3 차 종합평가 답안지

작성자: 이상훈

1, 2 은 각자 느낀점을 적었을 것이라 판단함

3. 파일

4. User 0 ~ 3 GB, Kernel 3 ~ 4 GB

5. User 0 ~ 2^63 byte Kernel 2^63 ~ 2^64 byte

6. ELF(Executable Linkable Format)

7. ls, chmod, sudo, gcc, gdb, vi, wget, mkdir, mkfifo, cp, mv, ps, kill, cat 등등

8. task_struct 구조체를 생성

9. 현재 실행중인 Task 에 대한 포인터

10. uname -a

11. Task 가 동작하다 주어진 Time Slice 가 만료되어 제어권을 양도하기 위해
    Context Switching 등이 발생했을 경우 다른 Task 가 Register 를 변경할 수 있으므로
    자신이 현재 어떠한 작업들을 진행했는지를 기록할 필요가 있다.
    이중에서 레지스터등의 정보를 저장하기 위한 용도이다.

12. wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.9.1.tar.gz
    tar zxvf linux-4.9.1.tar.gz

13. kill -l

14. O(N)은 데이터의 양이 많아지면 많아질수록 알고리즘의 성능이 떨어지는데 비해
    O(1)은 데이터의 양이 많아지더라도 알고리즘의 성능이 언제나 동일하다.

15. mv orih.c orig.c

16. ls -aR

17. 세마포어란 여러 태스크가 동시다발적으로 특정한 공유 자원에 접근하는 것을 방지하기 위해 사용한다. 여기서 특정 공유 자원이 되는 영역을 Critical Section 이라 한다. 특정한 작업을 반드시 보장해줘야 하는 경우에 적용하며 그렇지 않은 경우에는 오히려 성능의 저하를 유발시킬 수 있다.

18. Static Priority : 1 ~ 99
    Dynamic Priority : 100 ~ 139

19. chmod

20. 페이지 프레임, 4 KB

21. 첨부 파일 참고

22. task_struct 구조체에 있는 mm_struct 내에 start_code, end_code 등으로 기록됨

23. C언어를 사용하기 위해서는 반드시 Stack이 필요하다.
    Kernel 영역에서도 동작하는 코드가 올라가기 위한 Text 영역
    전역 변수가 있는 Data 영역, 동적 할당하는 Heap, 지역 변수를 사용하는 Stack이 존재한다.
    이는 역시 User 영역에서도 동일하므로 양쪽에 모두 메모리 공간이 구성된다.

24. task_struct 구조체 내부에 mm_struct 구조체 안에 보면 pgd 라는 필드가 있다.
    Page Directory 를 의미하는 것으로 pgd -> pte -> page 로 3단계 Paging 을 수행한다.
    각각 10bit, 10bit, 12bit로 VM 의 주소를 쪼개서 Indexing 을 한다.

25. task_struct 구조체 내부에 있는 files_struct 내부의 file 로 간다.
    가보면 file_operations 구조체가 있고 여기에 open, read, write, release 등이 존재한다.
    이들은 파일 시스템에 따라 적절한 동작을 수행할 수 있도록 함수 포인터로 만들어져 있다.
    즉 EXT2 파일 시스템이면 ext2_open, ext2_read, ext2_write 등이 자동으로 동작을 하며
    EXT4 인 경우엔 ext4_open, ext4_read, ext4_write 가 동작하게 된다.
    예로 NTFS 일지라도 이것은 변함이 없으며 EXT4 에서 NTFS 로 복사를 한다 가정하면
    ext4_open, ntfs_open, ext4_read, ntfs_write, ext4_release, ntfs_release 가 동작하게 된다.

26. 내부 인터럽트와 외부 인터럽트로 나뉜다.
    내부 인터럽트는 CPU 내에서 일어나는 인터럽트에 해당하며
    외부 인터럽트는 CPU 외부의 실제 Device가 발생시키는 인터럽트다.
    여기서 예외적으로 존재하는 것이 SW 인터럽트인 System Call 에 해당한다.

27. task_struct 내의 mm_struct 내의 pgd 가 해당 정보를 가지고 있다.

28. x86 – CR3, ARM – CP15

29. CPU 마다 RQ, WQ 가 1개씩 존재한다.
    active 배열과 expired 배열이 존재해서
    어떤 우선순위의 Process 가 현재 올라가 있는지 bitmap 을 체크하게되며
    queue 에는 만들어진 task_struct 가 들어가 있게 된다.
    즉, bitmap 을 보고 우선순위에 해당하는 것이 존재하면
    빠르게 queue[번호] 로 접근해서 해당 task_struct 를 RQ 에 넣거나
    주어진 Time Slice 가 다했는데 수행할 작업이 남아 있다면
    RQ 에서 WQ 로 집어넣는등의 작업을 수행한다.
    결국 Scheduling 이란 작업이 Multi-Tasking 을 지원하는데 있어 핵심인 기술이다.
    여러 Task들을 동시다발적으로 동작하는것처럼 보이게하는 트릭이라 할 수 있겠다.
    여기서 RQ 에서 WQ 로 들어갈 때 Context Switching 이 발생하여 HW Context 를 저장한다.
    이 정보는 task_struct 에 있는 thread_info 에 저장되고
    task_struct *current 에 의해 자기 자신의 정보를 언제든 찾을 수 있게 구성되어 있다.

30. 파일

31. System Call은 User가 Kernel에 요청하여 작업을 처리할 수 있도록 지원한다.
    내부적으로 굉장히 복잡한 과정을 거치지만 User가 read()등만 호출하면
    실질적인 복잡한 모든 과정은 Kernel이 수행하게 되어 편의성을 제공해준다.
    또한 앞서 살펴봤듯이 VFS 에 의해 알아서 파일 시스템에 대한 부분도 고려해준다.

32. Process 는 자신의 고유한 공간으로 가상의 4 GB 를 가지고 있다.
    실제 이 공간을 모두 사용하지 않으며 Demand Paging 에 따라 필요한 경우에만
    실제 물리 메모리 공간을 Paging Mechanism 을 통해 할당받아 사용한다.
    또한 실제 mm_struct 내에 있는 28 바이트가 가상 메모리의 실체에 해당한다.
    메모리가 부족하면 swap 으로도 들어가기 때문에
    대규모 데이터를 메모리에 올려서 처리할 경우
    이와 같은 구조로 구성한다면 문제 없이 메모리를 재활용할 수 있다.

33. 첨부 파일 참고

34. 첨부 파일 참고

35. set follow-fork-mode child

36. Blocking 연산의 경우 CPU를 계속 잡고 있으므로 다른 일을 수행할 수 없다.
    Non-Blocking 연산의 경우 CPU를 잡고 있지 않고 이런 일을 할꺼니까 준비되면 알아서 해줘
    라는 식으로 처리를 하기 때문에 Blocking 연산에 비해 성능면에서 뛰어난다.
    그러나 반드시 Blocking 연산이 필요한 경우 또한 존재한다.
    특정 연산이 보장되어야만 하는 경우에는 Blocking 처리를 해야 한다.

37. 첨부 파일 참고

38. 첨부 파일 참고

39. 첨부 파일 참고

40. 첨부 파일 참고

41. 첨부 파일 참고

42. 리눅스 커널에 존재하는 Task 를 나타내는 구조체인
    task_struct 내에 files_struct 내에 file 구조체에 대한 포인터가 fd_array 다.
    거기서 우리가 System Programming 에서 얻는 fd 는 바로 이 fd_array 에 대한 index 다.

43. buf.st_mode에는 리눅스 커널 inode의 i_mode와 같은 값이 들어가 있다.
    파일의 종류 4비트와 setuid, setgid, sticky bit, 그리고 rwx가 3개씩 존재한다.

44. 첨부 파일 참고

45. 첨부 파일 참고

46. Linux 에서 사용할 수 있는 Windows 에서의 DLL 과 같은 녀석이다.
    동적 링크 라이브러리라고 하여 실행 시간에 필요한 라이브러리를 가져다 붙인다.
    고속으로 처리하기 위해서는 Dynamic 방식보다는 Static 방식이 훨씬 성능이 좋다.
    그러나 스토리지 측면에서 봤을 경우에는 Dynamic 으로 활용하는 것도 고려해야할 부분이다.
    만드는 실행 파일 내부에서 공통적으로 같은 것들이 활용된다면
    같은 코드가 여러 실행 파일에 들어가면서 무거워지기 때문이다.

47. gcc -L/$(pwd) -Wl,-rpath=$(pwd) -Wall -o 실행파일명 소스파일명 -l링크할라이브러리명

48. 첨부 파일 참고

49. 첨부 파일 참고

50. CPU(HW) 의존적인 코드가 위치한 영역이다.
    각종 CPU 아키텍처별 코드가 모아져 있다.

51. ARM 은 하위 호환이 안되고 다양한 반도체 벤더들이 개발을 하고 있기 때문에
    해당 디렉토리에 들어가면 회사별 주요 제품들의 이름이 보이는 것을 확인할 수 있다.

52. 시스템 프로그래밍 7 회차 수업 내용 참고

53. 시스템 프로그래밍 9 회차 수업 내용 참고

54. 디렉토리에 sticky 비트가 붙은 경우엔 공유 폴더로서 동작한다.
    파일에 sticky 비트가 붙은 경우엔 해당 파일을 Paging 할 때 swap 과 함께 작업한다.

55. 회로도 상에서 Reference 전압과 Feedback 전압의 차이가
거의 없다는 전제하에 출력 전압은 안정이 될 것이다.
여기서 Vref 와 Vfeed 의 차이가 크게 난다면 효율이 떨어질 것이다.

56. DC-DC 컨버터의 스위치가 ON 되면 코일을 충전하면서 폐회로가 구성된다.
스위치가 OFF 되면 코일과 다이오드에 의해 다시 폐회로가 구성되어 동작하게 된다.

57.

$$V_s = i_L R + L\frac{di_L}{dt} \Rightarrow i_L R + L i_L'$$
$$i_L' + \frac{R}{L} i_L = \frac{V_s}{L}$$
$$\mu = e^{\frac{R}{L}t}$$
$$e^{\frac{R}{L}t} i_L' + \frac{R}{L} i_L e^{\frac{R}{L}t} = \frac{V_s}{L} e^{\frac{R}{L}t}$$
$$\frac{d}{dt}\left(i_L e^{\frac{R}{L}t}\right) = \frac{V_s}{L} e^{\frac{R}{L}t}$$
$$\int \frac{d}{dt}\left(i_L e^{\frac{R}{L}t}\right) = \int \frac{V_s}{L} e^{\frac{R}{L}t}$$
$$i_L e^{\frac{R}{L}t} = \frac{V_S}{R} e^{\frac{R}{L}t} + C$$
$$i_L = \frac{V_S}{R} + C e^{-\frac{R}{L}t}$$
$$i_L = \frac{V_S}{R} - \frac{V_S}{R} e^{-\frac{R}{L}t} (\because t(0) = i_L = 0)$$

58.

$$0 = i_L R + L\frac{di_L}{dt} \Rightarrow i_L R + L i_L'$$
$$i_L' + \frac{R}{L} i_L = 0$$
$$\mu = e^{\frac{R}{L}t}$$
$$e^{\frac{R}{L}t} i_L' + \frac{R}{L} i_L e^{\frac{R}{L}t} = 0$$
$$\frac{d}{dt}\left(i_L e^{\frac{R}{L}t}\right) = 0$$
$$\int \frac{d}{dt}\left(i_L e^{\frac{R}{L}t}\right) = \int 0$$
$$i_L e^{\frac{R}{L}t} = C$$
$$i_L = C e^{-\frac{R}{L}t}$$
$$i_L = \frac{V_S}{R} e^{-\frac{R}{L}t}\left(\because t(0) = i_L = \frac{V_S}{R}\right)$$

59. 인덕터는 전기적 관성의 특성을 가지고 있다.

60. ESR, ESC, ESI 는 기생 저항, 기생 커패시턴스, 기생 인덕턴스에 해당한다.

61. 수업시간에 배웠던 Altium Designer 로 PCB Library 를 만든다.

62. 라플라스 변환 결과에 s = jw 를 대입하면 임피던스가 된다.

63. 수업 시간에 Kp(비례 계수) 와 Ki(적분 계수)를 모두 구했으며
    임피던스를 기반으로 전달함수를 구해 쉽게 해석 했었는데 당시의 기억을 상기하길 바람.
    (노트 필기가 있다면 해당 부분을 참고하도록 한다)