Shared Memory

- Similarities with semaphore management.
- Creating a shared memory segment.
- Removing shared memory segments.
- Attaching shared memory to a process.
- Detaching shared memory from a process.

Good News, Everyone!

As it turns out, the system calls for shared memory and semaphores are remarkably similar. In fact, the format of most of the commands is identical.

Creating / Gaining Resource:

$$\begin{tikzcd} \texttt{shmget(...)} \arrow[r, leftrightarrow, dashed] & \texttt{semget(...)} \end{tikzcd}$$

Modifying Parameters / Removing:

$$\begin{tikzcd} \texttt{shmctl(...)} \arrow[r, leftrightarrow, dashed] & \texttt{semctl(...)} \end{tikzcd}$$

Accessing:

$$\begin{tikzcd} \begin{matrix}\texttt{shmat(...)}\\ \texttt{shmdt(...)}\end{matrix} \arrow[r, leftrightarrow, dashed] & \texttt{semop(...)} \end{tikzcd}$$

Creating Shared Memory

```
int shmget(IPC_PRIVATE, int size, IPC_CREAT | 0666)
```

Returns the <u>key</u> (`shmid`) for the newly created shared memory segment. It does NOT return a pointer to the memory.

The size, in bytes, of the segment that you want to create.

Informs the kernel that you want to create a new, unique shared memory segment.

These are the permissions for the memory segment. These follow the same format as the semaphore permissions.

(Note: This creates the memory segment, but it does not give any way for your process to access the memory. This is achieved through the call to `shmat()`.)

Accessing A Previously-Created Memory Resource

```
int shmget(key_t shmid, 0, 0666)
```

Returns the memory group key `MemGroup`, or -1 on Failure.

This is the key from a previously created memory group. In other words, it is the return value from the creation of a shared memory segment (previous slide).

Since the memory group was already created, there is no reason to specify the number of bytes it contains.

Since we are no longer creating the shared memory segment, we remove the `IPC_CREAT`. This way, the program will throw an error if we specified an invalid key.

Removing Shared Memory

```
int shmctl(key_t shmid, IPC_RMID, (struct shmid_ds *) 0)
```

-1 on Failure

This is the group key for the shared memory segment that you want to remove.

Informs `shmctl` that you want to remove the shared memory segment from the system.

Since this call to shmctl does not modify the parameters of the shared memory segment, we can just pass 0 or a NULL structure as the last argument.

(Note: The shared memory segment cannot actually be removed until it is detached from all processes that are using it.)

Attaching Shared Memory

Once a shared memory segment is created, it must be mapped to a process before it can be accessed. This is accomplished through `shmat()`.

```
void* shmat(int shmid, (void *) 0, 0)
```

Returns a pointer to the shared memory, or -1 on Failure (Notice: it does not return NULL on Failure!!)

This is the memory group key for the shared memory segment you want to attach to.

On rare occasions, it is useful to map the shared memory to a specific address. We will never need this, so just use `(void *) 0` here to let the system choose an address.

Aside from `SHM_RDONLY`, it is unlikely that we will need any of the flags.

Detaching Shared Memory

When a process is finished using shared memory, it should detach it. The man pages on my Gentoo machine state that shared memory is automatically detached when a process exits, but it is still a good idea to detach it manually.

```
int shmat(void *shmaddr)
```

-1 on Failure

This is the address that the shared memory was attached to (i.e, the return value from `shmat()`)

Image by Nick Burger

This is here to take up space