**오퍼레이팅 시스템 4주차**

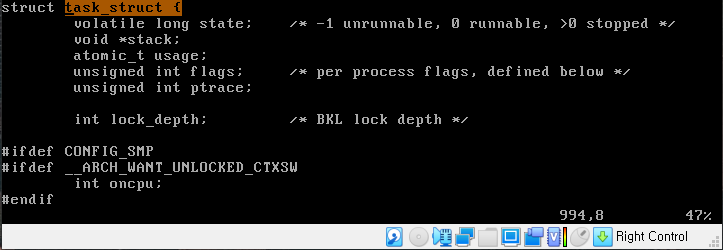
**121800626 성시열**

3.1) task\_struct is defined in include/linux/sched.h (search for "task\_struct {"). Which fields of the task\_struct contain information for process id, parent process id, user id, process status, the memory location of the process, the files opened, the priority of the process, program name?

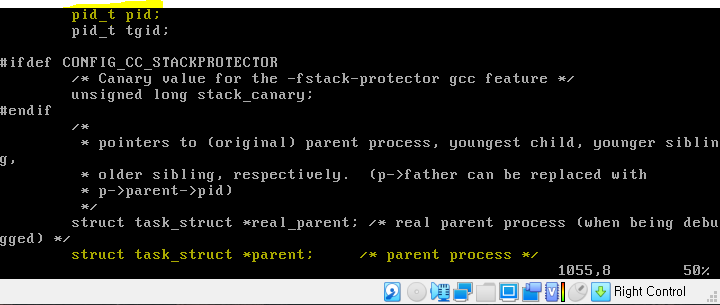




해당 위치로 가서 sched.h를 열어보았다.



task\_struct { 를 찾아본 결과



Process id가 pid\_t pid에 저장되어 있고, parent process id는 struct task\_struct \*parent에저장되어 있다.



User id는 uid에 저장되어 있다.



Process status는 volatile long state에 저장되어 있다.



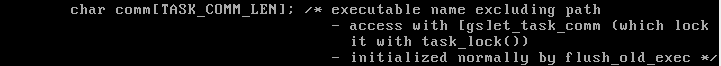
The memory location of the process는 struct mm\_struct \*mm에 저장되어있다.



the files opened은 struct fs\_struct \*fs에 저장되어있다.

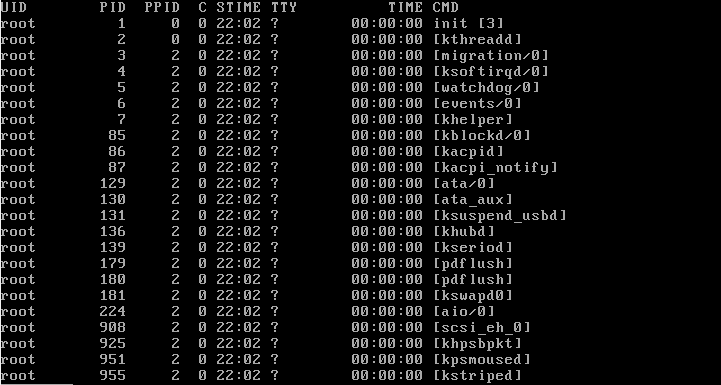


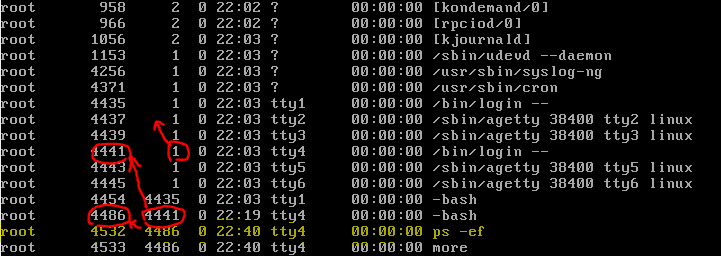
the priority of the process은 unsigned int rt\_priority에 저장되어있다.



program name은 char comm[TASK\_COMM\_LEN]에 저장되어있다.

3.2) Display all processes with "ps –ef". Find the pid of "ps -ef", the process you have just executed. Find the pid and program name of the parent process of it, then the parent of this parent, and so on, until you see the init\_task whose process ID is 0.



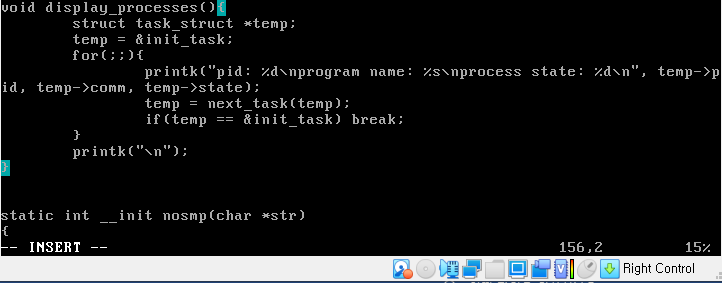


linux-2.6.25.10 위치에서 프로세스 전체의 정보를 파악해보았다. 최상단을 보면 알 수 있듯이 User ID, Parent ID, Parent Process ID 등을 파악할 수 있다. ps -ef의 pid는 4532이다. ps -ef의 PPID를 통해 따라 올라가보았다. Ps -ef 🡪 -bash 🡪 /bin/login -- 🡪 init[3] 🡪 PID 0 임을 알 수 있다.

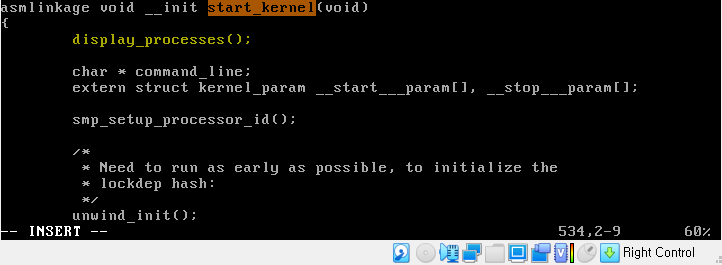
3.3) Define display\_processes() in init/main.c (right before the first function definition). Call this function in the beginning of start\_kernel(). Confirm that there is only one process in the beginning. Find the location where the number of processes becomes 2. Find the location where the number of processes is the greatest. Use "dmesg" to see the result of /().



해당 위치로 이동하여 main.c를 열었다.



첫 번째 함수 정의인 nosmp 이전에 display\_processes()를 정의한다. 코드는 강의자료 process queue를 참고하였다. 포인터형 struct인 temp를 선언하고 이에 init\_task를 넣습니다. 이 때 init\_task는 커널의 process descriptor 입니다. 이후 temp에서 pid, 프로그램 이름, 프로세스 상태 등의 정보를 반복해서 출력해주고, temp의 주소와 init\_task의 주소가 같아지는, 즉 모든 프로세스를 다 출력하였을 때 빠져나옵니다.



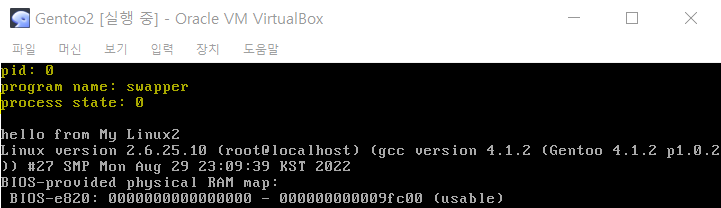
이후 start\_kernel 의 시작과 동시에 display\_processes 함수를 호출한다.



홈 디렉토리로 이동한 뒤 변경된 부분을 recompile & reboot를 진행한다.

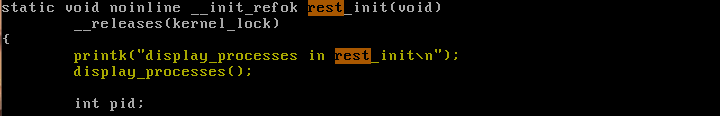


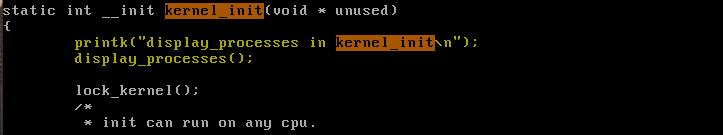
재부팅 한 뒤 위를 통해 부팅 메시지를 확인한다.



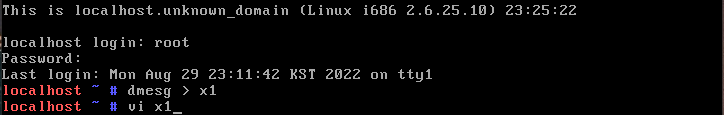
부팅 메시지 상단에 pid, program name, process state가 출력됨을 확인하였다.



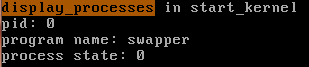
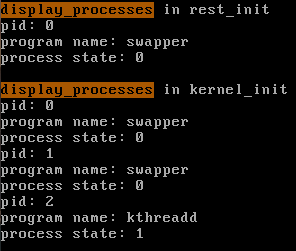




Process number가 2가 되는 곳을 찾기 위해 부팅 시 호출되는 다양함 함수들에 display\_processes를 추가하였다. 이후 recompile & reboot를 진행하였다.

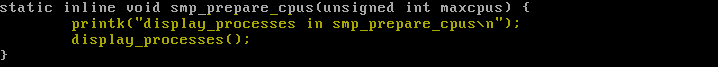


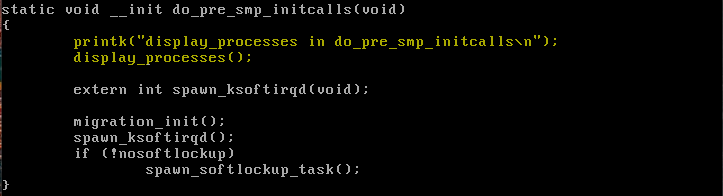
이후 부팅 메시지를 확인해보았다.

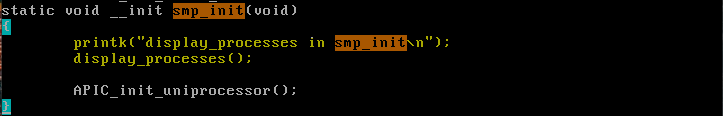
 

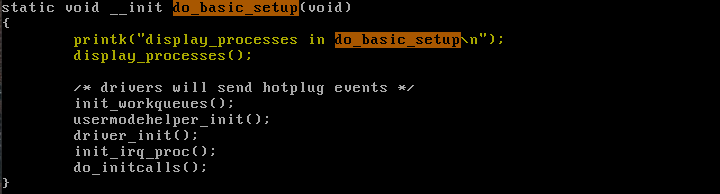
확인해본 결과 kernel\_init에서 pid가 0에서 2까지 변화되므로 프로세스의 개수가 2개임을 확인하였습니다.

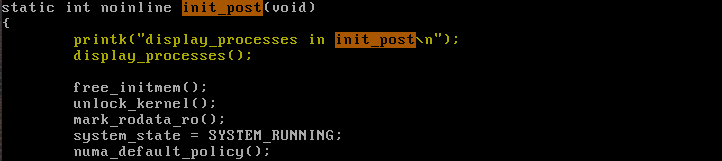
가장 많은 프로세스의 개수를 파악하기 위해 kernel\_init이 호출하는 함수 Lock\_kernel, Smp\_prepare\_cpus, Do\_pre\_smp\_initcalls, Smp\_init, Sched\_init\_smp, Cpuset\_init\_smp, Do\_basic\_setup, Init\_post 중에서 main.c에 존재하는 함수 에 display\_processes를 호출하였다.



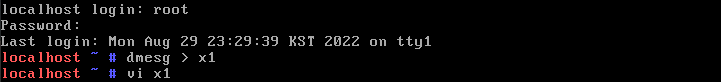




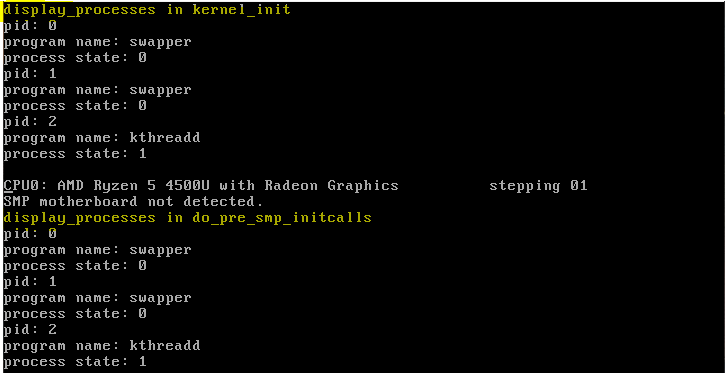


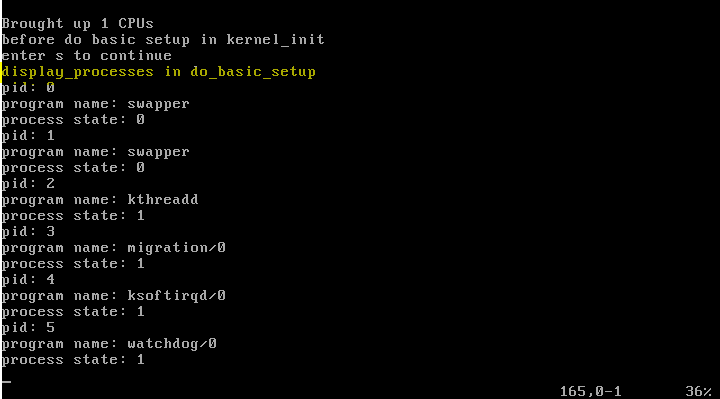


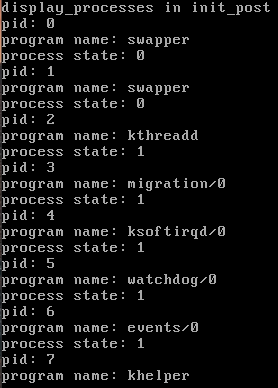
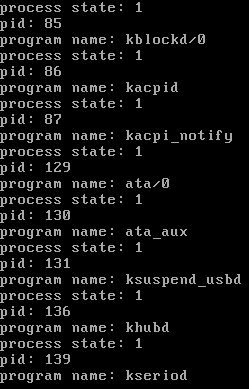
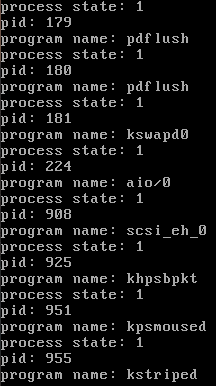
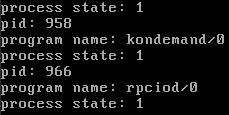
Recompile and reboot 한 뒤



부팅 메시지를 확인해보았다.





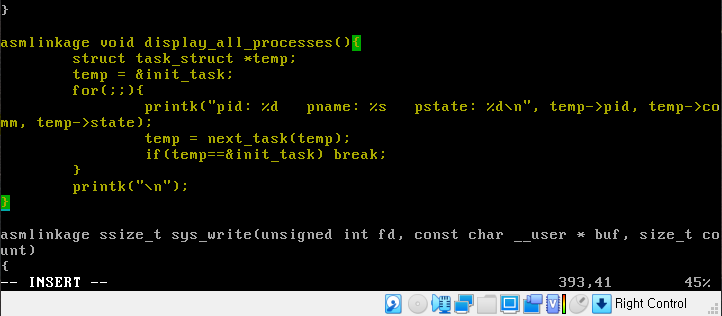
   

확인해본 결과 do\_basic\_setup 이 6개이고, init\_post는 무수히 많음을 알 수 있다. 이를 통해 init\_post가 가장 많은 프로세스를 호출하는 것을 알 수 있다.

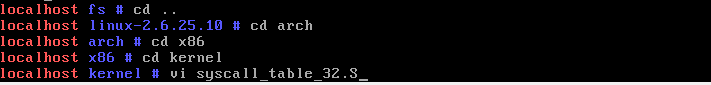
3.4) Make a system call that, when called, displays all processes in the system. Run an application program that calls this system call and see if this program displays all processes in the system.



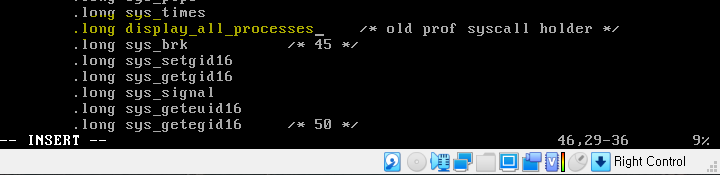
System call 할때마다 이므로 read\_write.c에서 함수를 추가한다.



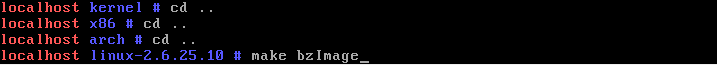
이 전에 main.c에서 추가했던 display\_processes와 동일하게 하되, 함수 이름을 바꿔주었고, 많은 프로세스가 출력될 것으로 예상되어, 출력문 형태를 한 줄에 모두 출력하도록 바꿔주었다.



해당 위치로 가 syscall\_table의 빈 번호를 찾는다.



빈 번호(44번)를 찾아 다음과 같이 바꿔주었다.



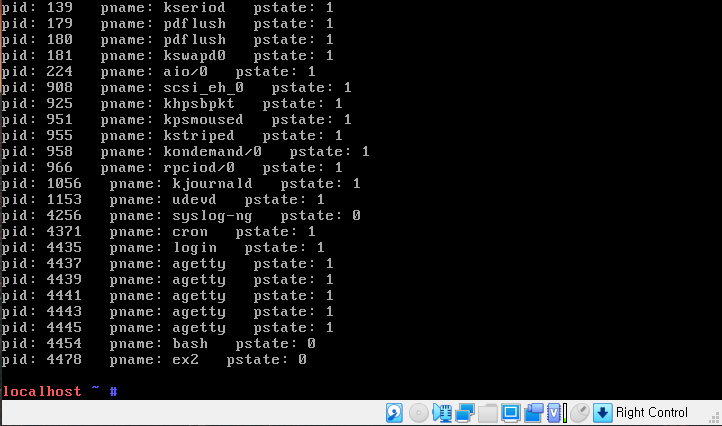
저장 후 나와서 recompile & reboot를 진행한다.



44번을 호출하는 ex2.c를 생성하였다.

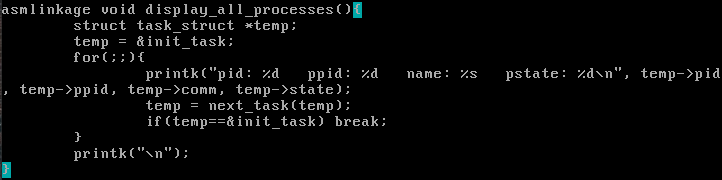


컴파일 후 log level을 낮춰주고 실행하였다.



실행해본 결과 위와 같은 프로세스들이 출력됨을 알 수 있다.

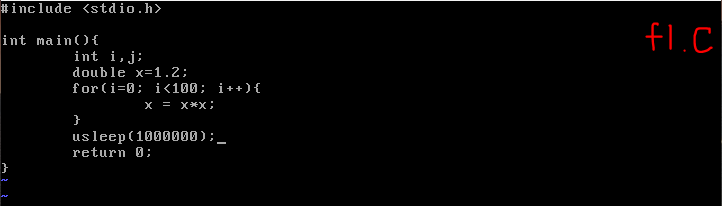
3.4.1) Make a system call that, when called, displays all ancestor processes of the calling process in the system. For /example, if ex1 calls this system call, you should see: ex1, ex1’parent, ex1’s parent’s parent, etc. until you reach pid=0 which is Linux itself.

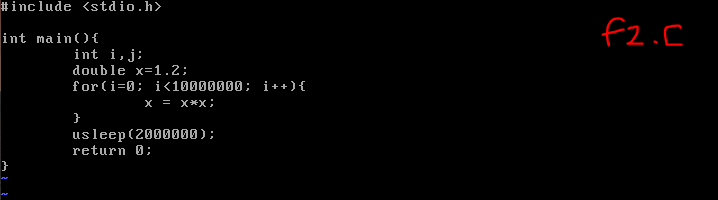


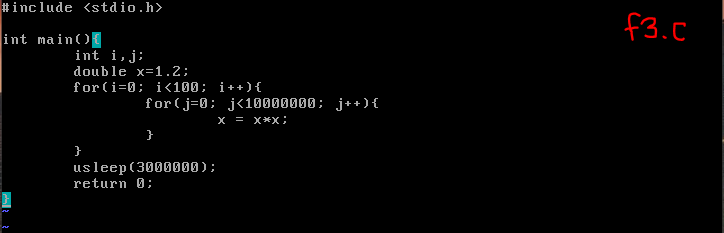


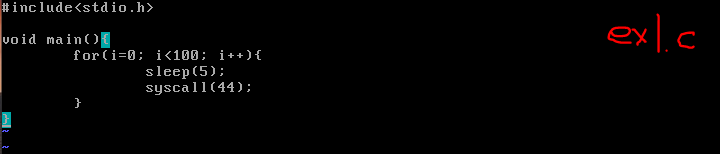
ppid를 출력해보려 하였으나 init\_task.h를 확인해본 결과 존재하지 않는 변수여서 출력하지 못하였다.

3.5) Run three user programs, f1, f2, and f3, and run another program that calls the above system call as follows. State 0 means runnable and 1 means blocked. Observe the state changes in f1, f2, f3 and explain what these changes mean.







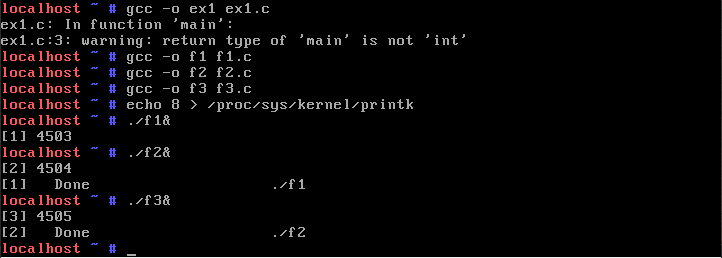


#./f1&

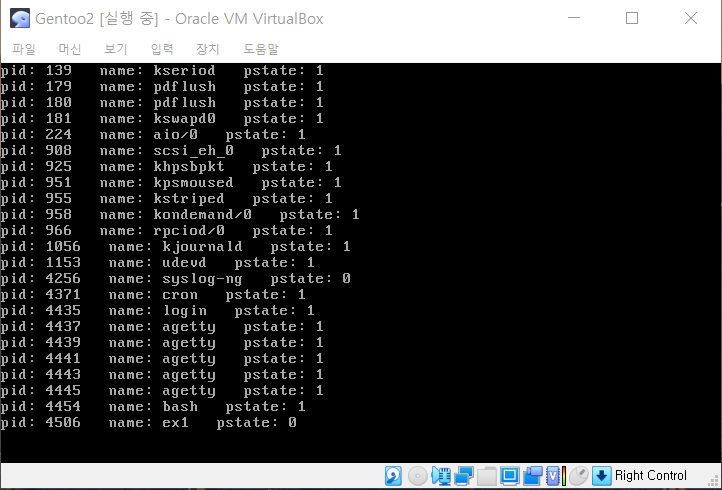
#./f2&

#./f3&

#./ex1



문제에 주어진대로 모든 파일을 작성하고 컴파일 하였다. 문제에서 주어진대로 파일을 실행한 결과 다음과 같이 출력되었다. 각 실행파일 별 pid가 4503, 4504, 4505임을 알 수 있고, 다음 파일을 실행할 때마다 이전 파일이 종료됨을 확인하였다.



ex1.c를 실행시켜보았다. 가장 마지막 줄에 ex1이 실행되고 있음을 확인하였고, pid가 4506임을 알 수 있다. 프로세스 상태가 0임을 알 수 있는데 이는 runnable 함을 의미한다. f1, f2, f3는 위에서 볼 수 있듯이 다음 문장이 실행되면서 done 되므로 프로세스 실행이 중지되었다는 것으로 추측한다.

3.6) Modify your display\_processes() so that it can also display the remaining time slice of each process (current->rt.time\_slice) and repeat 3.5) as below to see the effect. "chrt -rr 30 ./f1" will run f1 with priority value = max\_priority-30 (lower priority means higher priority). "-rr" is to set scheduling policy to SCHED\_RR (whose max\_priority is 99).

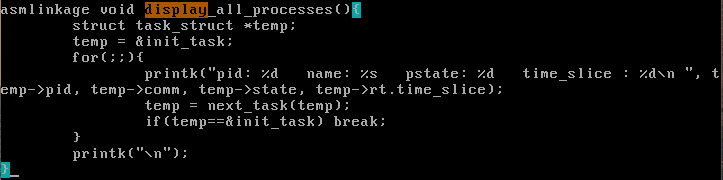
#chrt –rr 30 ./f1&

#chrt -rr 30 ./f2&

#chrt -rr 30 ./f3&

#chrt -rr 30 ./ex1

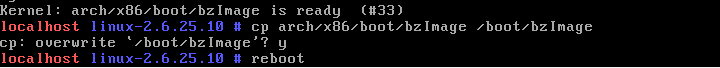
display\_all\_processes()로 이동하였다.

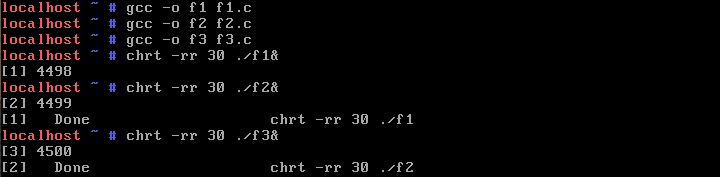


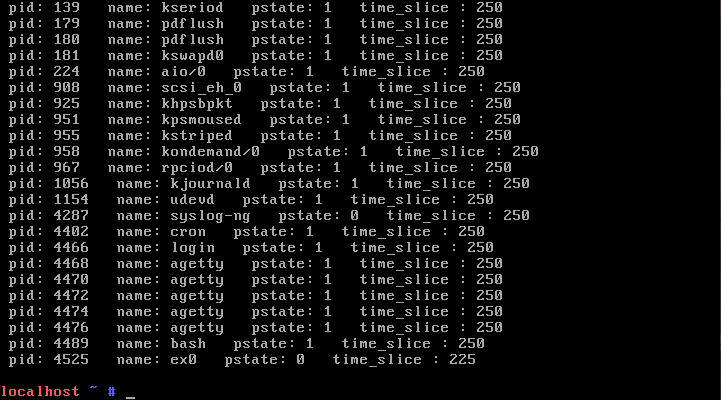
출력해주는 printk 부분에서 temp->rt.time\_slice 를 추가하였다.



수정 후 recomplie을 해보았다.







chrt 명령어는 프로세스의 우선순위를 변경하는 명령어이다.

-rr 30 옵션은 실시간 스케줄링 정책을 SCHED\_RR이라는 정책으로 설정하는 것이다.

SCHED\_RR : 우선순위가 같을 때 태스크를 목록 어디에 삽입할 지를 결정하는데 사용한다.