Inter-Process Communication

경희대학교 컴퓨터공학과

조 진 성

Inter-Process Communication

Mechanisms for processes to communicate with each other

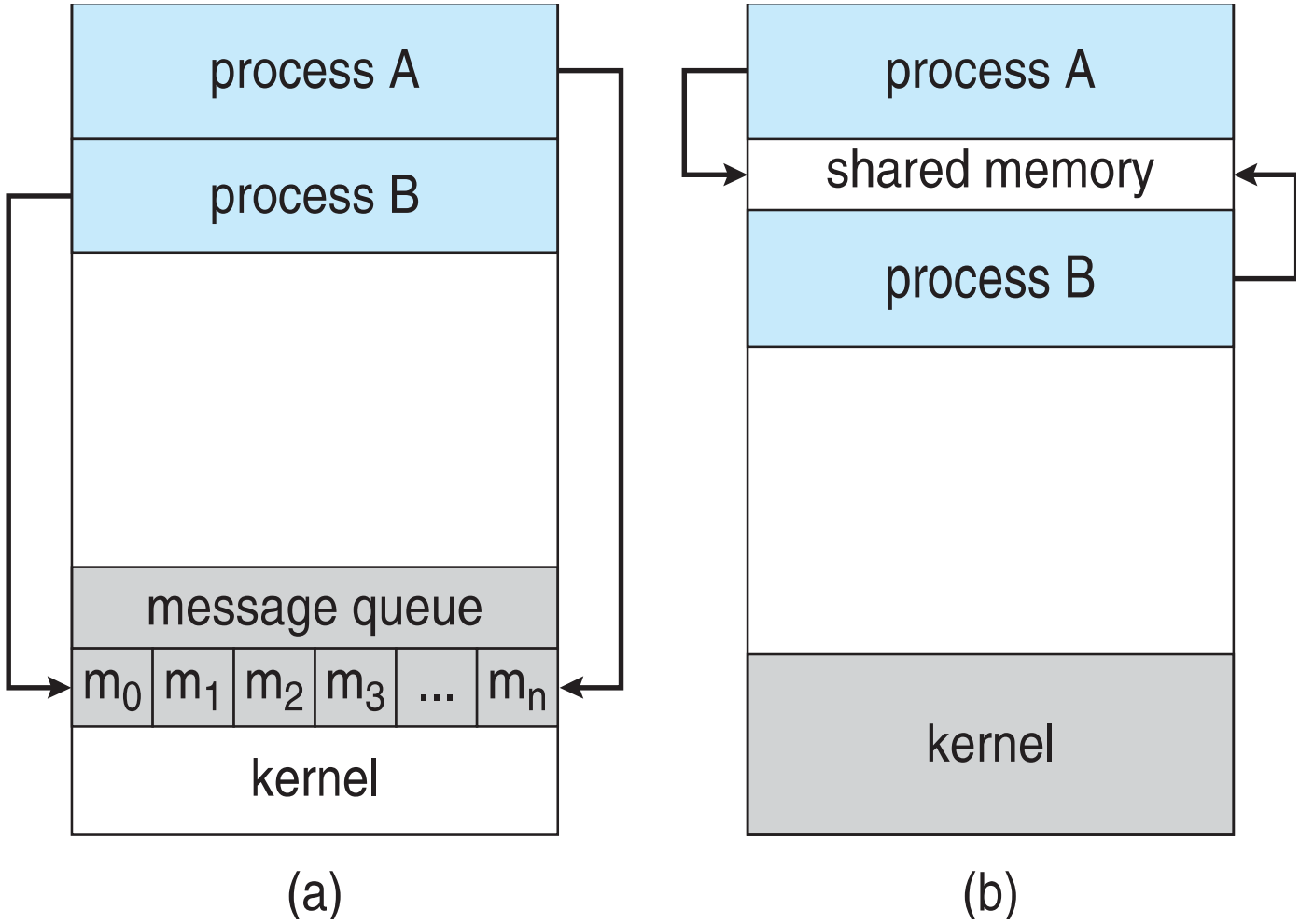
Linux/Unix IPC

- ✓ pipes
- ✓ FIFOs
- ✓ message queue
- ✓ shared memory
- ✓ sockets

Inter-Process Communication (IPC)

Communication models

- ✓ (a) message passing vs. (b) shared memory



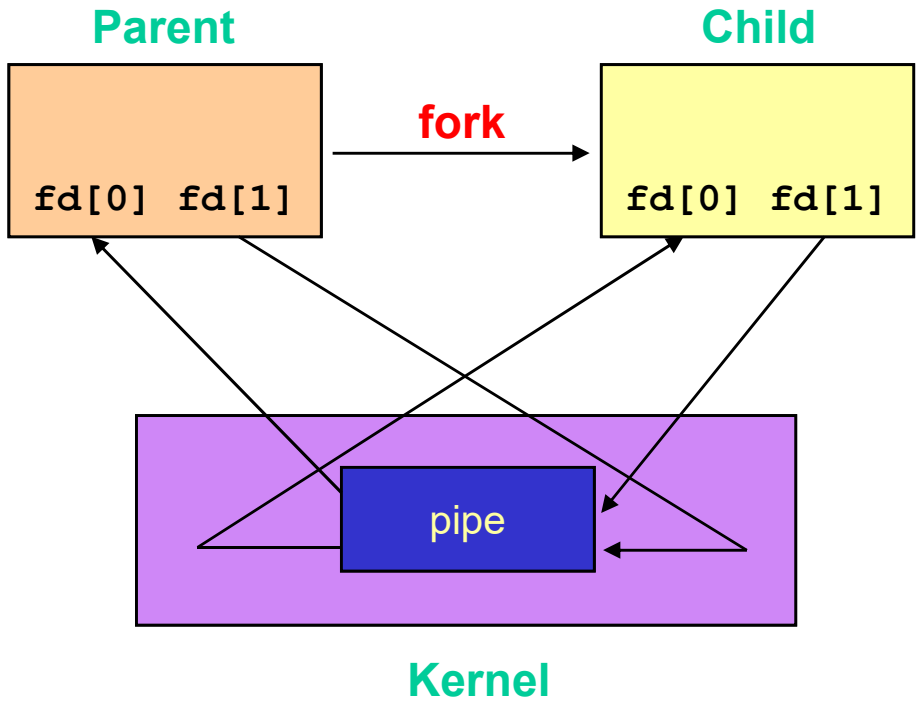
Pipes

The oldest form of UNIX IPC

- ✓ half-duplex
- ✓ between processes that have a common ancestor

pipe

- ✓ `#include <unistd.h>`
- ✓ `int pipe(int fd[2]);`
- ✓ return: 0 if OK, –1 on error



Half-duplex pipe after a **`fork`**

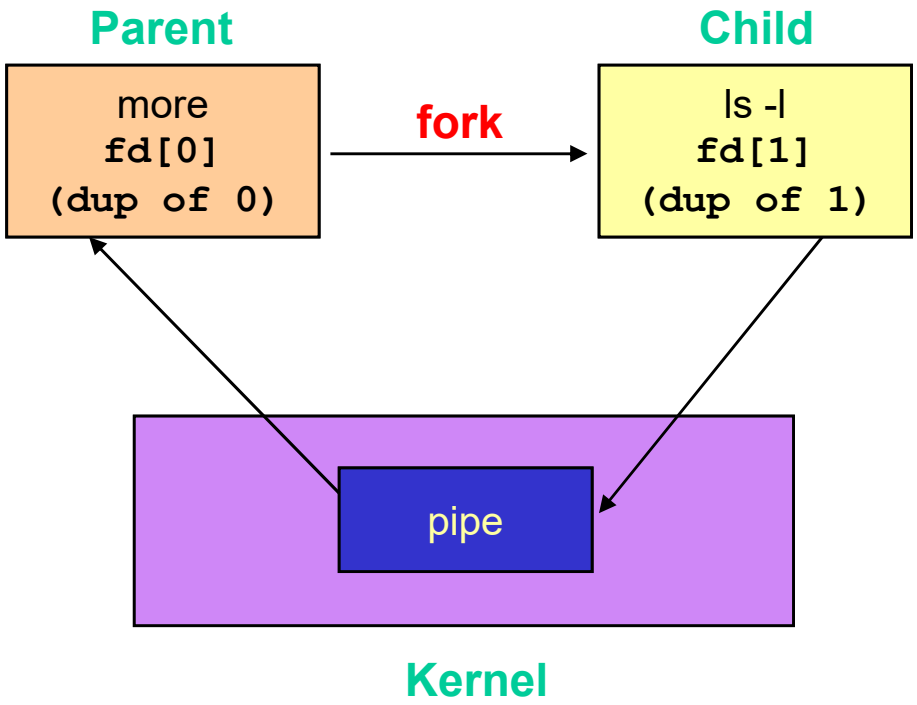
IPC through pipes

- ✓ Once we have created a pipe using **`pipe`**,
- ✓ we can use the normal file I/O functions (e.g. **`read, write`**)

Dup

Create a copy of a given file descriptor

- ✓ `#include <unistd.h>`
- ✓ `int dup(int oldfd);`
- ✓ `int dup2(int oldfd, newfd);`
- ✓ return: newfd if OK, –1 on error



Exercise

Send data from parent to child over a pipe

```
$ gcc -o pipe pipe.c (or make pipe)
$ ./pipe
```

Synchronization between parent and child using pipe

```
$ gcc -o sync sync.c synclib.c (or make sync)
$ ./sync
```

Make my own '`ls -l | more`' program using **`pipe & dup`** system call

```
$ gcc -o mymore mymore.c mymore.c (or make mymore)
$ ./mymore
```


FIFOs

Named pipes

- ✓ full-duplex
- ✓ between unrelated processes that don't have a common ancestor

Create a FIFO

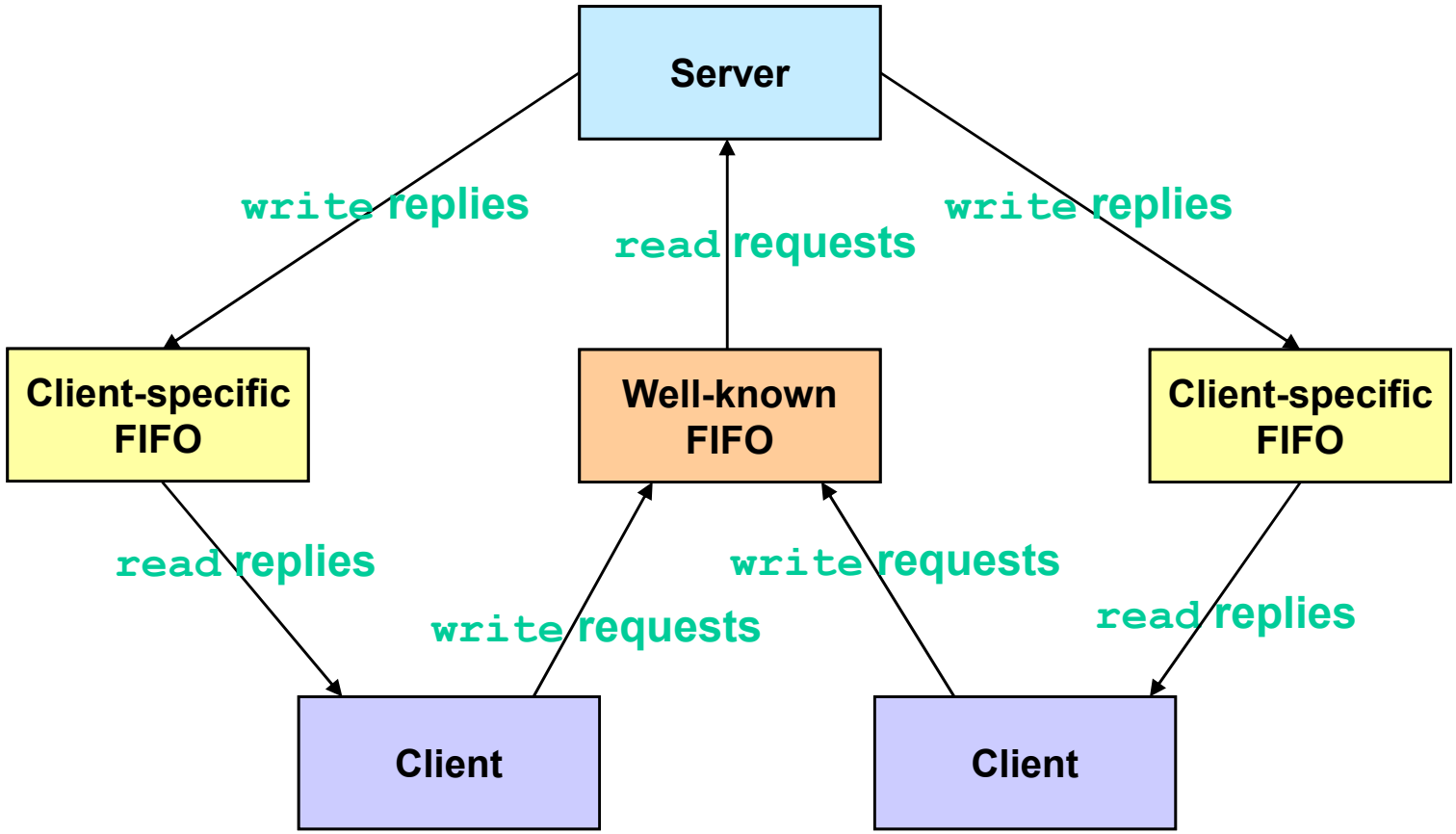
- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/stat.h>`
- ✓ `int mkfifo(char *pathname, mode_t mode);`
- ✓ return: 0 if OK, –1 on error
- ✓ Cf) **`mkfifo`** command

IPC through FIFOs

- ✓ Once we have created a FIFO using **`mkfifo`**,
- ✓ we can use the normal file I/O functions (e.g. **`open, read, write, close`**)

FIFOs (Cont'd)

Client-server communication using FIFOs



Exercise

Client-server communication using FIFOs

```
$ gcc -o fifos fifos.c (or make fifos)
$ gcc -o fifoc fifoc.c (or make fifoc)
$ ./fifos

$ ./fifoc
```


Message Queues

A linked list of messages stored within the kernel

A message consists of

- ✓ a long integer that have the positive integer message type
- ✓ message data

```
struct mymsg  {
    long mtype;        /* positive message type */
    char mtext[512];   /* message data */
};
```

IPC through message queues

- ✓ **msgget**
  - Open an existing queue or create a new one
- ✓ **msgsnd**
  - Place a message onto the queue
- ✓ **msgrcv**
  - Fetch a message from the queue

System Calls for Message Queues

Obtain a message queue ID

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/msg.h>`
- ✓ `int msgget(key_t key, int flag);`
- ✓ return: message queue ID if OK, –1 on error

Message queue control operations

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/msg.h>`
- ✓ `int msgctl(int msqid, int cmd, struct msqid_ds *buf);`
- ✓ return: 0 if OK, –1 on error

System Calls for Message Queues (Cont'd)

Message queue control operations (Cont'd)

- ✓ The second argument, **cmd**
  - **IPC_STAT** : fetch the **msqid_ds** structure for this queue
  - **IPC_SET** : set the part of **msqid_ds** structure
  - **IPC_RMID** : remove the message queue from the system
- ✓ The third argument, **buf**

```
struct msqid_ds  {
  struct ipc_perm  msg_perm;   /* IPC structure: permission and owner */
  struct msg      *msg_first;  /* ptr to fist message on queue */
  struct msg      *msg_last;   /* ptr to last message on queue */
  ulong            msg_cbytes; /* current # of bytes on queue */
  ulong            msg_qnum;   /* # of messages on queue */
  ulong            msg_qbytes; /* max. # of bytes on queue */
  pid_t            msg_lspid;  /* pid of last msgsnd() */
  pid_t            msg_lrpid;  /* pid of last msgrcv() */
  time_t           msg_stime;  /* last-msgsnd() time */
  time_t           msg_rtime;  /* last-msgrcv() time */
  time_t           msg_ctime;  /* last-change time */
};
```


System Calls for Message Queues (Cont'd)

Send a message

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/msg.h>`
- ✓ `int msgsnd(int msqid, void *ptr, size_t nbytes, int flag);`
- ✓ return: 0 if OK, –1 on error
- ✓ The fourth argument, **`flag`**
  - **`IPC_NOWAIT`**

Receive a message

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/msg.h>`
- ✓
```
int msgrcv(int msqid, void *ptr, size_t nbytes, long type,
           int flag);
```
- ✓ return: size of data portion of message if OK, –1 on error
- ✓ The fifth argument, **`flag`**
  - **`IPC_NOWAIT`**

System Calls for Message Queues (Cont'd)

Receive a message (Cont'd)

- ✓ if **`type == 0`**,
  - the first message on the queue is returned
- ✓ if **`type > 0`**,
  - the first message on the queue whose message type equals **`type`** is returned
- ✓ if **`type < 0`**,
  - the first message on the queue whose message type is the lowest value less than or equal to the absolute value of **`type`** is returned
- ✓ if **`IPC_NOWAIT`** is specified in **`flag`**,
  - return is made with an error of **`EAGAIN`**

Exercise

IPC between two processes using message queue

```
$ gcc -o msgq1 msgq1.c (or make msgq1)
$ gcc -o msgq2 msgq2.c (or make msgq2)
$ ./msgq1

$ ./msgq2
```

Note:

- ✓ If a process creates a message queue, its data structure remains in the kernel even though the process has terminated
- ✓ You have to remove it through **`msgctl()`** with **`IPC_RMID`** parameter

  (Or, reboot the system !!!)

Shared Memory

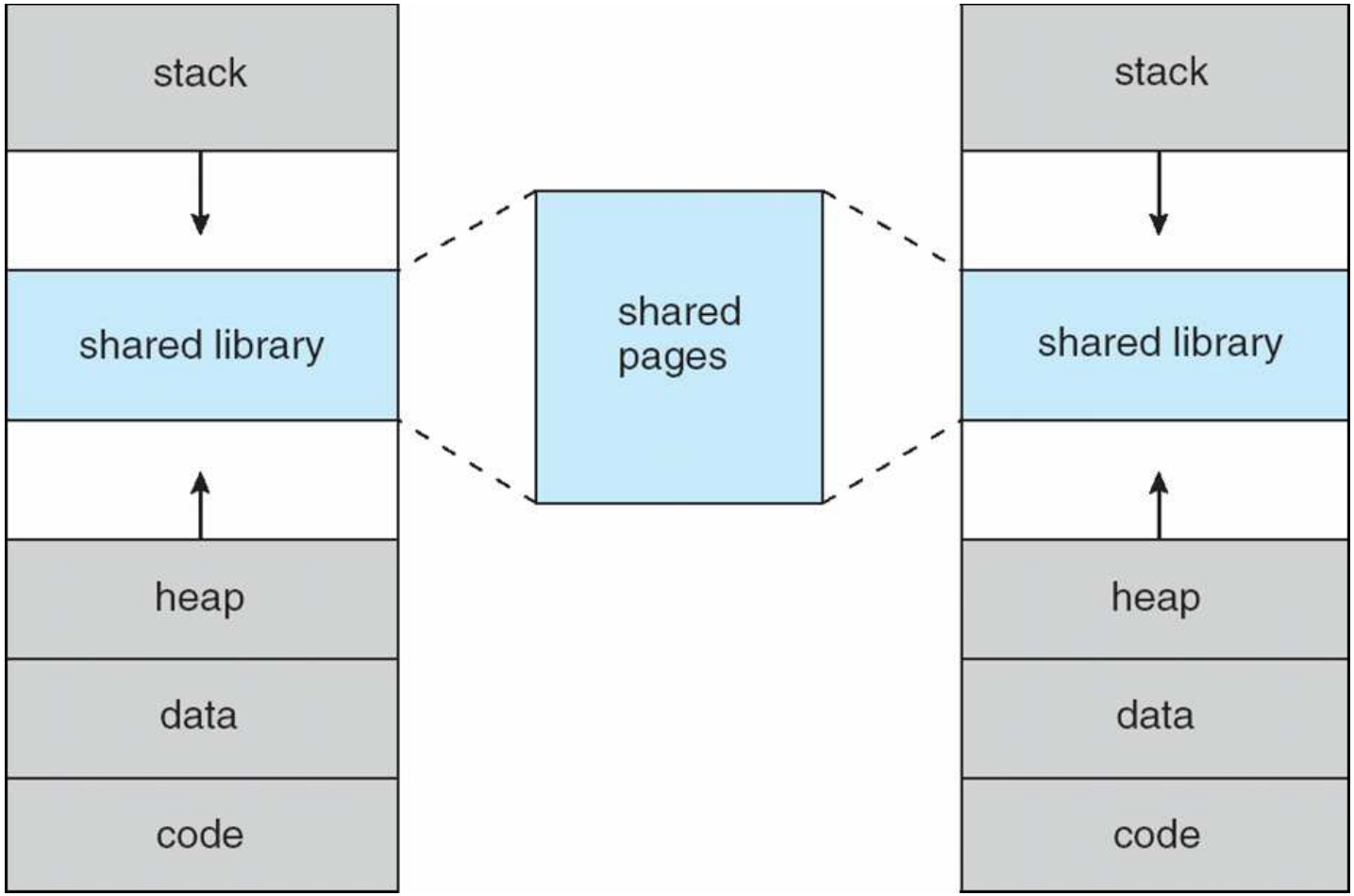
Allows two or more processes to share a given region of memory

- ✓ The fastest form of IPC because data does not need to be copied between them
- ✓ Need to synchronize shared memory access
  - Semaphores are used often

IPC through shared memory

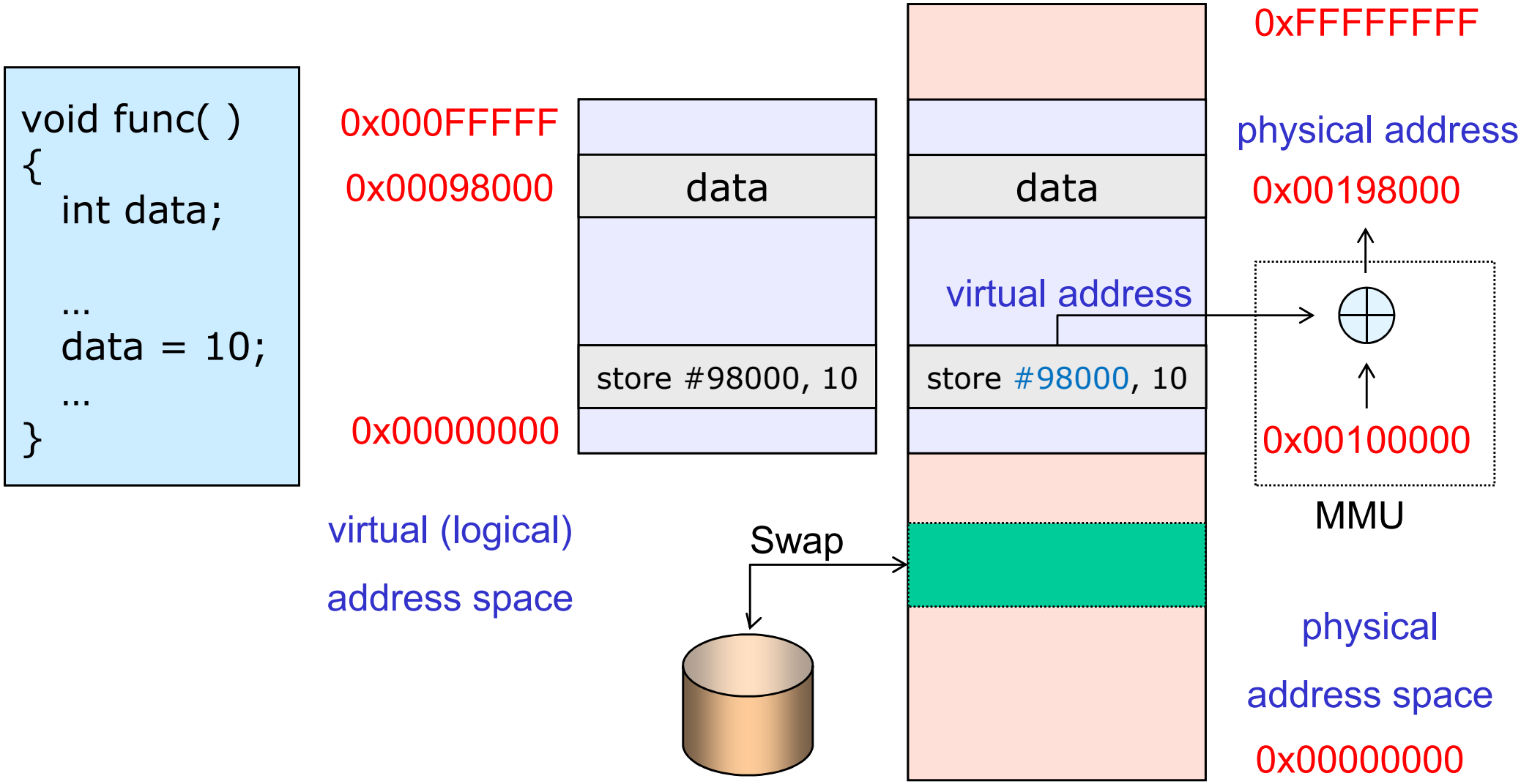
- ✓ **`shmget`**
  - Obtain a shared memory identifier
  - Open an existing segment or create a new one
- ✓ **`shmat`**
  - Attach a shared memory segment to the process' address space
- ✓ **`shmdt`**
  - Detach a shared memory segment
  - **`shmdt`** does not remove the identifier and its associated data structure from the system

Shared Pages Example



Virtual Memory Management

Address mapping



System Calls for Shared Memory

Obtain a shared memory ID

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/shm.h>`
- ✓ `int shmget(key_t key, int size, int flag);`
- ✓ return: shared memory ID if OK, –1 on error
- ✓ The second argument, **`size`**
  - if a new segment is being created, we must specify its minimum size, **`size`**
  - if we are referencing an existing segment, we can specify **`size`** as 0

Shared memory control operations

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/msg.h>`
- ✓ `int shmctl(int shmid, int cmd, struct shmid_ds *buf);`
- ✓ return: 0 if OK, –1 on error

System Calls for Shared Memory (Cont'd)

Shared memory control operations (Cont'd)

- ✓ The second argument, **cmd**
  - **IPC_STAT** : fetch the **shmid_ds** structure for this shared memory
  - **IPC_SET** : set the part of **shmid_ds** structure
  - **IPC_RMID** : remove the shared memory segment from the system
  - **SHM_LOCK** : lock the shared memory in memory
  - **SHM_UNLOCK** : unlock the shared memory segment
- ✓ The third argument, **buf**

```
struct shmid_ds  {
  struct ipc_perm  shm_perm;    /* IPC structure: permission and owner */
  struct anon_map *shm_amp;     /* pointer in kernel */
  int              shm_segsz;   /* size of segment in bytes */
  ushort           shm_lkcnt;   /* # of times segment is being locked */
  pid_t            shm_lpid;    /* pid of last shmop() */
  pid_t            shm_cpid;    /* pid of creator */
  ulong            shm_nattch;  /* # of current attaches */
  ulong            shm_cnattch; /* used only for shminfo */
  time_t           shm_atime;   /* last-attach time */
  time_t           shm_dtime;   /* last-detach time */
  time_t           shm_ctime;   /* last-change time */
};
```


System Calls for Shared Memory (Cont'd)

Attach a shared memory segment

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/shm.h>`
- ✓ `void *shmat(int shmid, void *addr, int flag);`
- ✓ return: pointer to shared memory segment if OK, –1 on error
- ✓ The third argument, **`flag`**
  - **`SHM_RND`, `SHM_RDONLY`**
- ✓ if **`addr == 0`**, (Recommended)
  - the segment is attached at the first available address selected by the kernel
- ✓ if **`addr != 0`** and **`SHM_RND`** is not specified,
  - the segment is attached at the address given by **`addr`**
- ✓ if **`addr != 0`**, and **`SHM_RND`** is specified,
  - the segment is attached at the address given by (**`addr`**-(**`addr`** modulus **`SHMLBA`**))
- ✓ if **`SHM_RDONLY`** is specified,
  - the segment is attached read-only

System Calls for Shared Memory (Cont'd)

Detach a shared memory segment

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/ipc.h>`
- ✓ `#include <sys/shm.h>`
- ✓ `int shmdt(void *addr);`
- ✓ return: 0 if OK, –1 on error

Exercise

Shared memory example (& memory map)

```
$ gcc -o shm shm.c (or make shm)
$ ./shm
```

IPC between two processes using shared memory

```
$ gcc -o sipc1 sipc1.c (or make sipc1)
$ gcc -o sipc2 sipc2.c (or make sipc2)
$ ./sipc1

$ ./sipc2
```

Note:

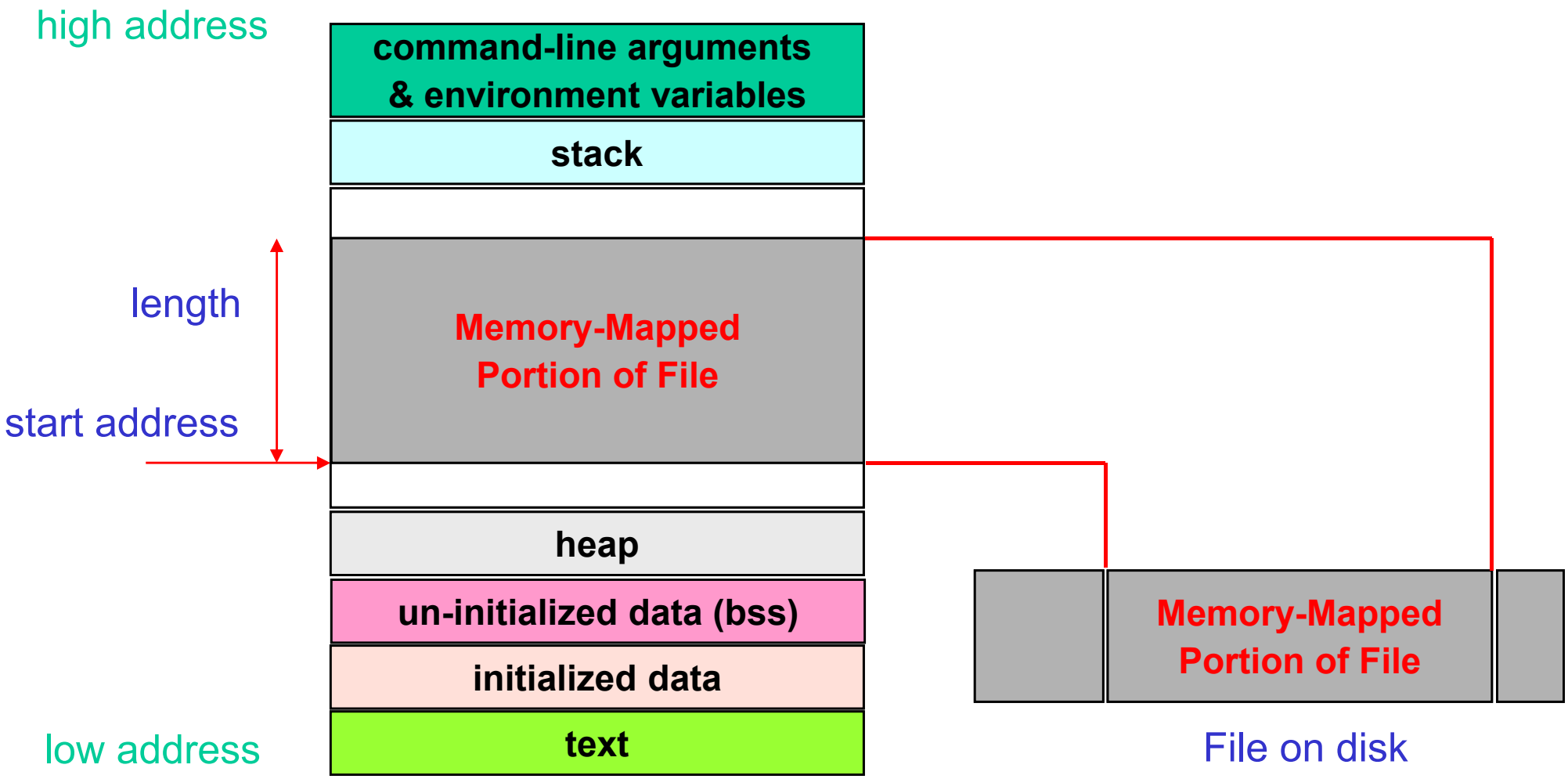
- ✓ If a process creates a shared memory, its data structure remains in the kernel even though the process has terminated
- ✓ You have to remove it through **`shmctl()`** with **`IPC_RMID`** parameter

  (Or, reboot the system !!!)

Memory-Mapped File

Map a file on disk into a buffer in memory

- ✓ perform I/O without using **`read`** or **`write`**



System Calls for Memory-Mapped File

Map pages of memory

- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/mman.h>`
- ✓
```
caddr_t mmap(caddr_t addr, size_t len, int prot, int flag,
             int fd, off_t off);
```
- ✓ return: starting address of mapped region if OK, –1 on error
- ✓ The first argument, **`addr`**
  - 0 (recommended) : system choose the starting address
  - can be a specific value
- ✓ The third argument, **`prot`**
  - **`PROT_READ`** : region can be read
  - **`PROT_WRITE`** : region can be written
  - **`PROT_EXEC`** : region can be executed
  - **`PROT_NONE`** : region cannot be accessed
- ✓ The fourth argument, **`flag`**
  - **`MAP_FIXED`** : return value must equal addr
  - **`MAP_SHARED`** : store operations modify the mapped file
  - **`MAP_PRIVATE`** : store operations modify a copy of mapped file

System Calls for Memory-Mapped File (Cont'd)

Unmap a memory-mapped region

- ✓ Automatically unmapped when the process terminates, or
- ✓ `#include <sys/types.h>`
- ✓ `#include <sys/mman.h>`
- ✓ `int munmap(caddr_t addr, size_t len);`
- ✓ return: 0 if OK, –1 on error

Exercise

Make my own `cp` program using memory-mapped file

```
$ gcc -o mycp3 mycp3.c (or make mycp3)
$ ./mycp3 mycp3.c mycp3.bak
$ ls -l mycp3.c mycp3.bak
```

IPC between parent and child using memory mapped file of /dev/zero

```
$ gcc -o mipc mipc.c synclib.c (or make mipc)
$ ./mipc
```


Summary

Inter-Process Communication (IPC)

- ✓ Mechanisms for processes to communicate with each other

System calls in Linux for IPC

- ✓ Pipe: **`pipe, read, write, close`**
- ✓ FIFO: **`mkfifo, open, read, write, close`**
- ✓ Message queue: **`msgget, msgctl, msgsnd, msgrcv`**
- ✓ Shared memory: **`shmget, shmctl, shmat, shmdt`**
- ✓ Memory-mapped file: **`mmap, munmap`**