집중교육2 lab8_9 assignment

201521032 한태희

0. 개요

lab8에서는 새로운 리눅스 System Call을 만드는 방법에 대해서 배웠고, lab9에서는 새로운 Kernel Module을 만들고 insert하는 방법에 대해서 배웠다. 이 지식을 바탕으로, 정수 배열을 랜덤 생성한 뒤, 배열 속에 소수가 몇 개 들어가 있는지 판별하는 코드를 유저 모드에서 작동하는 버전, System call을 이용하여 작동하는 버전, Kernel Module init에서 작동하는 버전 3가지를 구현한다. 그리고 3가지 버전의 성능을 측정한 뒤 비교한다.

1. 디자인과 구현

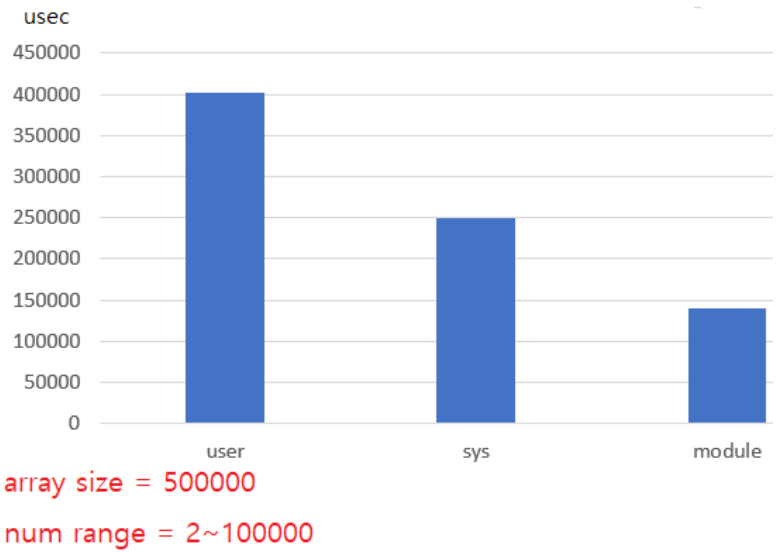
소수 판별 알고리즘은 판별할 정수 N이 있으면 2 부터 $\sqrt{N}$ 사이의 정수 i를 모두 나누어 나누어 떨어지는 것이 하나도 없으면 소수로 판별하는 방법을 사용하였다. (약수의 성질에 의해 전체를 검사할 필요는 없다)

System call을 이용하는 방법은 기존 유저모드에서 작동하는 코드에서, NSIZE 크기만큼의 정수 배열 arr를 받아 소수의 개수를 판별하는 부분을 System call을 이용하여 계산하게 수정하였다. 이때, 각 정수 하나마다 System call을 호출하여 정수인지 판별하고, 합계를 user mode에서 집계하는 방식은 System call에 의한 오버헤드가 매우 클 것이다. 따라서 System call은 한번만 사용하면 되도록 System call에 배열의 시작지점을 나타내는 int pointer 와 배열의 크기를 나타내는 int를 인자로 넘겨서, 커널 모드에서 반복문을 통해 소수의 개수를 구하고 소수의 총합을 반환하도록 구현했다. 이를 위해서 유저 모드에서 malloc으로 선언했던 메모리 공간을 copy_from_user() 함수를 이용해 kmalloc으로 선언한 커널 메모리 공간으로 복사할 수 있도록 했다.

Kernel Module을 이용하는 방법은 모듈의 init 작동시에 유저 모드에서 실행되는 내용과 동일한 절차가 실행되도록 작성하였다. (실습에서 사용했던 방법) 이러한 방법으로 실행을 하게 되면 유저 모드에서 실행되는 구간은 하나도 없이 커널에서 작동하게 된다. 모듈을 빌드할 때는 기존에 사용하던 헤더파일들을 사용할 수 없었는데, 기존 유저 코드에서 사용했던 랜덤 발생 함수인 rand()와 타임 스탬프 측정 함수인 gettimeofday()는 커널 모드에서 사용할 수 있는 방법을 찾지 못했다. 대신에 rand()를 get_random_bytes()로, gettimeofday()를 ktime_get_real()로 대체하여 사용하는 방법을 알아내어 적용하였다. rand()와 get_random_bytes()는 둘 다 임의의 수를 생성하는 의도를 가지고 작성했지만, 난수의 분포가 다른 것인지 생성되는 배열에서 평균적으로 생성되는 소수의 개수가 차이가 나는 문제가 발생했다.

2. 성능 측정과 비교

성능 측정을 할 때 배열의 크기는 500000(오십만)으로 고정하였고, 랜덤 생성되는 숫자의 범위는 2~100000(십만)으로 고정하였다. 성능 측정 결과는 usec (마이크로초) 를 기준으로 측정하였다. 시간을 측정하는 구간은 메모리 할당, 난수 생성 구간, 결과 출력 구간은 제외하고 입력받은 배열이 소수인지 판별 하는 반복문의 실행 시간만을 측정하였다. 위의 조건에서 3가지 구현한 방법을 독립변인으로 측정하였다. 측정 결과는 10번 실행한 결과의 평균이다.



측정 결과, 유저 모드가 평균 401539 마이크로 초로 제일 느린 실행 시간을 보였고, 그 다음 시스템 콜 이용이 249603 마이크로 초, 모듈 이용이 139460 마이크로초로 더 빠르게 실행되었다.

user와 sys_call의 차이점을 분석해보면, user mode로만 동작하는 코드는 운영체제가 관리하는, 프로세스마다 할당되는 가상 페이지 테이블을 거쳐야 메모리에 접근할 수 있다. 하지만 kernel mode에서 kmalloc을 이용해 할당받은 메모리 공간은 실제 physical memory와 동일한 주소 공간을 사용하기 때문에 그러한 오버헤드가 없어서 메모리 접근이 더 빨라진다. 유저 모드에서 커널 모드로 진입하기 위해선 시스템 콜, 메모리 복사 (유저 모드 공간의 배열을 커널 모드 공간으로 복사) 등의 오버헤드가 발생하긴 하지만, 그러한 페널티를 상쇄시키고도 남을 정도로 커널에서의 직접 메모리 접근이 유저 모드보다 효율이 좋은 것을 알 수 있다.

module로 구현한 경우엔 sys_call 보다도 더 효율이 좋았는데, sys_call의 경우엔 측정을 유저 영역에서 했지만, module은 측정을 커널 영역 안에서 했기 때문에 더 효율이 좋게 측정된 것으로 보인다. system call로 인한 interrupt 지연 시간도 없고, 시간 측정 함수 자체도 kernel에서 사용하는 버전을 사용했기 때문에, 더 빠른 측정이 발생한 것으로 추정된다. 또한 난수 생성 방식도 다르기 때문에, 그 문제로 인해 평균적인 숫자의 크기가 달라져 그것이 교란 요인이 되었을 수도 있다.