Operating Systems Lab (C+Unix)

Enrico Bini

University of Turin

Outline

IPC shared memory

- System V implements *shared memory*: remember, when allocating by `malloc` you are allocating over the heap (which is private to the process!)
  - If memory is allocated by `malloc` and then a process is forked
  - the two processes **do not share** the allocated memory
  - pointers represent logical addresses (not physical)
- Shared memory is a fast way for processes to communicate: no kernel structure (buffers, queues, etc) mediating the access to the shared memory:
  - this is good: fast way to implement IPC
  - this is bad: high risk of inconsistent behavior if memory is read "in the middle" of a write
- Once a process writes to a shared memory segment, the data written becomes immediately accessible to the other processes accessing the shared memory segment
- Simplicity of usage: assignments are made with the same syntax of private memory: no special function to access, no `write`, `read`, `msgsnd`, `msgrcv`, just "="

Lifecylce of a shared memory segment

1. **Creation** by `shmget(...)`: size of memory must be specified
   - the shared memory segment is allocated over an area accessible to all processes
2. **Attaching** the shared memory area to the process address space
   - after a shared segment is attached to a private address space, it can be normally used by the process
   - any data written to the shared memory segment becomes immediately visible to other processes sharing the same segment
3. **Detaching** the segment from the process address space
   - the shared memory is no longer visible, but it still exists (it is a **persistent** object)
4. **Deallocation** of the shared memory segment

Creating a shared memory segment

- The system call

```
int shmget(key_t key, size_t size, int shmflg)
```

returns the identifier of a shared memory segment associated to `key` of size **at least** `size` (the allocated size is a multiple of `PAGE_SIZE`)

  - `shmflag` is a list of ORed ("|") options including:
    - read/write permissions (least significant 9 bits)
    - `IPC_CREAT`:
      (1) create a new shm segment associated to the `key`, if it doesn't exist
      (2) return the existing shm ID associated to the `key`, if it exists
    - `IPC_EXCL` (used only with `IPC_CREAT`): the call fails (with `errno=EEXIST`) if the shm segment exists

- Shared memory segments are persistent object: they will survive to the process death, they must be erased expicitly

Attaching a shared memory area to a process

- To attach a shared memory segment to the address space of a process

```
void *shmat(int shmid, NULL, int shmflg)
```

  - shmid, ID of the shm object
  - second argument used for advanced features: setting to NULL is safe
  - shmflg, flags
    - SHM_RDONLY, uses the shared memory in read-only mode
    - plus others for advanced settings
- it returns a pointer to the shared memory segment
- Typical usage (malloc-like)

```
struct my_data * datap; /* shared data struct */

shm_id = shmget(IPC_PRIVATE, sizeof(struct my_data), 0600);
datap = shmat(shm_id, NULL, 0);
/* From now on, all processes accessing to
   datap->something, read/write in shared mem */
```

Detaching a shared memory

- A shm segment is detached by

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

- `shmaddr` is the address of the segment we want to detach, previously returned by a `shmat` call
- **Implicit detaching** of a shm segment occurs when:
  1. the process terminates
  2. the control flow passes to another process by `exec()`
- detaching is **not** deallocation

Control (and deallocation) of a shm segment

- A shared memory segment is controlled by

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

  - shmid, the ID of the shared memory object
  - cmd, is the command to be made (IPC_STAT, IPC_RMID, ...)
  - the third argument may be used depending on the command cmd

- To mark a shared memory for deallocation

```
int shmctl(int shmid, IPC_RMID, NULL);
```

  - **Important**: the actual deallocation happens only when the last process is detached from the shared memory segment
  - Deallocating the shm segment immedately would create problems to the processes still using the segment
    - these problems cannot be detected by some errno (as for message queues), because the access to memory segment is made by assignments "=", not by any function calls

Example on shared memory

- Many child processes filling a shared table
- Each process needs to get a unique entry in the table, then it can write without conflict
  - *Makefile*
  - *test-shm.h*
  - *test-shm-parent.c*
  - *test-shm-child.c*

Shared memory: POSIX APIs

- For historical reasons, the course follows the System V API
- However, today the POSIX standard is dominant
- Here is a one slide overview
  - `man shm_overview` for an overview of POSIX shared memory
  - POSIX API uses file descriptor instead of IDs
  - `shm_open(...)` similar to `shmget(...)`
  - `mmap(...)` similar to `shmat(...)`, but can do much more (mapping any file descriptor to memory space)
  - `shm_unlink(...)` similar to `shmdt(...)`