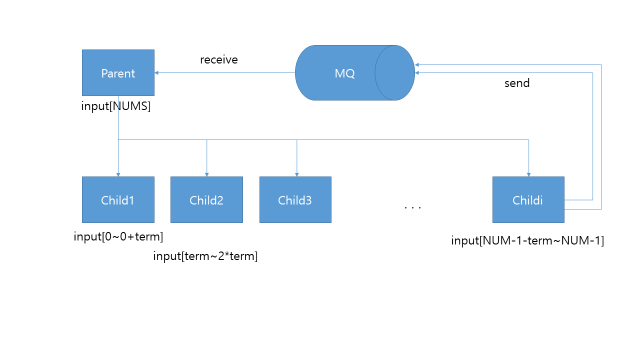
ku\_ff.c Design & Implementation

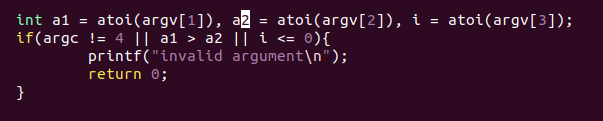
201611218\_이기현



전체적인 구조를 도식화한 그림은 위와 같습니다.

//common





프로그램 실행 시에 전달되는 a1, a2, i 값들을 받아서 예외처리를 하면서 시작합니다.

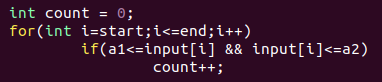


parent, children 프로세스 모두 같은 POSIX 메시지 큐를 사용하기 때문에 fork() 하기에 앞서 mq\_open 시스템 콜로 mqdes 를 저장해 놓습니다. sender 는 write 권한이, receiver 는 read 권한이 필요하기 때문에 O\_RDWR 로 둘 다 가능하게끔 선언하였습니다.



자식 프로세스 개수에 따른 배열 간격을 term 이라는 변수에 저장합니다. 예를 들어 NUMS가 20 이고 자식이 4명이라면 간격은 5가 됩니다. 자식 머릿수가 배열 개수보다 많다면 간격을 1로 설정합니다. 설정된 간격(term)은 자식 수만큼 반복문을 돌면서 그 자식이 접근할 input의 인덱스 값을 구할 때 사용됩니다. 배열 크기가 20, 자식이 4명일 때 각 인덱스 값은 [0~4], [5~9], [10~14], [15~19] 가 됩니다. 시작과 끝 범위가 start, end 에 저장됩니다.

//child



이제 자식 수만큼 fork() 하여 프로세스를 생성하고 시작 인덱스 start 부터 끝 인덱스 end 까지 조건에 맞는 값들을 카운트합니다.

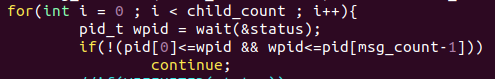


카운트가 끝나면 이 값을 메시지 큐로 send 하고 자식 프로세스는 종료합니다.

//parent



메시지 큐 안에 있는 count들을 모아 저장할 all\_count 와 메시지의 개수(msg\_count)입니다. msg\_count 는 send가 실행된 자식 명 수이고 자식 프로세스가 배열 수보다 많을 때 처리하기 위해 사용됩니다.



sslab 게시판에 써주신 것처럼 프로세스는 생성하되 넘치는 프로세스는 메시지를 send 하지않고 그대로 종료하도록 sender가 구현 되어있습니다. 따라서 부모 프로세스는 child\_count 만큼 reaping 을 하지만 send 를 실행하지 않은 자식 프로세스들에 대해선 receive를 하지 않고 continue; 합니다. send 를 실행한 자식들만 reaping 후에 receive 하는 형태입니다.





메시지 큐로부터 count 를 받아와 value에 저장하고 all\_count 에 더해줍니다.

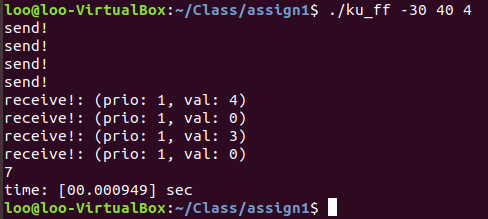


parent 가 자식들을 모두 reaping 하고 나면 all\_count를 출력하고 mq를 close, unlink 합니다.





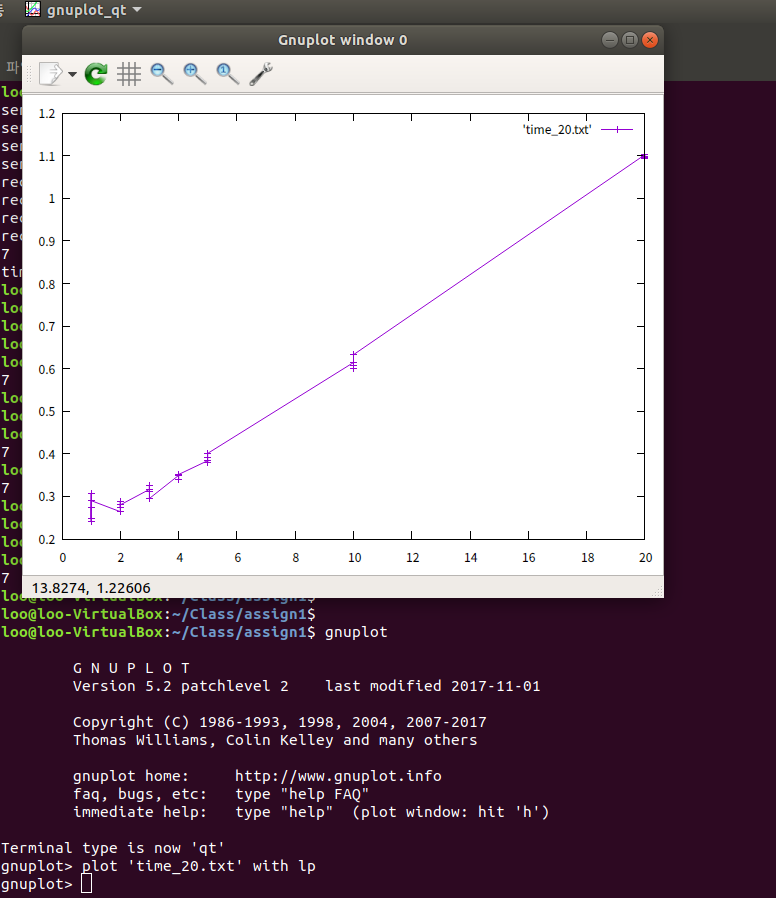
프로그램의 실행 속도를 분석하기 위해 gettimeofday 함수가 코드의 시작과 끝에 들어가 있습니다. 자식 수에 따른 실행 속도를 그리기 위해 txt 파일을 생성하고 각 4번 정도 시행하여 데이터를 얻었습니다.



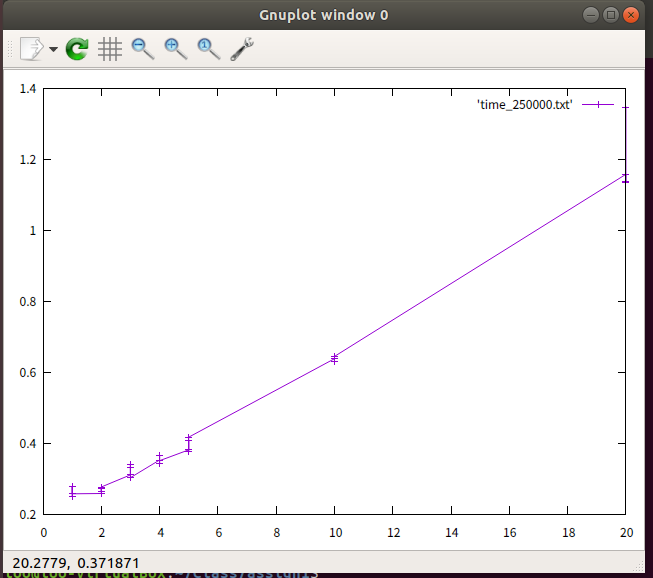
위는 NUMS 20 인 sample 파일 예시입니다. -30이상 40이하 숫자의 개수를 4명의 자식 프로세스로 분할하여 센 결과 화면입니다. priority는 프로그램에서 주 고려사항이 아니기 때문에 모두 1로 선언하여 받았습니다.

//conclusion

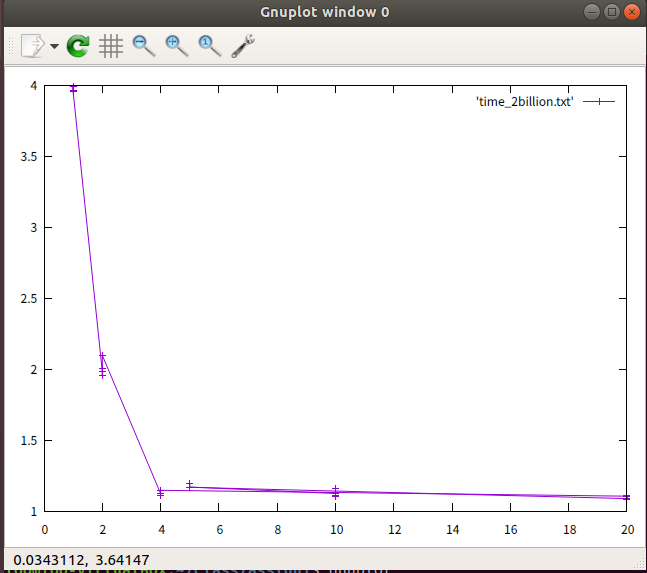
제가 사용하는 버추얼 박스의 우분투 cpu core 개수가 1개로 설정되어 있어서 프로세스 개수를 늘려도 속도 변화가 없었습니다. 그래서 cpu core 개수 제한을 4로 늘리고 그래프를 그려보았습니다. 툴은 gnuplot을 사용하였습니다.



NUMS 20 일 때 자식 수에 따른 실행 속도 그래프입니다. 예상했던 결과와 달리 프로세스 개수가 늘어날수록 실행 속도가 늘어났습니다.



데이터 개수가 적기 때문이 아닐까 생각이 들어 int 형 전역 변수의 최대치에 가까운 250000 으로 NUMS를 두고 해보았으나 결과는 비슷했습니다.



더 이상 배열 크기를 늘릴 수 없어 20억번의 반복문으로 실행 시간을 체크해 보았습니다. 프로세스가 1개 일 때 20억, 2개일 때 10억\*2, 4개일 때 5억\*4 로 늘려가며 실행했습니다.

예상했던 결과에 가까운 그래프가 그려졌습니다. 250000이라는 숫자도 프로세스 하나로 매우 거뜬하게 순식간에 끝내버리는 모습을 보고 자식을 늘려가는 fork() 과정에서 생기는 오버헤드가 실행시간을 늘린 게 아닐까 하고 예상할 수 있었습니다. 그 오버헤드를 덮을 정도로 숫자가 커야 병렬 프로그램의 의의를 나타낸다는 결론을 내렸습니다.