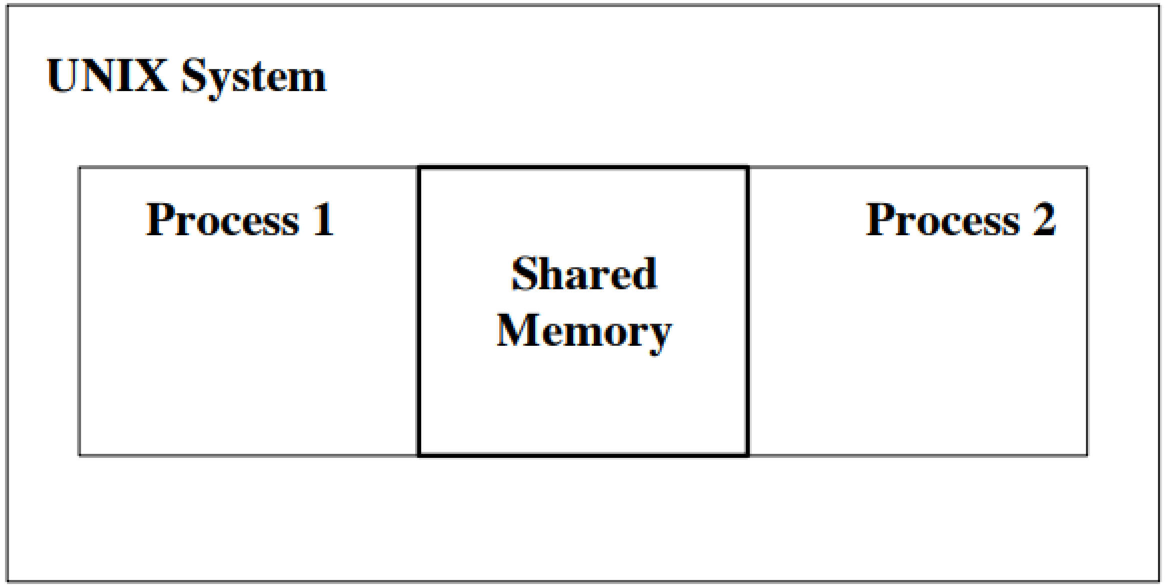
Shared Memory

Shared Memory

- Multiple processes accessing the same memory zone
- The memory is accessed as part of the memory zone data of the process
- Advantage is speed; access is instant (no message transfer)



Shared Memory

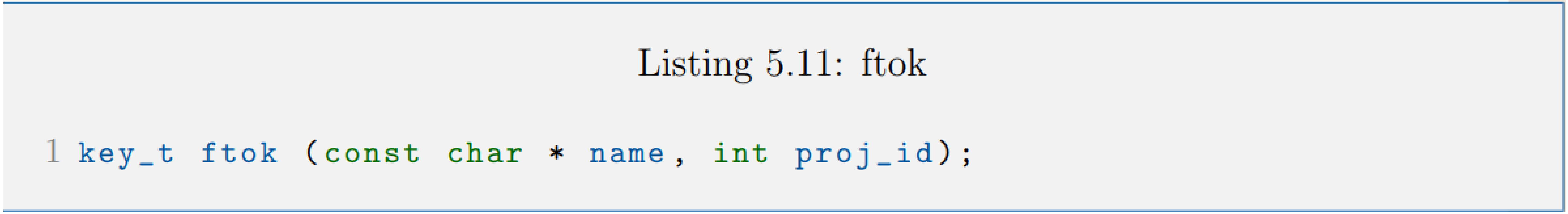
- Operations used in shared memory are:
- Creating a Zone.
- Attaching the process area.
- Access to data.
- Release of the area

Shared Memory

- Functions used in shared memory are:
- ftok: identifier(key) generation function.
- shmget: creation function.
- shmat: attachment process in the area function.
- shmctl: control of the area (used for release).

Shared Memory: The ftok() function

- The ftok() function in C is used to generate a unique key based on a given file path and a project identifier.
- This key is commonly used with other IPC (Inter-Process Communication) mechanisms
- such as message queues, semaphores, and shared memory segments
- to identify and access resources shared between processes.



Listing 5.11: ftok

```c
key_t ftok (const char * name, int proj_id);
```

Shared Memory: The ftok() function

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

int main() {
  char *pathname = "/tmp/example.txt"; // Pathname
of the file
  int proj_id = 1; // Project identifier

  // Generate a key using ftok()
  key_t key = ftok(pathname, proj_id);
  if (key == -1) {
    perror("ftok");
    return 1;
  }

  printf("Generated key: %d\n", key);

  return 0;
}
```

In this example:

1- We define a pathname variable representing the path to a file.

This file will be used to generate the key.

2- proj_id is an arbitrary integer identifier for the project.

3- We call ftok() with the pathname and proj_id as arguments.

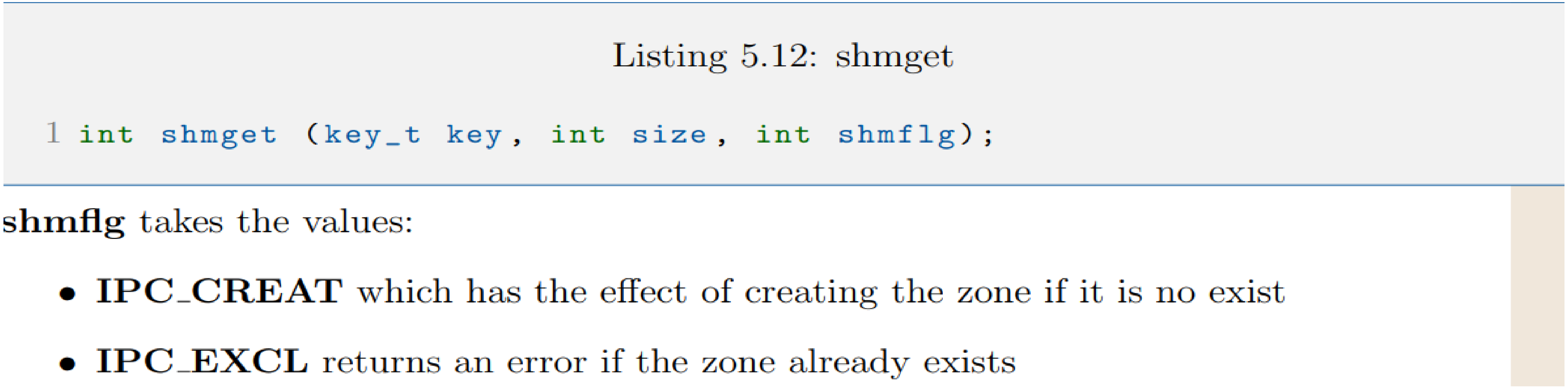
4- It returns a key of type key_t.

The generated key is used to identify a resource shared between processes, such as a message queue, semaphore set, or shared memory segment.

Each resource created with IPC mechanisms like msgget(), semget(), or shmget() is associated with a unique key, allowing processes to access the same resource using that key.

Shared Memory: shmget() function

- The shmget() function in C is used to create or access a shared memory segment.
- Shared memory is a technique used for inter-process communication, where multiple processes can share a common region of memory.
- This shared memory segment can be used to exchange data between processes efficiently.



Listing 5.12: shmget

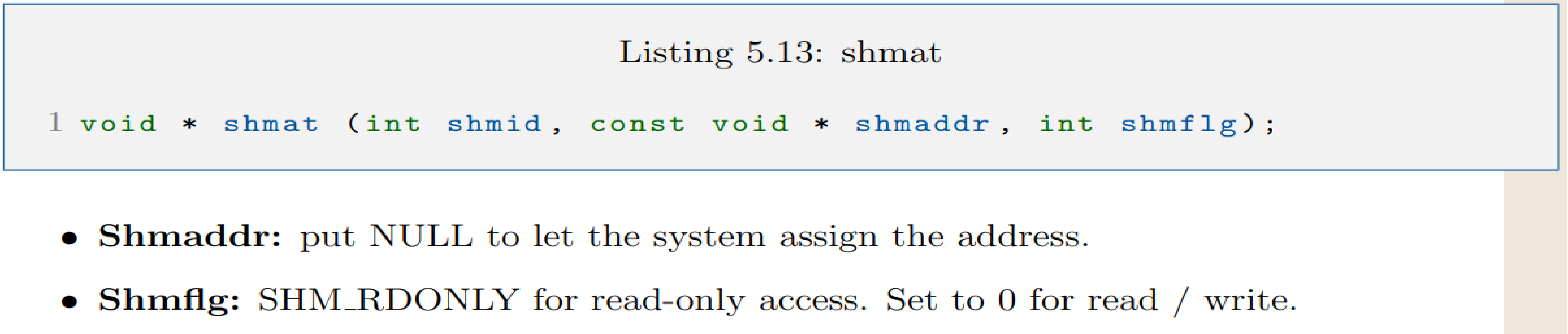
```c
int shmget (key_t key, int size, int shmflg);
```

shmflg takes the values:

- **IPC_CREAT** which has the effect of creating the zone if it is no exist
- **IPC_EXCL** returns an error if the zone already exists

Shared Memory: shmat() function

- The shmat() function in C is used to attach a shared memory segment to the process's address space.
- Once a shared memory segment is attached, the process can access it as if it were a part of its own memory.
- This makes inter-process communication (IPC) via shared memory very efficient as data does not need to be copied between processes.



Listing 5.13: shmat

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

- **Shmaddr:** put NULL to let the system assign the address.
- **Shmflg:** SHM_RDONLY for read-only access. Set to 0 for read / write.

Shared Memory: shmget() function

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

#define SHM_SIZE 1024  // Size of the shared memory
segment

int main() {
    key_t key = 1234;  // Key to identify the shared
memory segment
    int shmid;  // Shared memory ID
    char *shmaddr;  // Pointer to the shared memory
segment

    // Create or access the shared memory segment
    shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }
// Attach the shared memory segment to the process's
address space
    shmaddr = shmat(shmid, NULL, 0);
    if (shmaddr == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    // Write some data to the shared memory
    sprintf(shmaddr, "Hello, shared memory!");

    // Detach the shared memory segment from the
process's address space
    if (shmdt(shmaddr) == -1) {
        perror("shmdt");
        exit(1);
    }

    return 0;
}
```

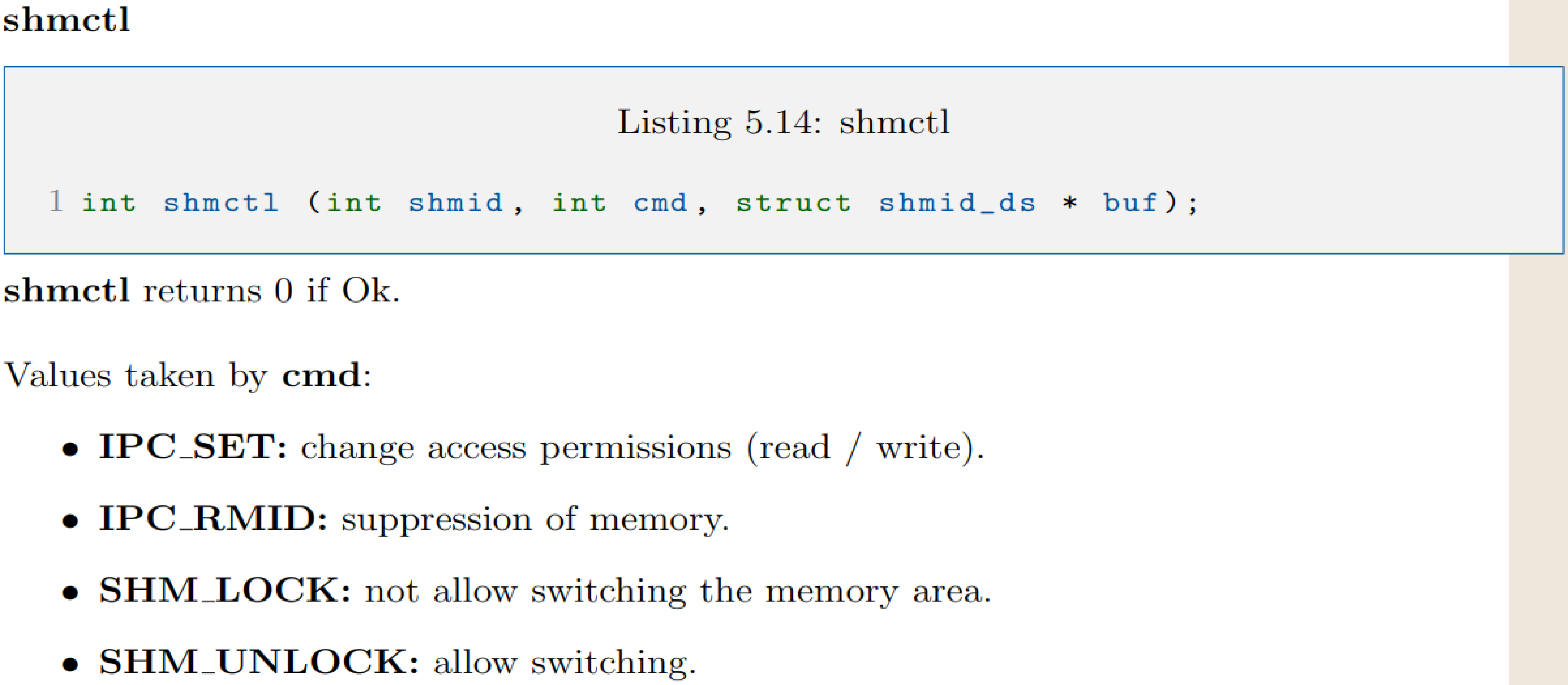
Shared Memory

- shmaddr = shmat(shmid, NULL, 0);
- means "attach the shared memory segment identified by shmid at an address chosen by the system for reading and writing, and store the address of this attached segment in shmaddr."
- Once attached, the processes can use the address pointed to by shmaddr to read from and write to the shared memory segment as if they were accessing regular memory.

Shared Memory: shmctl()

- The shmctl() is used to perform various control operations on a shared memory segment.
- The shmctl() function allows a process to configure and manage shared memory segments based on the command and options provided.
- It's commonly used to change properties of the segment, get information about it, or remove it from the system.

Shared Memory: shmctl()



shmctl

Listing 5.14: shmctl

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

shmctl returns 0 if Ok.

Values taken by **cmd**:

- **IPC_SET:** change access permissions (read / write).
- **IPC_RMID:** suppression of memory.
- **SHM_LOCK:** not allow switching the memory area.
- **SHM_UNLOCK:** allow switching.

Shared Memory: shmget() function

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

int main() {
    key_t key;
    int shmid;

    // Get key
    key = ftok("path_to_some_file", 'R');

    // Create the shared memory segment
    shmid = shmget(key, 1024, 0644 | IPC_CREAT);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }
```

```
 // Perform some operation with the shared memory...

    // Now remove the shared memory segment
    if (shmctl(shmid, IPC_RMID, NULL) == -1) {
        perror("shmctl");
        exit(1);
    }

    printf("Shared memory segment removed\n");
    return 0;
}
```

Shared Memory: shmctl()

- In this example, the shmctl() function is used with the IPC_RMID command to mark the shared memory segment for deletion.
- After this call, new processes will not be able to attach to this segment, and it will be destroyed once all attached processes have detached from it.

Shared Memory: Example

- Develop a C program that simulates the management of a shared resource (e.g., a printer, a computation task scheduler, etc.) among multiple concurrent processes using shared memory.

Shared Memory: Example

Specific Tasks

1- Initialization Program:

- Use ftok to generate a unique key.
- Create a shared memory segment using shmget.
- Attach the shared memory segment using shmat.
- Initialize the resource status as available in the shared memory.
- Detach the memory segment and leave it available for other processes.

2- Client Program:

- Attach to the existing shared memory segment created by the initialization program.
- Check if the resource is available. If not, wait until it becomes available.
- Once available, mark the resource as busy, simulate resource usage (e.g., sleep for a few seconds), and then mark it as free again.
- Finally, detach from the shared memory segment.

3- Cleanup Program:

- Attach to the shared memory segment.
- After ensuring no process needs the resource anymore, use shmctl with IPC_RMID to remove the shared memory segment.

Shared Memory: Example

1. Initialization Program (init.c): This program creates and initializes the shared memory segment.

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

#define SHM_KEY_PATH "resource_key"
#define SHM_KEY_ID 65

int main() {
    key_t key;
    int shmid;
    int *resource;

    // Generate unique key
    key = ftok(SHM_KEY_PATH, SHM_KEY_ID);
    if (key == -1) {
        perror("ftok");
        exit(1);
    }
  // Create the shared memory segment
    shmid = shmget(key, sizeof(int), 0666 | IPC_CREAT);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }
    // Attach the shared memory segment
    resource = (int *)shmat(shmid, NULL, 0);
    if (resource == (void *)-1) {
        perror("shmat");
        exit(1);
    }
    // Initialize the resource as available (0)
    *resource = 0;

    // Detach the shared memory
    if (shmdt(resource) == -1) {
        perror("shmdt");
        exit(1);
    }
    printf("Shared memory segment initialized.\n");
    return 0;
}
```

Shared Memory: Example

2. Client Program (client.c): This program attaches to the shared memory segment, tries to acquire the resource, uses it, and then releases it.

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

#define SHM_KEY_PATH "resource_key"
#define SHM_KEY_ID 65

int main() {
    key_t key;
    int shmid;
    int *resource;

    // Generate unique key
    key = ftok(SHM_KEY_PATH, SHM_KEY_ID);
    if (key == -1) {
        perror("ftok");
        exit(1);
    }
```

```
// Access the shared memory segment
    shmid = shmget(key, sizeof(int), 0666);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Attach the shared memory segment
    resource = (int *)shmat(shmid, NULL, 0);
    if (resource == (void *)-1) {
        perror("shmat");
        exit(1);
    }

    // Try to acquire the resource
    while (__sync_lock_test_and_set(resource, 1)) {
        printf("Resource busy, waiting...\n");
        sleep(1);  // Wait before trying again
    }

    printf("Resource acquired, using...\n");
    sleep(5);  // Simulate resource usage
```

Shared Memory: Example

2. Client Program (client.c): This program attaches to the shared memory segment, tries to acquire the resource, uses it, and then releases it.

```
    // Release the resource
    __sync_lock_release(resource);
    printf("Resource released.\n");

    // Detach from the shared memory
    if (shmdt(resource) == -1) {
        perror("shmdt");
        exit(1);
    }

    return 0;
}
```

Shared Memory: Example

3. Cleanup Program (cleanup.c): This program removes the shared memory segment after all processes are done using it.

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

#define SHM_KEY_PATH "resource_key"
#define SHM_KEY_ID 65

int main() {
    key_t key;
    int shmid;

    // Generate unique key
    key = ftok(SHM_KEY_PATH, SHM_KEY_ID);
    if (key == -1) {
        perror("ftok");
        exit(1);
    }
// Find the shared memory segment
    shmid = shmget(key, sizeof(int), 0666);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Remove the shared memory segment
    if (shmctl(shmid, IPC_RMID, NULL) == -1) {
        perror("shmctl");
        exit(1);
    }

    printf("Shared memory segment removed.\n");
    return 0;
}
```

Shared Memory: Example

1- Run ./init to initialize the shared memory.

2- Run multiple ./client instances concurrently to simulate multiple processes trying to access the resource.

3- Once all client processes have finished, run ./cleanup to clean up the shared memory.