\n");
        return EXIT_FAILURE;
    }

    fd = shm_open(argv[1], O_CREAT|O_RDWR, 0666);
        /* /dev/shm/filename as the shared object, creating if not exist */
    if(fd == -1) {
        ERR_EXIT(“con: shm_open()");
    }
    system("ls -l /dev/shm/");

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    ret = ftruncate(fd, shmsize);
    if(ret == -1) {
        ERR_EXIT(“con: ftruncate()");
    }
```

POSIX Shared Memory

- Producer-Consumer problem illustrating POSIX shared-memory API.
  - Algorithm 8-4: shmpthreadcon.c (2)

```
    char *argv1[] = {" ", argv[1], 0};
    childpid1 = vfork();
    if(childpid1 < 0) {
        ERR_EXIT("shmpthreadcon: 1st vfork()");
    }
    else if(childpid1 == 0) {
        execv("./alg.8-5-shmproducer.o", argv1); /* call producer with filename */
    }
    else {
        childpid2 = vfork();
        if(childpid2 < 0)
            ERR_EXIT("shmpthreadcon: 2nd vfork()");
        else if (childpid2 == 0)
            execv("./alg.8-6-shmconsumer.o", argv1); /* call consumer with filename */
        else {
            wait(&childpid1);
            wait(&childpid2);
            ret = shm_unlink(argv[1]);
            if(ret == -1) {
                ERR_EXIT("con: shm_unlink()");
            } /* shared object can be removed by any process knew the filename */
            system("ls -l /dev/shm/");
        }
    }
    exit(EXIT_SUCCESS);
}
```

POSIX Shared Memory

- Producer-Consumer problem illustrating POSIX shared-memory API.
  - Algorithm 8-5: shmproducer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "alg.8-0-shmdata.h"
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
    int fd, shmsize, ret;
    void *shmptr;
    const char *message_0 = "Hello World!";

    fd = shm_open(argv[1], O_RDWR, 0666); /* /dev/shm/filename as the shared object */
    if(fd == -1) {
        ERR_EXIT("producer: shm_open()");
    }

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    shmptr = (char *)mmap(0, shmsize, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("producer: mmap()");
    }

    sprintf(shmptr,"%s",message_0);
    printf("produced message: %s\n", (char *)shmptr);

    return EXIT_SUCCESS;
}
```

POSIX Shared Memory

- Producer-Consumer problem illustrating POSIX shared-memory API.
  - Algorithm 8-6: shmconsumer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "alg.8-0-shmdata.h"
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
    int fd, shmsize, ret;
    void *shmptr;

    fd = shm_open(argv[1], O_RDONLY, 0444);
    if(fd == -1) {
        ERR_EXIT("consumer: shm_open()");
    }

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    shmptr = (char *)mmap(0, shmsize, PROT_READ, MAP_SHARED, fd, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("consumer: mmap()");
    }

    printf("consumed message: %s\n", (char *)shmptr);
    return EXIT_SUCCESS;
}
```

POSIX Shared Memory

- Producer-Consumer problem illustrating POSIX shared-memory API.
  - Algorithm 8-6: shmconsumer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-4-shmpthreadcon.c -lrt
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-5-shmproducer.o alg.8-5-shmproducer.c -lrt
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-6-shmconsumer.o alg.8-6-shmconsumer.c -lrt
isscgy@ubuntu:/mnt/os-2020$ ./a.out myshm
total 0
-rw-r--r-- 1 isscgy isscgy 0 Mar 21 21:50 myshm
produced message: Hello World!
consumed message: Hello World!
total 0
isscgy@ubuntu:/mnt/os-2020$
```

```
        ERR_EXIT("mmap()");
    }

    printf("consumed message: %s\n", (char *)shmptr);
    return EXIT_SUCCESS;
}
```