## Shared memory / Memory mapped files

Để phục vụ cho việc mô phỏng các bài toán giải quyết tương tranh, chúng ta có thể sử dụng cơ chế Shared memory / Memory mapped files để tạo một vùng nhớ chung chia sẻ dữ liệu giữa các tiến trình. Các tiến trình có thể có quan hệ cha con hoặc là các tiến trình độc lập với nhau.

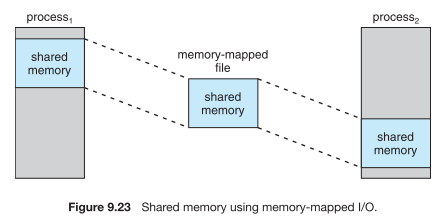
Hai phương pháp Shared memory và Memory mapped file có cùng một cách thức hoạt động

* Đầu tiên là mở hoặc tạo mới một tệp. Shared memory dùng hàm shm\_open() còn Memory mapped file sử dụng hàm open() thông thường.
* Tiếp theo mỗi tiến trình sẽ tải nội dung tệp vào RAM và ánh xạ vùng nhớ riêng của mình trong bộ nhớ ảo tới vùng nhớ chia sẻ chung đó, sử dụng hàm mmap().
* Thay đổi ở vùng nhớ chung có thể được đồng bộ lại về tệp ban đầu qua hàm msyn().
* Khi chia sẻ xong, tiến trình có thể ngắt ánh xạ tới vùng nhớ chung qua hàm munmap()
* Cuối cùng tệp sẽ được đóng lại hoặc xóa. Memory mapped file dùng hàm close() thông thường, còn Shared memory xóa tệp bằng hàm shm\_unlink()

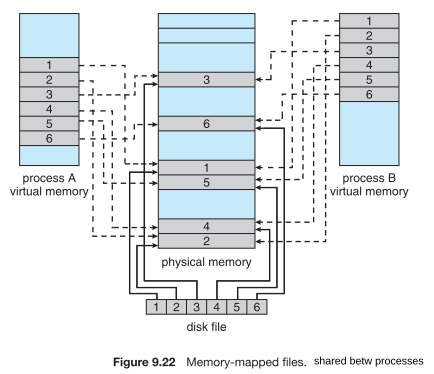


Điểm khác biệt là với Shared memory, chúng ta không quan tâm tới sự tồn tại của tệp cũng như việc cập nhật nội dung thay đổi xuống tệp, do vậy cần ít thao tác hơn. Trong cài đặt thực tế, hàm shm\_open() thực chất cũng chỉ là thêm phần đường dẫn tới một thư mục gắn với RAM (RAM disk), chẳng hạn /dev/shm và gọi hàm open() để mở/tạo một tệp trong đó.

Ở mức logic, chúng ta thấy các tiến trình ánh xạ vùng nhớ riêng của mình tới một vùng nhớ được chia sẻ chung. Vùng nhớ này thực chất là ảnh nội dung của một tệp.



Ở mức cài đặt, việc ánh xạ vùng nhớ chung vào vùng nhớ của từng tiến trình được thực hiện theo nguyên tắc ánh xạ trang bộ nhớ ảo sang trang bộ nhớ thực như trong hình dưới đây. Điều này dẫn tới thực tế là vùng nhớ của mỗi tiến trình được ánh xạ vào cùng một vùng nhớ thực chung nhưng sẽ có các địa chỉ ảo khác nhau. Do vậy trong shared memory chúng ta phải tránh sử dụng các con trỏ, chẳng hạn như danh sách móc nối. Con trỏ trong danh sách móc nối chỉ có thể trỏ tới 1 phần tử tiếp theo, tuy nhiên vấn đề là phần tử tiếp theo lại có địa chỉ ảo khác nhau trong mỗi tiến trình nên sẽ gây lỗi thực thi.



Các hàm được sử dụng trong shared memory

|  |  |  |
| --- | --- | --- |
| **Hàm** | **Tham số** | **Mô tả chức năng** |
| int shm\_open (const char \*name, int oflag, mode\_t mode); | name: tên của đối tượng  oflag: các chế độ mở đọc/ghi/tạo tệp  mode: thuộc tính tệp được tạo ra | Tạo / mở một tệp làm đối tượng shared memory |
| int shm\_unlink (const char \*name); | name: tên đối tượng shared mem | Xóa tệp đối tượng shared memory |
| int ftruncate(int fd, off\_t length); | fd: file descritor của đối tượng  length: độ dài thiết lập | Điều chỉnh độ dài vùng nhớ chia sẻ |
| void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset); | addr: địa chỉ vùng nhớ của tiến trình;  length: độ dài vùng nhớ;  prot: chế độ protection vùng nhớ;  flags: chia sẻ hay dùng riêng;  fd: file descriptor của đối tượng shared mem;  offset: địa chỉ thành phần của file | Tải nội dung tệp, tạo ánh xạ từ vùng nhớ ảo của tiến trình tới đối tượng bộ nhớ shared memory |
| int munmap(void \*addr, size\_t length); | addr: địa chỉ vùng nhớ tiến trình;  length: độ dài vùng nhớ | Xóa ánh xạ tới đối tượng shared memory. Khi tiến trình kết thúc, các ánh xạ cũng được tự động xóa. |

Ví dụ: Đoạn mã dưới đây chia sẻ một cấu trúc dữ liệu struct permission giữa tiến trình cha và tiến trình con. Tiến trình cha chờ tới khi tiến trình con kết thúc và in nội dung các trường được gán bởi tiến trình con. Hãy biên dịch và chạy chương trình dưới đây.

// sharedmem1.c

*/\* Code Listing 3.10:*

*Using POSIX shared memory to exchange data between processes*

*\*/*

*/\* Create unsized shared memory object;*

*return value is a file descriptor \*/*

**int** shmfd = shm\_open ("OpenCSF\_SHM", O\_CREAT | O\_EXCL | O\_RDWR, S\_IRUSR | S\_IWUSR);

assert (shmfd != -1);

*/\* Resize the region to store 1 struct instance \*/*

assert (ftruncate (shmfd, **sizeof** (**struct** permission)) != -1);

*/\* Map the object into memory so file operations aren't needed \*/*

**struct** permission \*perm = mmap (NULL, **sizeof** (**struct** permission),

PROT\_READ | PROT\_WRITE, MAP\_SHARED, shmfd, 0); *// addr==NULL allows the kernel to choose the addr for the mapping*

assert (perm != MAP\_FAILED);

*/\* Create a child process and write to the mapped/shared region \*/*

**pid\_t** child\_pid = fork();

**if** (child\_pid == 0){ *// child process*

perm->user = 6;

perm->group = 4;

perm->other = 0;

*/\* Unmap and close the child's shared memory access \*/*

munmap (perm, **sizeof** (**struct** permission));

close (shmfd);

**return** 0;

}

*/\* Make the parent wait until the child has exited \*/*

wait (NULL);

*/\* Read from the mapped/shared memory region \*/*

printf ("Permission bit-mask: 0%d%d%d\n", perm->user, perm->group, perm->other);

*/\* Unmap, close, and delete the shared memory object \*/*

munmap (perm, **sizeof** (**struct** permission));

close (shmfd);

shm\_unlink ("OpenCSF\_SHM");

**Bài 1**: Hãy dùng cơ chế shared memory thực hiện công việc sau

* Tiến trình cha ghi 1 xâu ký tự vào vùng nhớ chia sẻ và chờ tiến trình con kết thúc
* Tiến trình con chuyển thông điệp thành chữ hoa
* Tiến trình cha in ra thông điệp chữ hoa chuyển đổi bởi tiến trình con

Cho trước khung chương trình có tên sharedmem2.c, hãy hoàn tất phần còn lại và nộp qua google form.

**Bài 2**: Viết chương trình tạo 2 tiến trình con mô phỏng bài toán tương tranh gửi và rút tiền. Mỗi tiến trình con lặp lại 5 lần. Tiến trình gửi tiền mỗi lần tăng tài khoản lên 3, tiến trình rút tiền mỗi lần giảm tài khoản đi 2. Quan sát xem có sự cố về race condition xảy ra không?

Cho trước khung chương trình có tên guiruttien\_process.c, hãy hoàn tất phần còn lại và nộp qua google form.