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

소프트웨어학과 201720707 나용성

1. Describe your design and implementation

   user-space program, user-space test program, kernel module 이 세 프로그램은 모두 다음과 같은 과정을 통해 요구사항을 수행한다.

   1) 프로그램 실행 파라미터 n을 읽는다;
   2) 2~n범위의 자연수 n개를 갖는 배열을 생성한다.
   3) 배열에 있는 숫자 중 소수가 몇개인지 판별한다.
   4) 결과를 출력한다.
   5)

   먼저 1)과정은 user-space program과 user-space test program은 user-space에서 getopt를 이용해 프로그램 실행 시 input으로 주어진 n값을 읽어오도록 했고 kernel module에선 module_param api를 이용해 n값을 읽도록 했다.

   그 다음 2)과정은 user-space program의 경우엔 모두 user-space에서 수행해야 하므로 malloc으로 포인터 변수 arr에 n개의 int형 변수를 저장할 수 있는 크기의 메모리를 할당하고 rand() api를 활용해 n개의 2~n범위의 자연수를 저장했다. 그리고 system call을 활용한 user-space test program은 system call을 호출하는 횟수를 최소화하기 위해 systemcall을 n을 파라미터로 입력받아 kernel space에서 arr에 메모리를 할당하고 무작위 숫자를 생성해 소수 판별을 하는 과정까지 모두 한번에 처리하도록 하여 user-space에선 한번의 system call 만 호출하도록 하는것이 좋을 것이라 생각했다. 때문에 systemcall은 SYSCALL_DEFINE1 타입으로 선언했고 n은 user-space에 저장된 변수이므로 kernel-space에서 효율적으로 사용할 수 있도록 kernel_space에 n_kernel이라는 변수를 선언하고 copy_from_user 명령을 통해 n_kernel에 n에 저장된 값을 복사하고자 했다. 때문에 system call에서 입력받는 파라미터를 const unsigned int __user*, n인 포인터 변수로 선언하여 주소값을 system call의 파라미터로 입력받아 copy_from_user(&n_kernel, n, sizeof(unsigned int))를 실행하여 n_kernel에 n의 값을 복사했다.

   그 다음 user-space test program의 system call과 kernel module은 비슷하게 구현 했는데 먼저 kernel-space에서 배열을 할당해야 하므로 arr에 kmalloc을 이용해 메모리를 할당했다. Kmalloc은 x86 architecture에서 최대 4MB까지 할당할 수 있다고 하는데 4MB에는 2의 20승 개의 int형 변수가 저장될 수 있으므로 크기는 충분하다고 생각했고 kmalloc을 사용했을 때 physically contiguous한 메모리 공간을 가질 수 있어 성능에 이점이 있기 때문에 kmalloc을 사용했고 메모리 할당에 시간이 급한 상황은 아니므로 gfp_mask flag엔 GFP_KERNEL flag를 사용했다. 그 다음 random한 값을 생성하기 위해 get_random_bytes(arr, sizeof(int) * n_kernel)로)arr의 전체공간에 무작위 byte를 설정하고 for문을 통해 arr[i] = arr[i] % (n – 1) + 2; 코드를 수행해 각 index에 2~n범위의 값이 저장되도록 했다.

그 다음 3)과정은 세 프로그램 모두 공통으로 선언된 is_Prime함수를 arr의 모든 index에 수행하여 결과를 result변수에 더해 소수의 개수를 판별했다.

```
unsigned int
is_Prime(unsigned int num)
{
        unsigned int i;
        if(num % 2 == 0){
                if(num == 2) return 1;
                else return 0;
        }
        for(i = 3; i * i <= num; i+=2)
                if(num % i ==0) return 0;
        return 1;
}
```

그림1. Is_Prime함수의 코드

시간 측정은 user-space program, system call user-space test program은 gettimeofday api를 이용했고 kernel module은 ktime_get_ts64 api를 이용해 측정해 us scale로 출력되도록 했다.

2. Compare the performance among them

실행시간 성능측정은 세 프로그램의 n값이 10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000일 때의 실행시간을 각각 10회 측정한 값의 중앙값을 기록하여 수행했다.

n	10	50	100	500	1000	5000	10000	50000	100000
user-space program	3763	3785	3848	3942	4281	6291	8695	36701	82348
System call user-space test program	1143	1167	1180	1255	1458	2908	5054	26394	57579
Kernel module	153	173	186	289	420	1919	4300	18972	42375

측정한 값을 통해 안 사실을 설명해보자면 먼저 실행시간은 user-space program이 가장 느리고 그 다음으로 system call을 구현해 사용한 user-space test program이 빠르며 kernel module에서 가장 빠르다. 이 때 n이 클 때 user-space test program의 실행시간은 user-space program보단 kernel module에 가깝다. 그러므로 kernel space에서 연산을 수행할 때 성능의 이점이 있다는 것을 알 수 있다. 그 다음 user-space program과 user-space test program은 n이 작을 때도 3700, 1100 us이상의 시간이 필요했지만 kernel module에서 100us대로 매우 빨랐으며 n이 1000까지 상승할 때 앞의 두 프로그램은 4200, 1400 us정도로 n이 10일 때의 실행시간과 비교해 비율로 보면 큰 상승이 없었던 반면 kernel module에선 n이 1000일 때 420의 실행시간으로 약 3배가량 실행시간이 늘어났다. 그러나 절대값을 따졌을 땐 세 프로그램에서 각각 518, 315, 267 us 상승 했으므로 절대값 자체는 비슷하게 상승했다. 이를 통해 user-space를 사용하는 두 프로그램은 user-space에서 getopt를 통해 파라미터를 읽고 atoi로 변환하고 systemcall을 호출하는 등의 overhead가 있어 n이 작아도 최소요구 실행시간이 크고 kernel module은 다른 두 프로그램과 같은 overhead가 없으므로 n이 작으면 작은만큼 실행시간이 짧아지는 것이라고 생각할 수 있을 것 같다.