sfork

A fork with flexible shared memory regions

Supervised by:
Prof. Debadatta Mishra

Team RPC

- Chinmay Goyal
- Prashant Kumar
- Rishik Jain

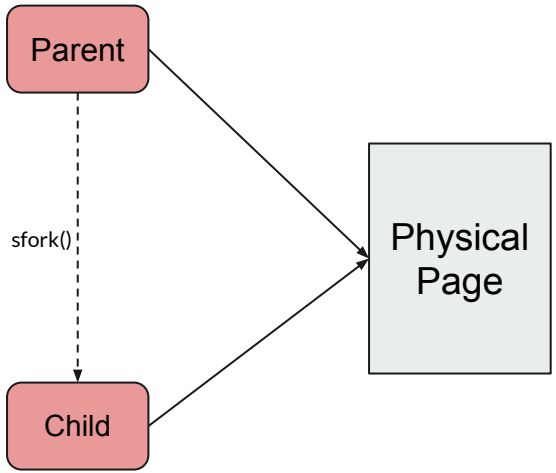
Group efforts

All group members have contributed equally to the project.

Understanding the Problem	File backed mapping	SFORK_POPULATE	Lazy Allocation	Failed exec implementation
Understand the need for sfork even though shared memory segments already exists	Implemented simple version of sfork by using VM_SHARED flag and file backed mapping	Implementing sfork through allocating all the memory at once. Comparatively easier to implement as only COW changes needed to be handled.	Allow for lazy allocation of shared memory pages. implemented in memory data structure shared_area_struct to account for the processes.	Tried to preserve shared memory segments across exec but failed to implement the solution completely and correctly.

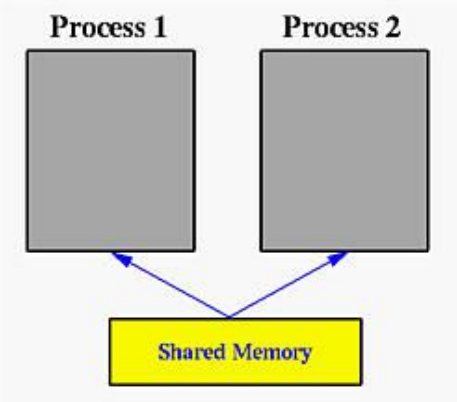
Objective

- Implement a variant of fork() syscall to include memory sharing between parent and child processes
- Add functionality to support directional behaviour in shared memory segments
- Compare *sfork* with existing shared memory mechanisms



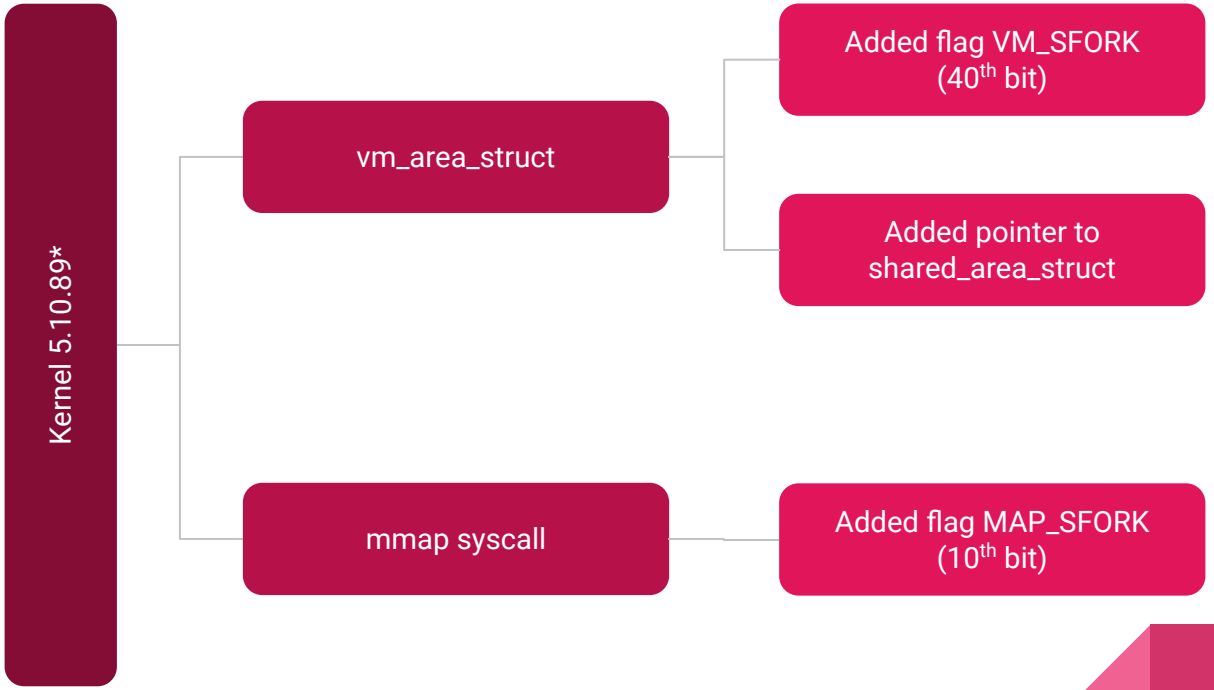
Existing implementation

- ‘shm’ APIs in the linux kernel
- Each process attaches its virtual space to this segment using an ID
- Shared memory segments are inherited across fork in a bidirectional manner



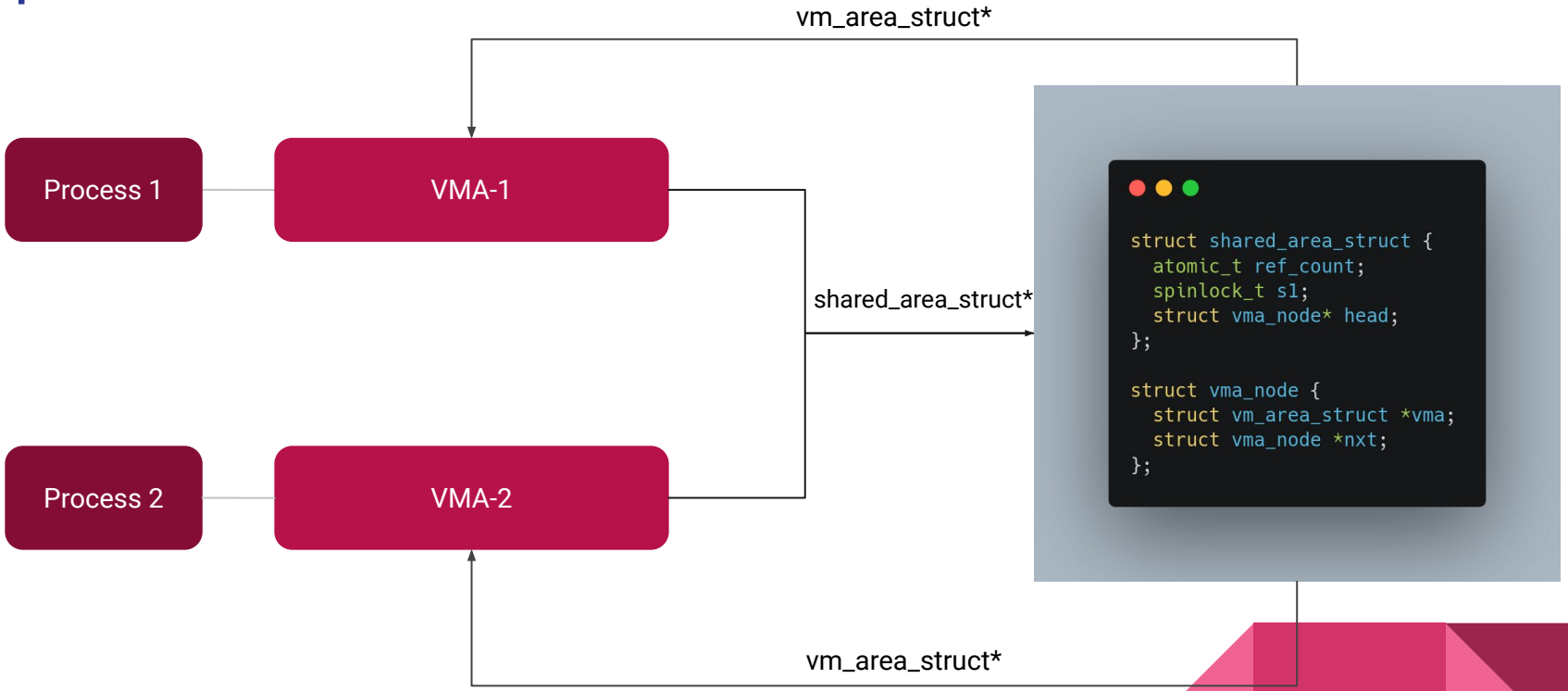

```c
int shmid = shmget(...);
void *mem = shmat(shmid, NULL, 0);

int pid = fork();

if(pid == 0){
  // child process
  /* do something */
  shmdt(mem);

}else{
  // parent process
  /* do something */
  shmdt(mem);
}
return 0;
```

Implementation details



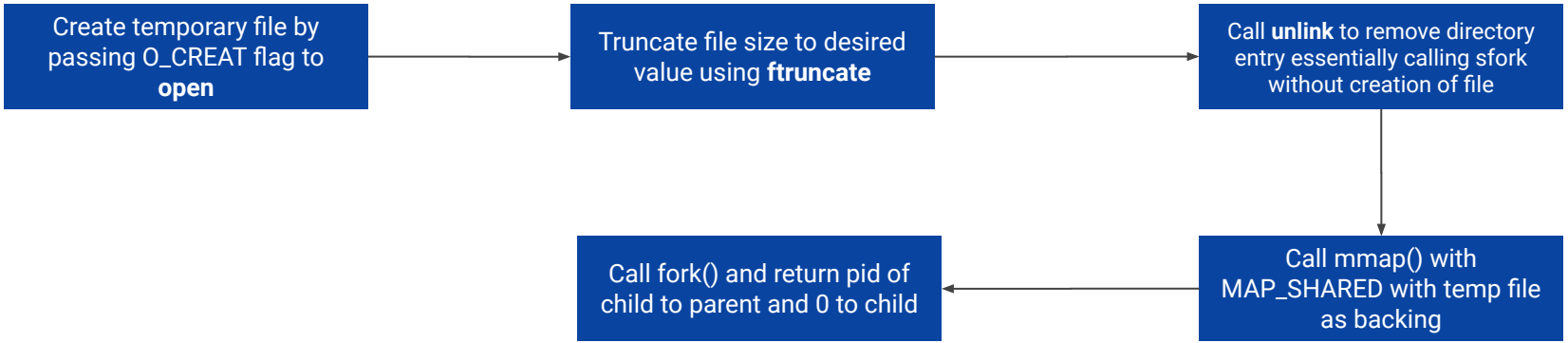
*Other implementation details not shown in the figure

Implementation details



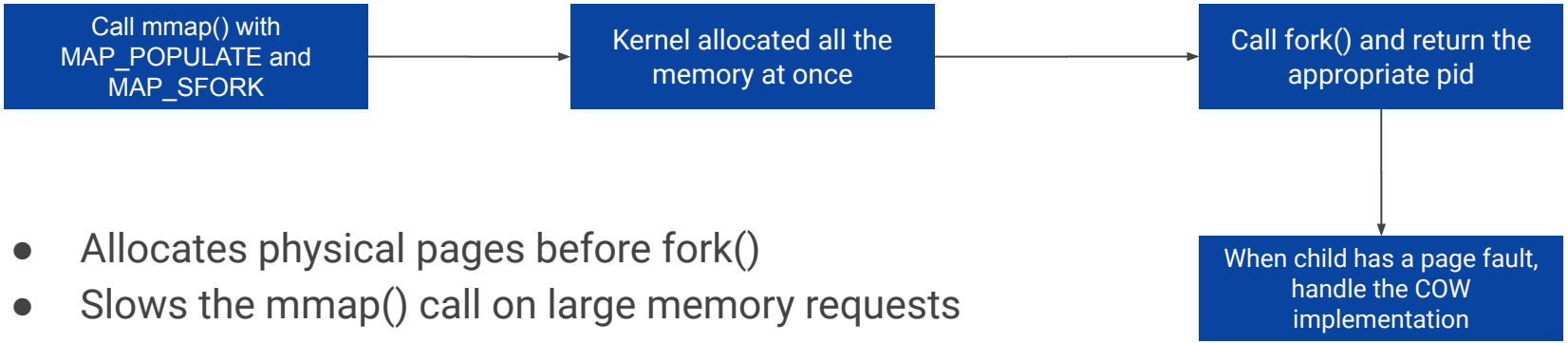
Design I

File Backed Mapping



Design II

Populate pages immediately

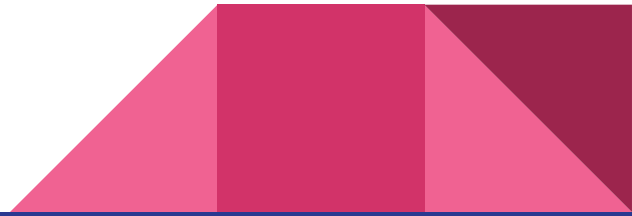


- Allocates physical pages before fork()
- Slows the mmap() call on large memory requests
- Users call sfork with SFORK_POPULATE flag

Design III

Lazy allocation

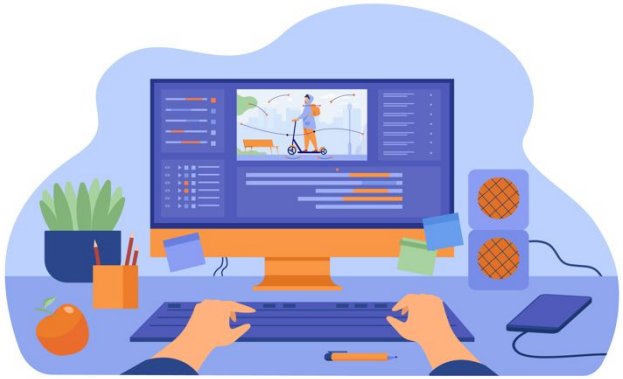
- Allocates physical pages on-demand
- When the first page fault occurs, physical page is allocated to all the faulting process.
- The process then takes a lock on *shared_area_struct* and allocate the same physical page for the same virtual address for all *vm_area_struct* present in the struct
- Time consuming if a large number of processes access the shared segment due to locking



Demo

Comparative Results

- System Configuration:
  - Operating System: Ubuntu 20.04.4 LTS x86_64
  - Kernel Source Code: linux 5.10.89 (modified)
  - CPU Qemu Virtual version 2.5+ (8) @2.207GHz
  - Memory: 3932 MB

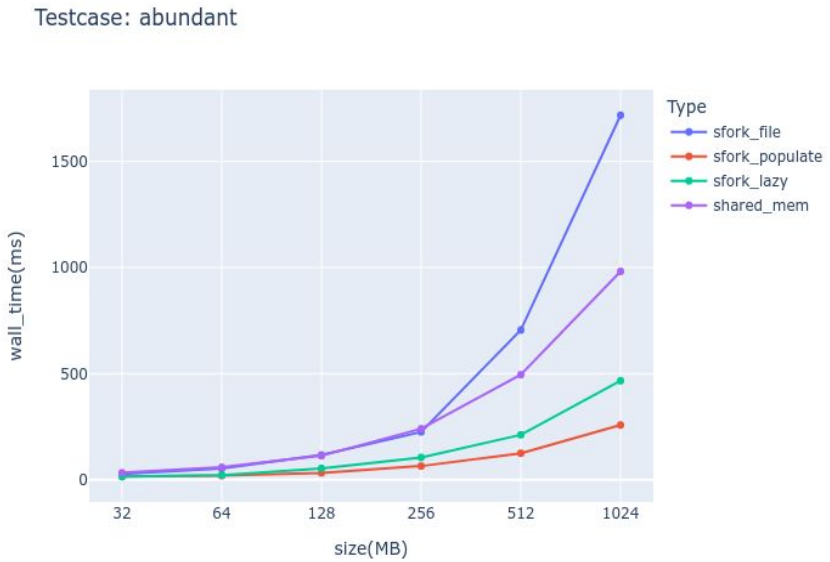


Comparative Results

While comparing the performances of the different implementations, a certain amount of memory was allocated and then accessed from the corresponding processes.

Abundant Access

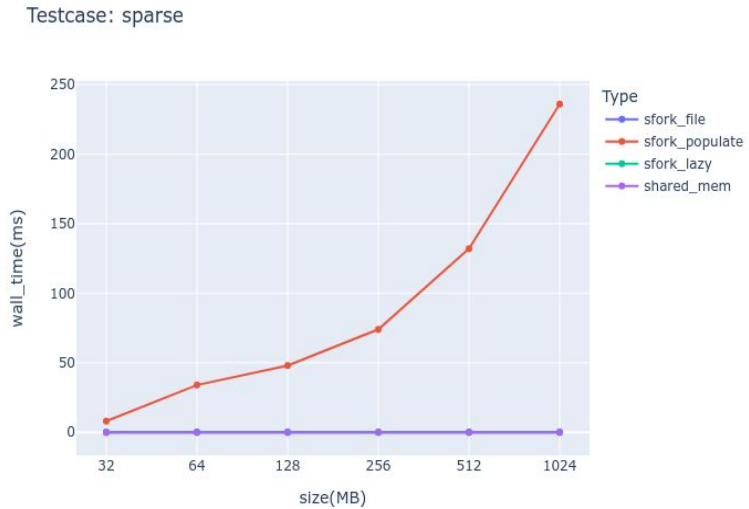
- In this test case, all the pages that have been allocated are accessed by the child and the parent processes. Only 2 processes exist in this case



Comparative Results

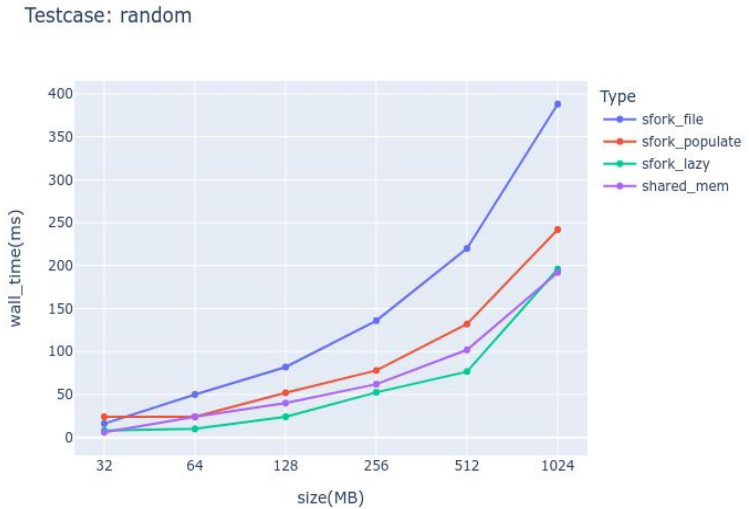
Sparse Access

Only 6 (3 each for child and parent) pages are accessed by the child and the parent processes. Only 2 processes exist in this case.



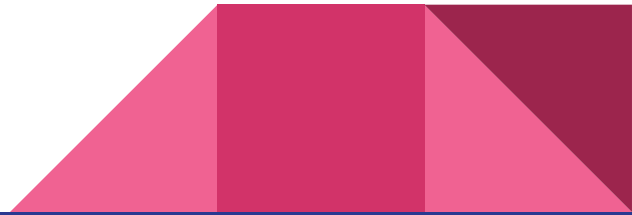
Random Access

¼ pages are randomly accessed by the child and the parent processes.



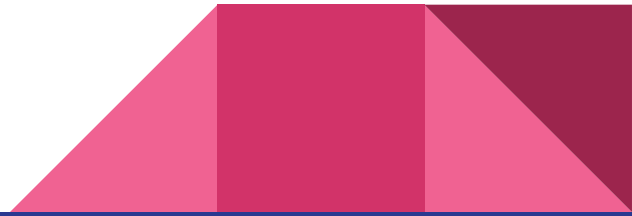
Conclusion

- sfork performs better than shm.
- SFORK_POPULATE performs better when the processes access a major chunk of the memory that they allocate.
- File backed implementation of sfork is comparatively slower, which shows that file backed mappings have some unnecessary overhead
- Usage of on-demand allocation of memory can be left to the discretion of the user when calling sfork.



Future Work

- The project can be extended to include the shared memory segments even after exec(). We tried to implement this but ran into troubles and could not make a working implementation
- Optimizations to the TLB for the shared memory segments. As the physical mapping of the virtual addresses would be same across the processes sharing the memory regions, the TLB does not need to be flushed for these entries.



Q&A

Thank You!!