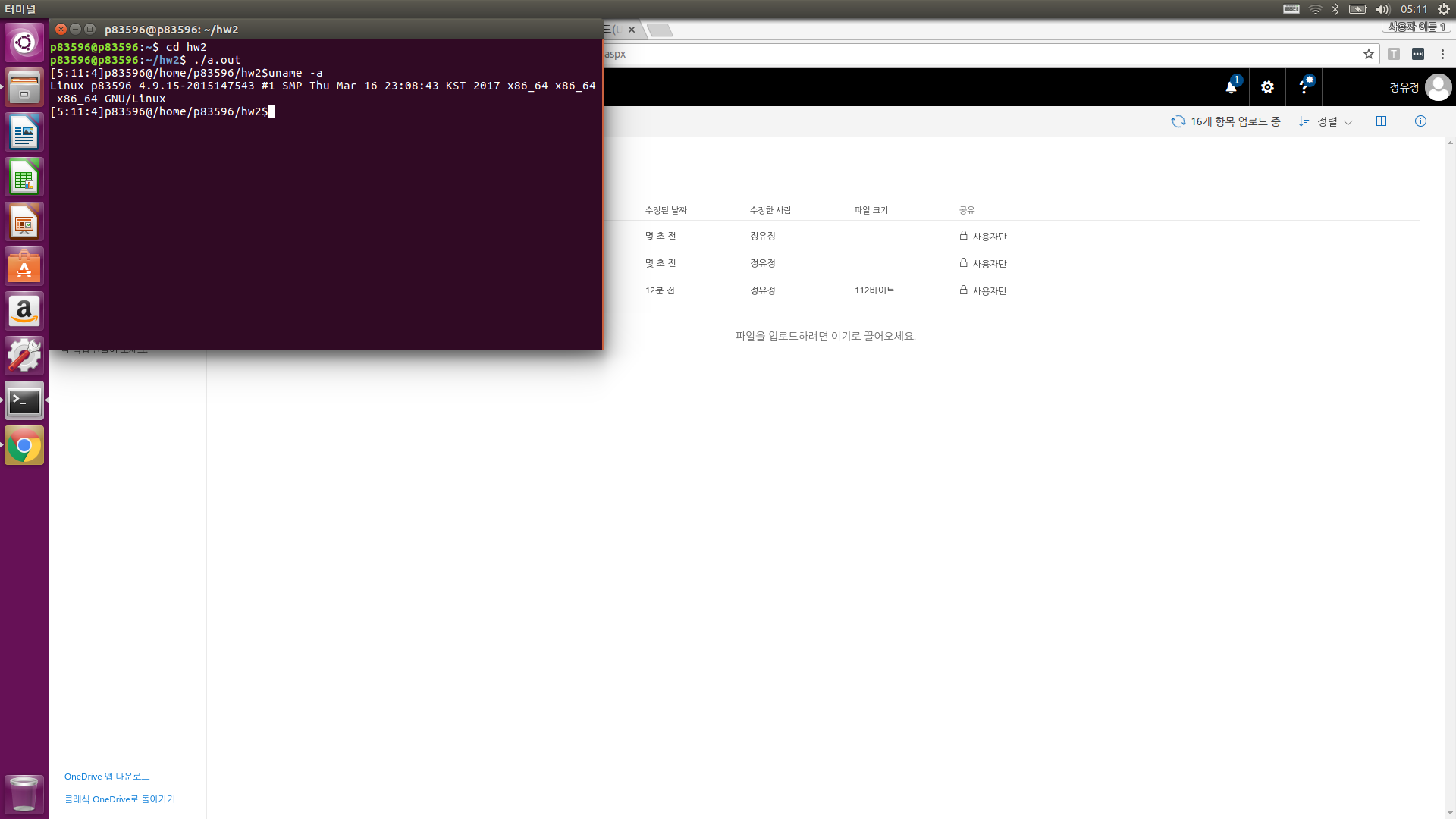
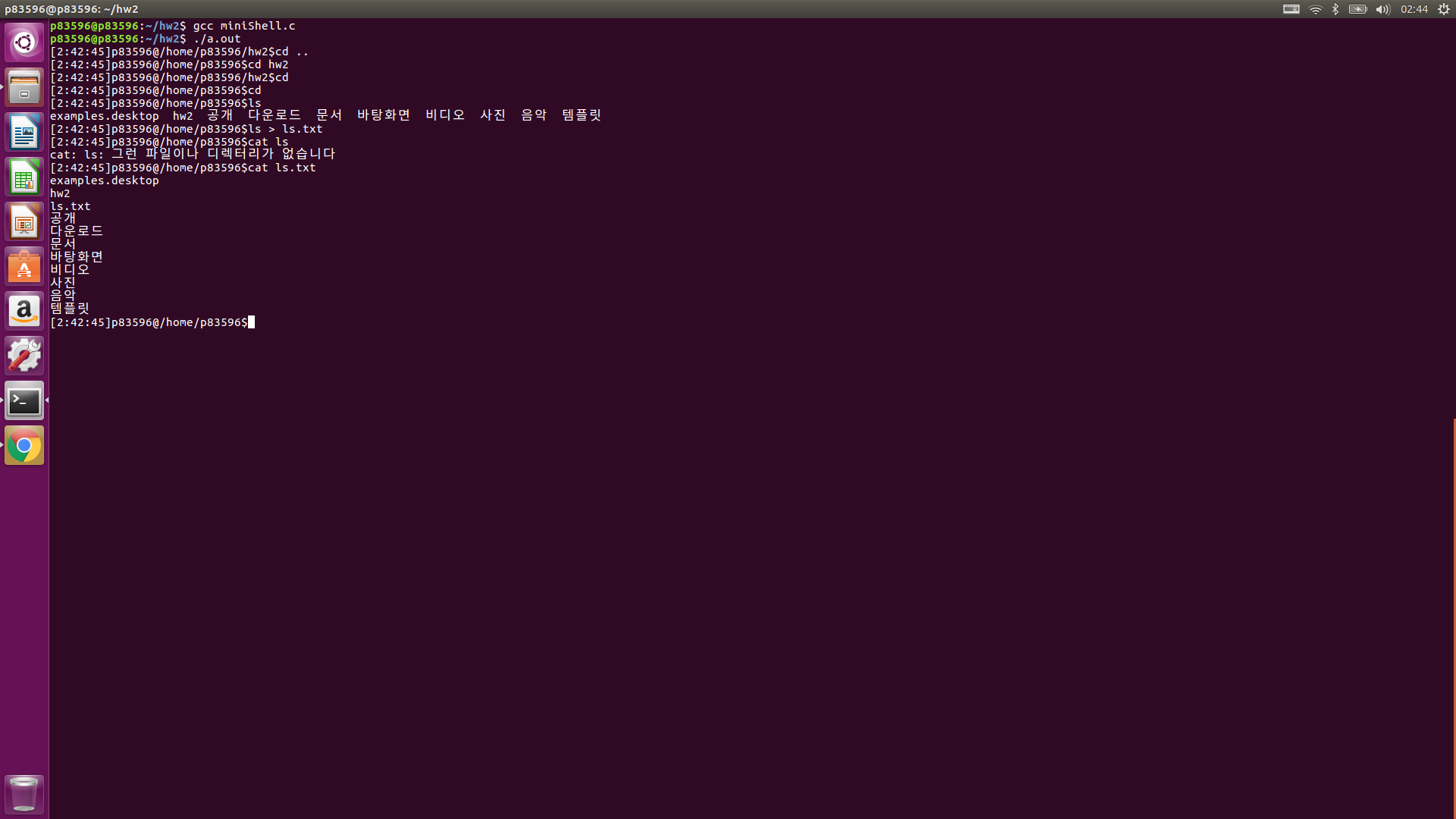
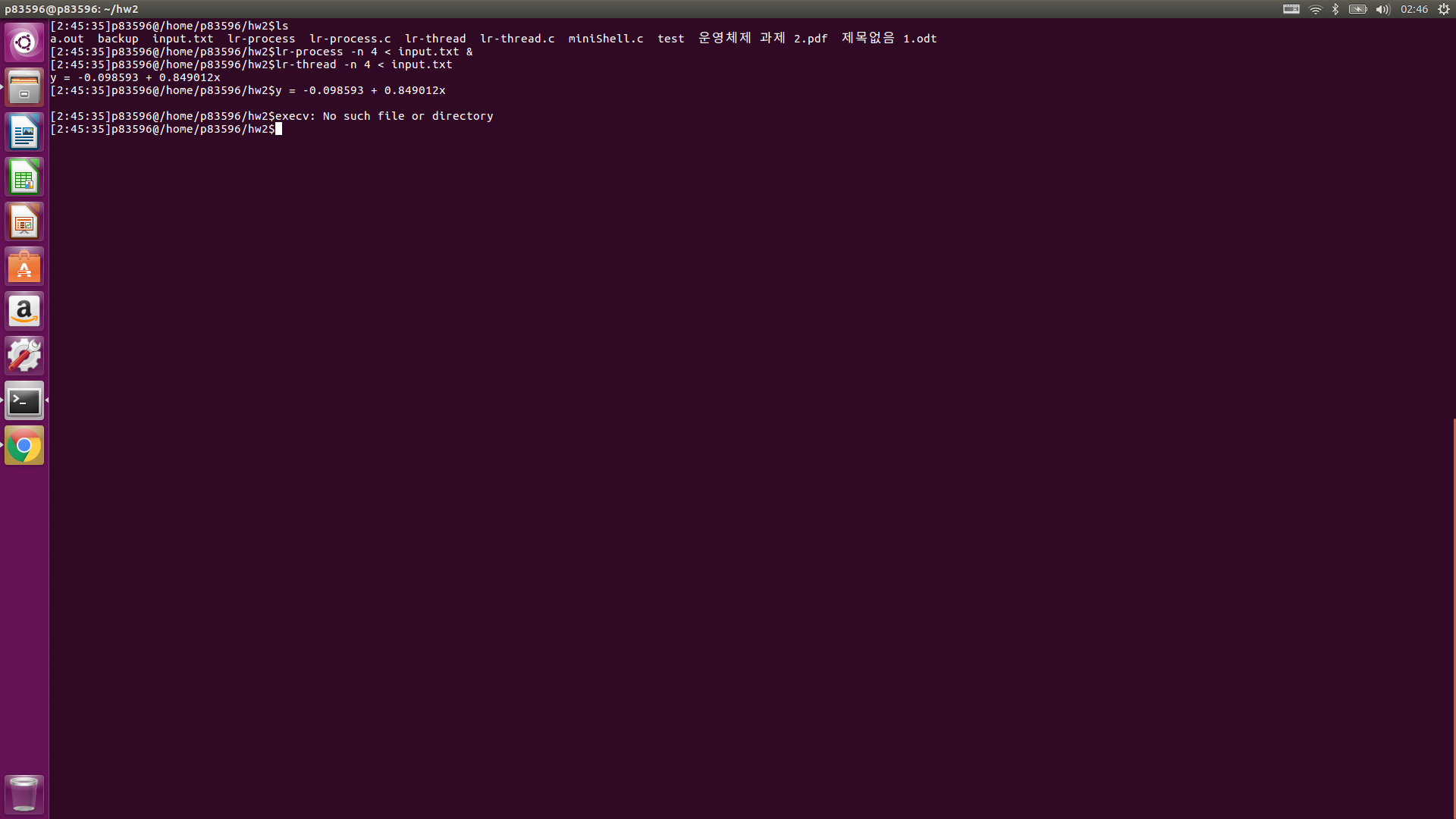
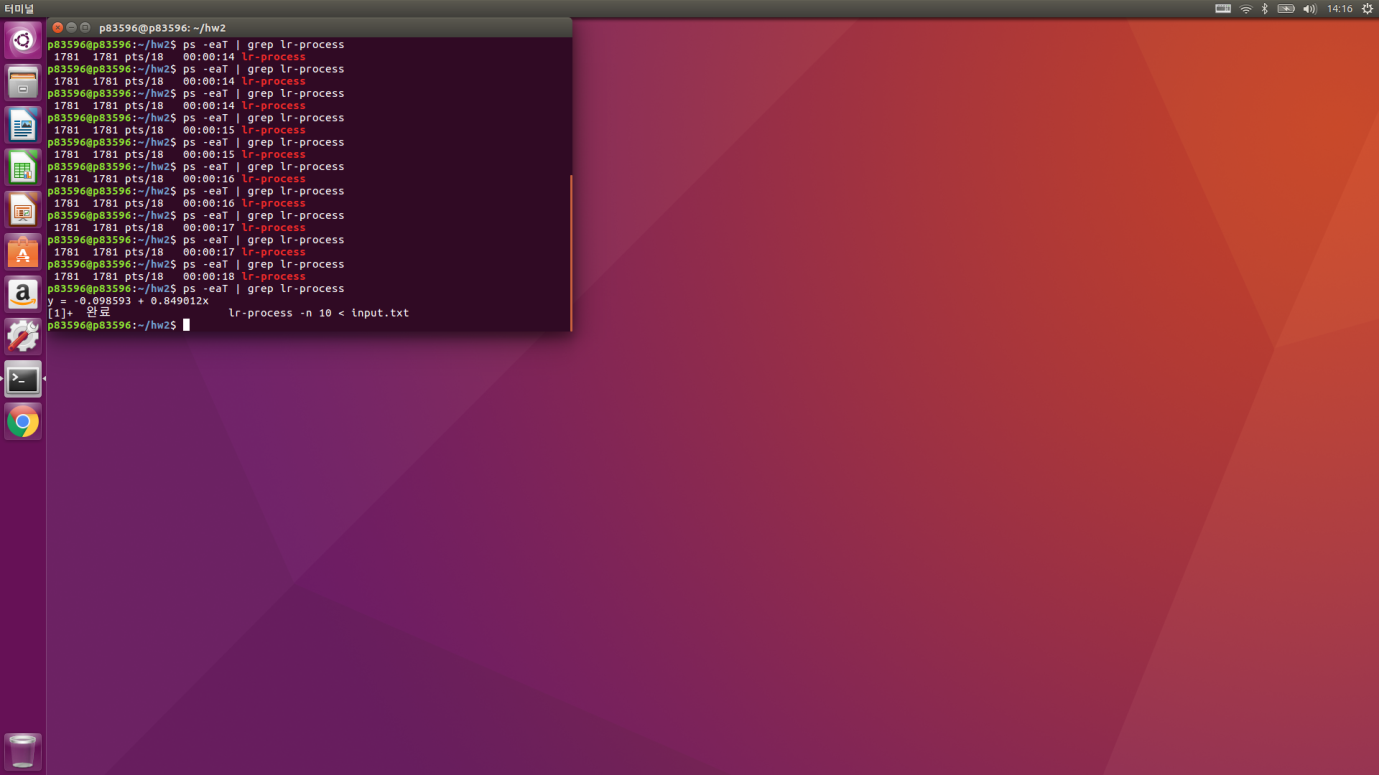
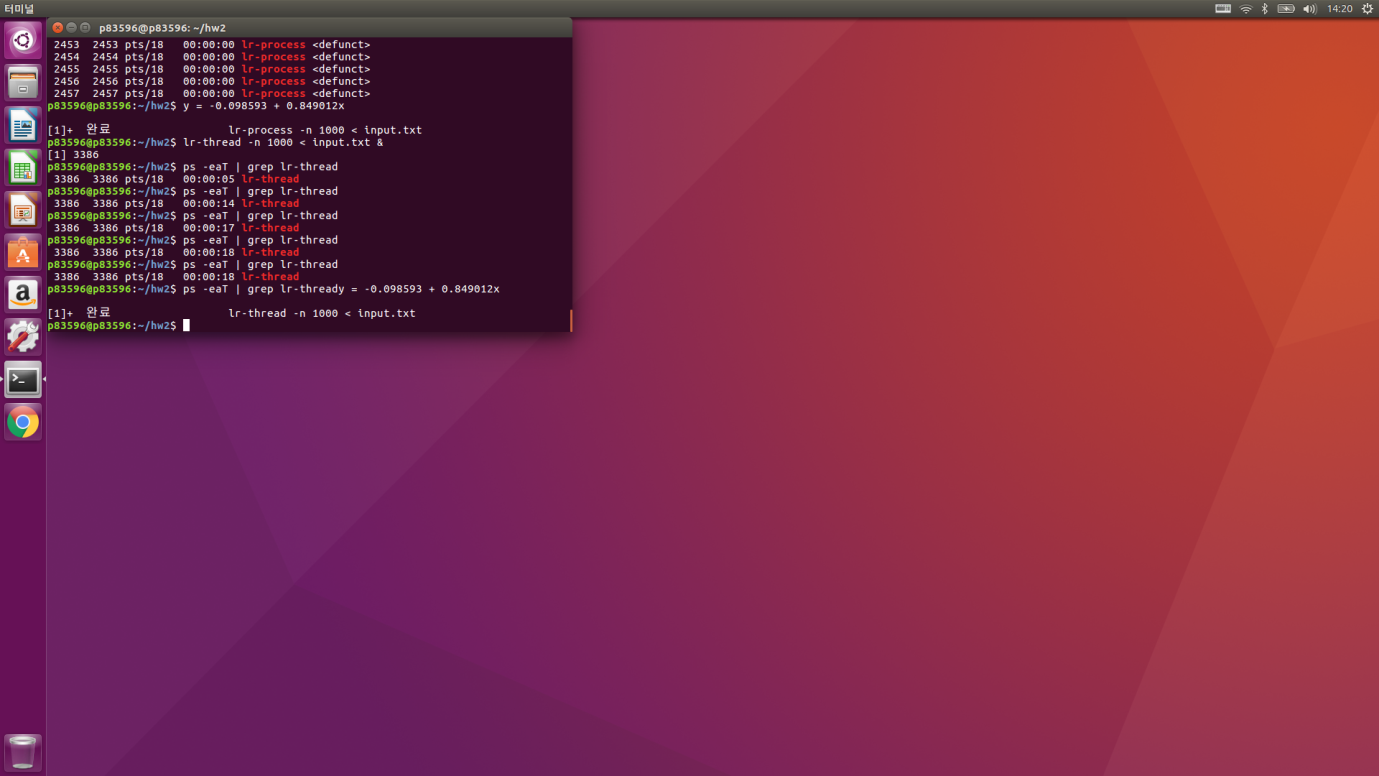
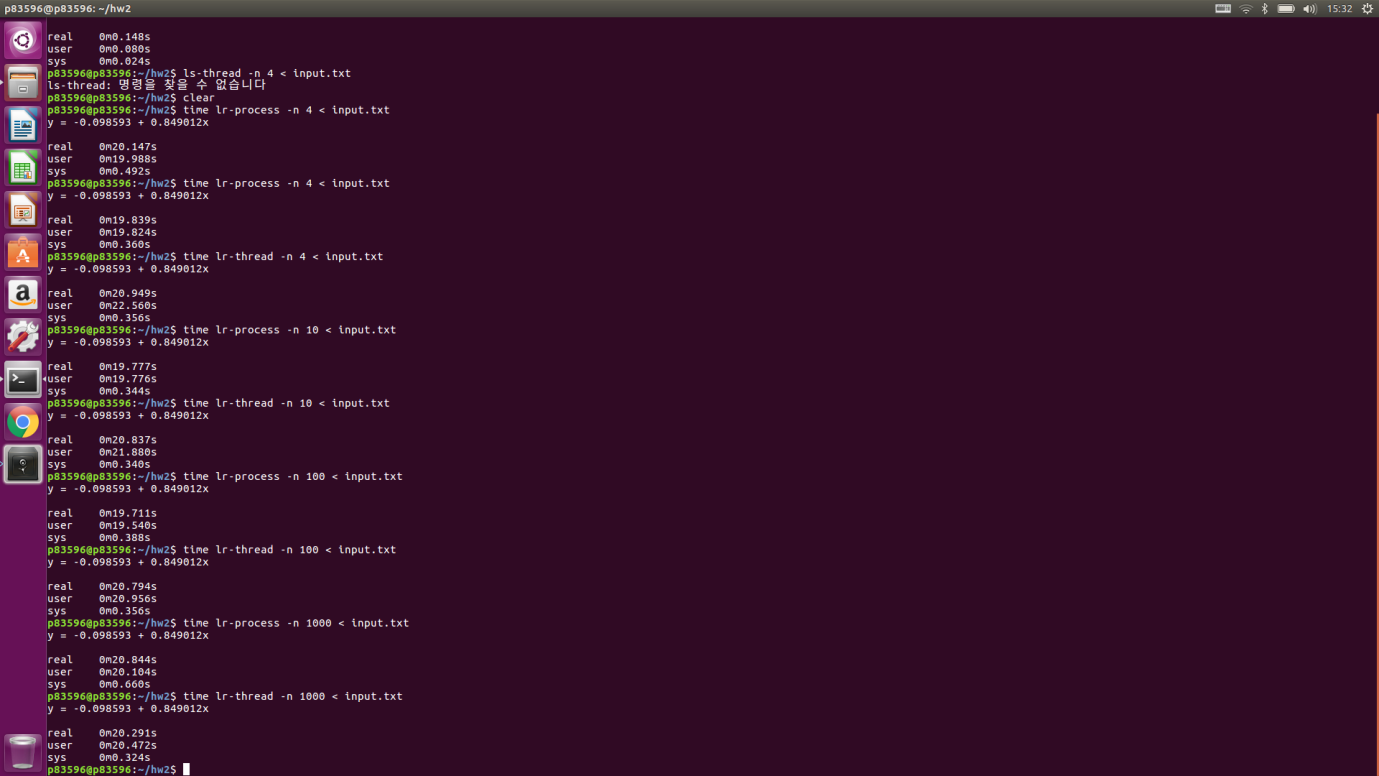
**2017년 1학기 운영체제 과제 #2  
사전조사 보고서**

**2015147543 정유정**

1. **쉘의 구조**  
   쉘(Shell)은 OS를 감싸는 껍데기와 같다. 사용자가 내린 명령어를 OS의 커널이 이해할 수 있는 형태로 번역해주는 command line interpreter를 쉘이라 한다. 예를 들어 사용자가 쉘에서 cat 명령어를 실행한다면 쉘은 fork() 함수를 통해 자식 프로세스를 만들고 자식 프로세스에서 /bin에 존재하는 cat 프로그램을 exec()함수를 통해 실행한다. 부모 프로세스인 쉘은 자식이 명령을 끝내고 죽기를 기다렸다가 다음 명령어를 기다리게 된다.
2. **명령어의 구현 방식**  
   **cd :** cd 명령어는 이전 디렉토리로 이동하는 명령어이다. 자식 프로세스는 부모 프로세스에 영향을 끼칠 수 없기 때문에 현재 쉘에서 이전 디렉토리로 이동하는 일은 fork() 함수를 이용하지 않고 현재 쉘에서 바로 수행한다. 현재 작업 디렉토리를 변경해 주는 chdir() 함수에 이동할 디렉토리 이름을 전달해주는 방식으로 구현한다.  
   **출력 리다이렉션 (stdout redirection) > :** dup2() 함수는 파일 디스크립터를 복사하는 함수이다. 파일 디스크립터 0은 Standard Input을, 1은 Standard Output을, 2는 Standard Error를 의미한다. 예를 들어 dup2(file.txt, 1) 이라고 하면 Standard Output을 file.txt로 내보낸다는 뜻이다. 이것을 이용해서 출력의 방향을 바꿀 수 있다.  
   **입력 리다이렉션 (stdin redirection)** **<:** 마찬가지로 dup2() 함수에 파일 디스크립터는 0을 사용한다.   
   **백그라운드 명령 & :** 명령어 맨 뒤에 &이 붙으면 쉘은 wait을 하지 않고 바로 다음 입력을 기다리도록 하면 된다.   
   **다른 명령어들 :** fork()로 자식 프로세스 생성 후 자식 프로세스에서 경로에 존재하는 파일을 실행시키는 execvp() 함수를 사용하면 된다.
3. **Process와 Thread의 차이**  
   프로세스는 프로세스간의 정보가 공유되지 않지만, 스레드는 하나의 프로세스 내에서 동작하기 때문에 정보가 공유된다. 멀티 프로세싱을 할 때는 프로세스를 스위칭 하는 과정이 필요하지만 스레드는 그 과정이 필요 없다. 그리고 새로운 프로세스를 생성하는 데 걸리는 시간이 새로운 스레드를 생성하는 데 걸리는 시간보다 길다. 그래서 멀티 프로세싱보다는 멀티 스레딩이 수행 시간이 더 짧게 된다.

**2017년 1학기 운영체제 과제 #2  
실습 과제 수행 보고서**

**2015147543 정유정**

1. **miniShell의 동작 과정**  
   쉘 프로세스가 종료되지 않도록 whil(1)로 무한반복문을 만들고 그 안에서 입출력이 진행되게 한다. 시:분:초는 time.h에 있는 tm 구조를 통해 값을 얻어내고 getenv(“USER”) 명령어로 현재 사용자명을 얻어낸다. get\_current\_dir\_name()으로 현재 디렉터리 경로를 얻어낸다. 그리고 fgets 함수를 이용해서 입력 문자열을 저장한다. 입력 문자열이 들어오면 먼저 strchr()을 사용해 쉘에서 미리 처리가 필요한 문자가 있는지 검사한다. cd명령어가 있으면 쉘 내에서 처리한다. >, <, & 명령어가 있으면 그 명령어를 strtok()함수로 분리해내고 나머지를 자식 프로세스가 처리해야 할 명령어로 전달한다. 전달받은 명령어는 strtok()함수를 이용해 공백을 기준으로 쪼개고 맨 앞에 명령어 부분을 execvp 함수를 실행해야 할 파일 이름으로 자식프로세스에 전달한다. 프로세스는 fork의 값이 0인지 아닌지로 구분하고 부모프로세스는 명령어가 백그라운드일 때 wait하지 않고 다음 입력을 받을 수 있도록 돌아가고, 백그라운드가 아닐 때는 wait한다.
2. **각 명령어를 구현한 방법**  
   **cd:** cd를 포함하는 입력 문자열의 개수를 세서 1개이면 cd만 입력된 것이므로 홈 디렉터리로 이동이므로 이 경우 chdir(getenv(“HOME”))으로 현재 디렉터리를 홈 디렉터리로 변경한다. 2개이면 해당 디렉터리로 이동하고, 3개 이상이면 잘못된 형식이므로 에러 메시지를 출력한다.  
   **>:** strtok()를 사용해서 > 기준 앞부분과 뒷부분으로 나눈다. 앞부분은 명령어이름, 뒷부분에는 출력할 파일 이름이 된다. 그 파일을 open하고 여기에 dup2(file,1)을 사용해서 출력 방향을 stdout에서 file로 변경한다.  
   **<:** strtok()를 사용해서 < 기준 앞부분과 뒷부분으로 나눈다. 앞부분은 명령어이름, 뒷부분은 입력 파일 이름이 될 것이다. 그 파일을 open하고 dupe2(file,0)을 사용해서 입력 방향을 stdin에서 file로 변경한다.  
   **&:** strchr()로 &를 찾으면 int background = 1로 바꾼다. 그리고 부모 프로세스에서 background == 1이면 background = 0으로 바꾸고 wait를 하지 않고 다음 반복문을 진행한다.   
   **기타 명령어:** execvp(명령어 문자열, 입력 문자열) 함수를 이용해서 명령어 파일을 실행한다. 경로는 디폴트로 /bin으로 설정되어 있다.
3. **Uname –a 실행 결과 화면과 개발 환경 명시**  
     
   Linux p83596 4.9.15-2015147543 #1 SMP Thu Mar 16 23:08:43 KST 2017 x86\_64 x86\_64 GNU/Linux
4. **결과 화면**  
     
   [시:분:초]username@현재경로$ (명령어) 형식이다. 여기에 cd 명령어를 입력하니 홈 디렉터리로 이동하였다. 또 ls 명령어를 ls.txt로 출력을 리다이렉션하고 cat 명령어를 통해 그 결과를 확인해 보았다.   
   lr-process를 백그라운드 명령어로 실행하고 lr-thread를 실행한 화면이다. 결과는 둘 다 y = -0.098593 + 0.849012x 가 나왔다. 입력 리다이렉션을 사용해 input.txt에 있는 데이터를 scanf()에 받도록 하였다. 위의 결과로 cd, ls, cat등 환경변수에 있는 명령어뿐만 아니라 입력 리다이렉션, 출력 리다이렉션, 백그라운드 명령어도 정상 작동되는 것을 알 수 있다.  
   프로세스와 스레드 별 수행시간은 각각 4개, 10개, 1000개에서 실행해보았는데 실행 시간이 다 비슷하게 나왔다. 아래는 실행한 화면이다.  
   lr-process –n 10: 18초  
     
   lr-thread –n 1000: 18초  
     
   예시로 든 사진은 process 10개일 때와 thread 1000개일 때이다. 둘 다 18초가 나왔고 프로세스나 스레드 수를 다르게 바꿔도 1초 정도 더 늘어나거나 하는 정도였는데, 1초는 사실 프로세스 수를 바꾸지 않고 똑같이 재실행해도 발생하는 정도의 오차라서 유의미한 차이로 보이지 않았다. 그래서 좀 더 정확한 실행 시간을 알아보기 위해 time 명령어를 사용해 보았다. 아래는 그 결과 화면과 표이다.  
   

|  |  |  |
| --- | --- | --- |
| **number** | **Process 실행시간** | **Thread 실행시간** |
| 4 | 19.839 sec | 20.949 sec |
| 10 | 19.777 sec | 20.837 sec |
| 100 | 19.711 sec | 20.794 sec |
| 1000 | 20.844 sec | 20.291 sec |

1000개보다 많은 프로세스, 스레드를 만들었을 때에는 메모리 부족 현상이 발생해서 그 이상으로는 실험해보지 못했다.

1. **결과 토의**  
   실험 결과를 보면, 프로세스의 수를 증가시켰을 때, 4개에서 100개까지는 실행시간이 0.1초가 줄어들었고, 1000개에서는 결과적으로 1초가 증가했다. 스레드 수를 증가시켰을 때, 4개에서 1000개까지 0.7초가 줄어들었다. 또 프로세스와 스레드 간 실행 시간은 스레드에서 약 1초가 더 길었다. 프로세스나 스레드는 생성하는데에도 시간이 걸리기 때문에 많이 만들면 만들수록 실행 시간이 늘어난다. 하지만 그만큼 한 프로세스나 스레드가 수행해야 할 연산량이 줄어들어서 실행 시간이 줄어든다. 실험 결과를 보면 개수를 4개에서 1000개로 증가시켜도 실행시간은 별로 차이가 나지 않았다. 이것은 프로세스 생성, 스레드 생성을 위해 늘어난 시간만큼 한 프로세스, 스레드가 처리해야 할 데이터 양의 감소로 발생한 실행 시간의 감소가 딱 맞물려서 겉으로 보기에는 실행 시간이 똑같이 나왔기 때문이다. 하지만 4개에서 1000개로 증가했을 때 프로세스에서는 실행 시간이 1초 늘어났는데 스레드에서는 0.7초가 감소한 것으로 보아서 스레드 996개 (1000개-4개)를 만드는 데 걸리는 시간이 프로세스 996개를 만드는 데 걸리는 시간보다 1.7초가 빠른 것으로 생각할 수 있다. 개수가 같을 때 스레드가 프로세스보다 연산 시간이 오래 걸리는 것은 계산 방식에서 차이가 났거나 아니면 아무래도 스레드는 한 프로세스의 자원을 공유하고, 프로세스는 독립적으로 자원을 보유하고 있기 때문에 스레드의 계산 성능이 프로세스의 계산 성능보다 떨어졌을 가능성도 있다고 생각한다.