[Lab-8/9] System calls & Kernel modules

소프트웨어학과 201921085 곽수정

- **Describe your design and implementation**

- **User-space program**

User space 프로그램은 main 함수에서 생성할 랜덤 숫자 개수를 num으로 받아오고, num 만큼 int를 저장할 arr를 생성하였습니다. rand() 함수를 통해 arr에 2~num 사이의 랜덤한 정수를 저장하고, arr 안의 정수들이 prime number인지 아닌지 판별하였습니다. arr 안의 수가 2가 아니면서 2로 나누어지거나 자기 자신이 아닌 숫자로 나누어지는 경우는 소수가 아니므로 제외하고, 자기 자신 이외의 자연수로 나눌 수 없는 경우에만 answer에 1을 더하여 소수 개수를 누적시키고 출력하였습니다. Performance를 비교하기 위해서 시간을 측정하는 gettimeofday() 함수를 사용하였고, 다른 프로그램과 동일하게 비교하기 위해 num을 출력하는 부분부터 소수 개수를 출력하는 부분까지 시간을 측정하고 이를 micro second로 출력하였습니다.

- **Syscall with user-space test program**

Syscall with user-space test 프로그램은 테스트를 위한 prime_sys_test.c 파일과 prime_sys.c로 나뉘게 됩니다. 먼저 prime_sys_test.c 에서 생성할 랜덤 숫자 개수를 num으로 받아오고 syscall() 함수를 통해 syscall table에 저장한 549번 시스템 호출과 함께 num 인자를 넘겨주게 됩니다. syscall의 return 값은 소수 개수가 되고 answer에 저장되어 출력됩니다. 또한 성능 측정을 위해 gettimeofday() 함수를 사용하였으며 syscall 함수 호출 전, 후에 위치하여 시간을 측정하고 이를 micro second로 출력하였습니다.

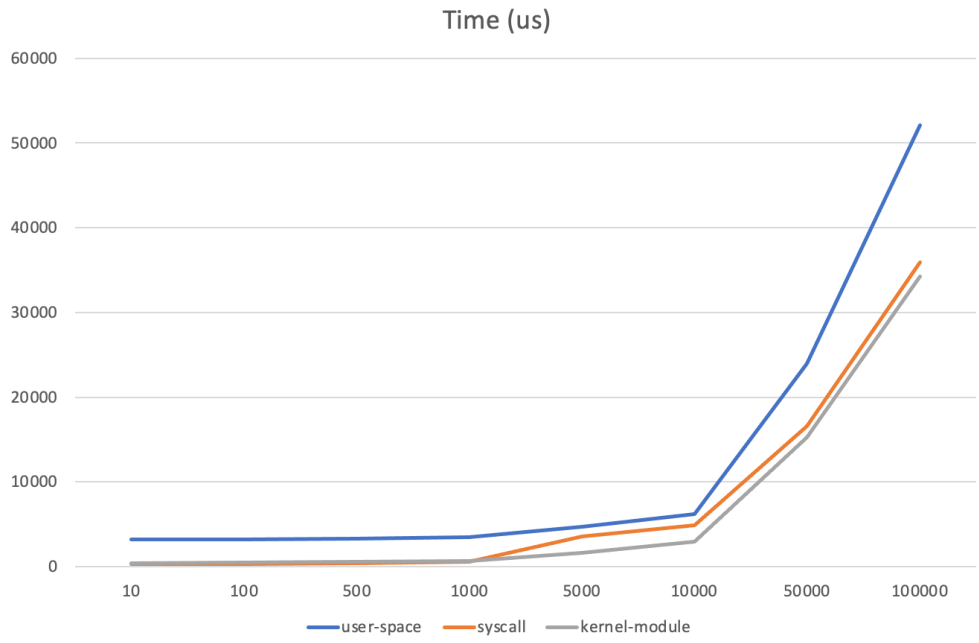
함수를 정의한 prime_sys.c 에서는 하나의 인자를 받아오기 위해 SYSCALL_DEFINE1 매크로를 사용하여 const int __user* 형 num 변수를 가져왔습니다. 사용자 영역의 데이터를 커널 영역으로 가져오기 위해서 copy_from_user 를 사용하였으며, 이를 통해 사용자 영역에 있는 num 의 주소를 base 로 sizeof(int) 크기만큼 커널 영역의 주소 &knum 에 copy 합니다. kernel 에서 arr 의 memory allocation 을 위해 kmalloc 을 사용하였는데, 연속된 physical 메모리를 연속된 virtual 공간에 매핑하여 할당하는 것이 page table mapping 에 있어서 vmalloc 보다 효율적일 것이라고 생각하여 kmalloc 을 사용하였습니다. 이후에 생성할 랜덤 숫자 개수를 출력하고, arr 안에 랜덤한 수를 넣어주고 (get_random_bytes 함수 사용) arr 안의 수가 소수인지 아닌지 판별하는 코드는 user-space 프로그램과 비슷하며 kfree 로 메모리 할당을 해제하고 소수 개수를 return 하였습니다.

- **Kernel module**

Kernel module에서는 생성할 랜덤 숫자 개수를 module_param을 통해 n으로 전달받도록 하였습니다. module_init에 의해 lkp_init 함수가 실행되며 이 함수 내에서 arr를 kmalloc을 통해 메모리를 할당하고, arr 안에 랜덤한 수를 넣고, 소수가 맞는지 아닌지 판별하고 이를 출력하는 부분은 syscall with user-space program 과 동일합니다. 성능 측정을 위해 ktime_get_ts64 함수를 사용하였으며 이를 micro second 시간으로 환산하여 출력하도록 하였습니다.

- **Compare the performance among them**

성능 측정을 위해 user-space, syscall, kernel module에서 gettimeofday() 또는 kime_get_ts64() 함수를 사용하여 micro second의 실행시간을 얻었습니다. 측정마다 11번씩 측정하여 그 중앙값(us)을 기록하였으며, n을 10 100 500 1000 5000 10000 50000 100000로 늘려가며 측정해보았습니다.



n	user-space	syscall	kernel-modul
10	3247	317	390
100	3248	333	463
500	3298	382	610
1000	3469	580	685
5000	4671	3539	1609
10000	6238	4894	2943
50000	23987	16539	15281
100000	52114	35911	34295

위 그래프와 기록된 시간을 통해 syscall과 kernel module이 항상 user-space 프로그램보다 성능이 좋았음을 알 수 있었습니다. 이는 syscall과 kernel module에서는 랜덤한 수를 생성할 때 발생하는 오버헤드를 줄이기 위해 get_random_bytes를 통해 arr 크기만큼 한번에 랜덤한 수를 넣어주었지만, user-space 프로그램에서는 입력받은 n만큼 반복하여 arr에 랜덤한 수를 넣어주었기 때문에 user-space 프로그램이 가장 오랜 시간이 걸린 것으로 생각하였습니다.

또한 syscall with user space test program와 kernel module을 비교하였을 때 n의 크기가 작을 때는 syscall이 더 오래걸리지만 n이 커질수록 성능이 비슷해졌습니다. kernel module에서는 module_param으로 인자를 받아왔으며, 인자를 받아온 이후부터 시간 측정을 시작하였습니다. 반면 syscall에서도 SYSCALL_DEFINE1 매크로를 통해 변수를 받아왔으나, 사용자 영역의 데이터를 커널 영역으로 가져오기 위해서 copy_from_user를 사용하였습니다. 이를 시간 측정 코드 범위에 포함시켜서 시간을 측정하였고 copy_from_user 함수에서 시간이 더 걸렸기 때문에 syscall이 kernel module 보다 시간이 조금 더 걸린 것으로 보았습니다.