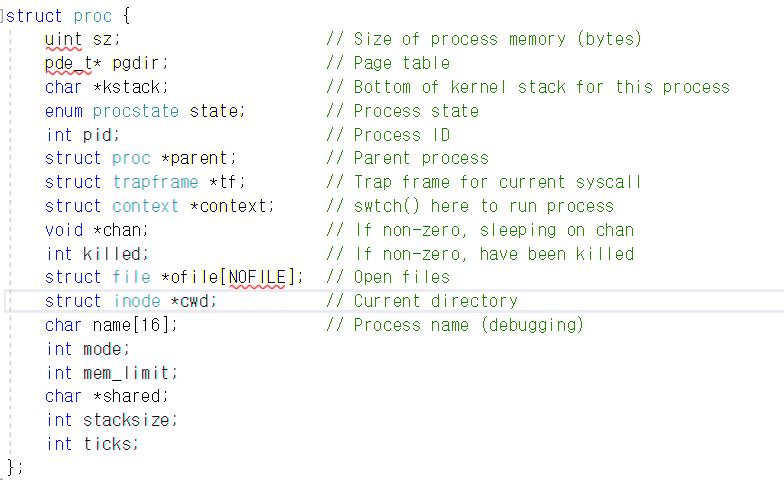
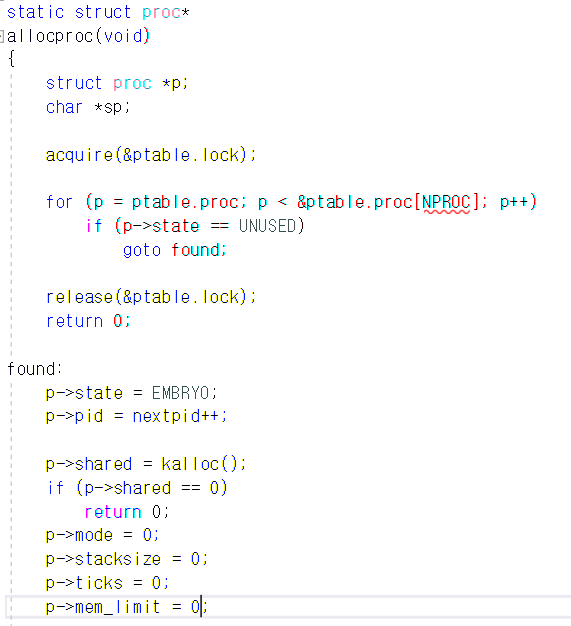
Project 2

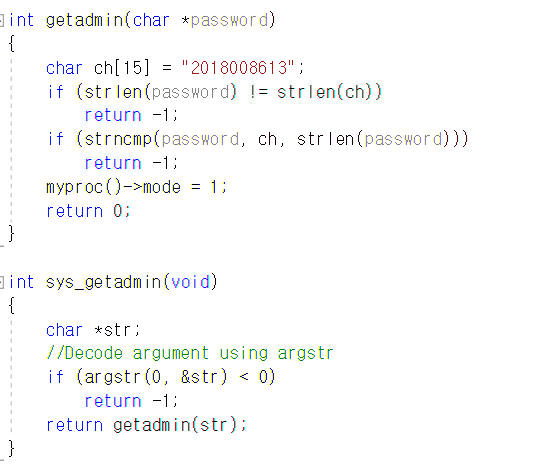
1. Administrator Mode



먼저 프로세스의 실행 모드를 구분해주기 위해 proc struct에 mode라는 변수를 추가해주었습니다. Mode가 0이면 user mode고, 1이면 administrator mode로 구분했습니다.

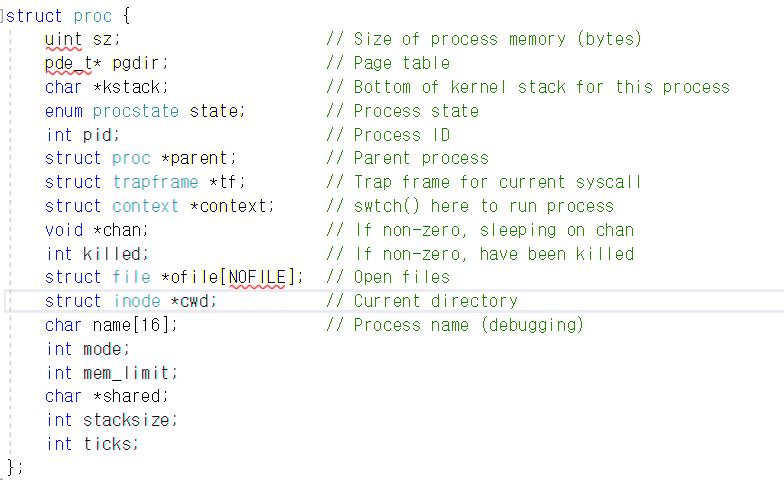


그리고 allocproc에서 mode를 0으로 초기화시켜서 user mode로 만들어주었습니다.

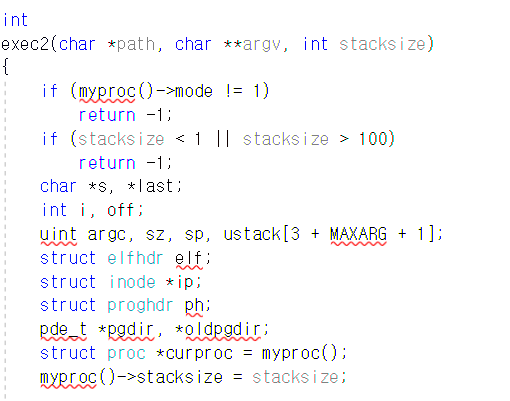


그리고 proc.c에 getadmin, sys\_getadmin 함수를 추가해 시스템 콜을 만들어 주었습니다. Password가 학번인 2018008613과 일치하면 0을 myproc()->mode를 1로 바꾸어 administrator mode로 바꾼 뒤 0을 return해주고, 일치하지 않다면 -1을 return 해주었습니다.

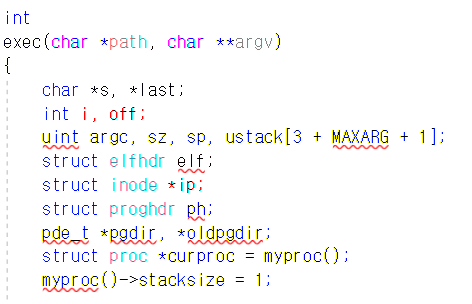
1. Custom stack size



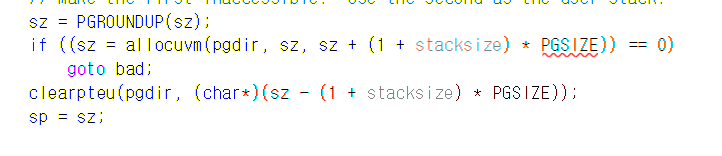
먼저 proc struct에 stacksize를 추가해주었습니다.



그리고 exec.c에 exec2 함수를 만들어주었습니다. exec함수와 유사한데, 먼저 myproc()->mode가 1인지를 확인함으로써 관리자 권한을 얻었는지 확인하고 그렇지 않다면 -1을 return 해주었습니다. 그리고 stacksize가 1이상 100이하의 값이 아닐 때에도 -1을 return 해주었습니다. 그리고 myproc()->stacksize에 stacksize를 넣어주었습니다.

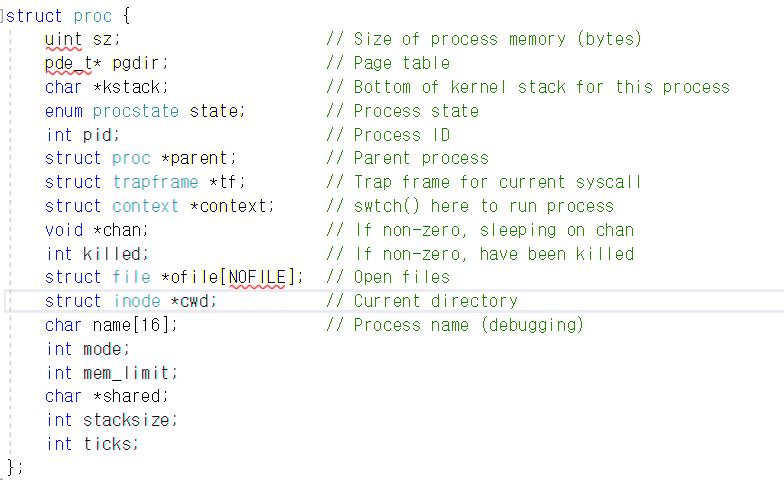


Exec 함수에서도 myproc()->stacksize에 1을 넣어주었습니다.

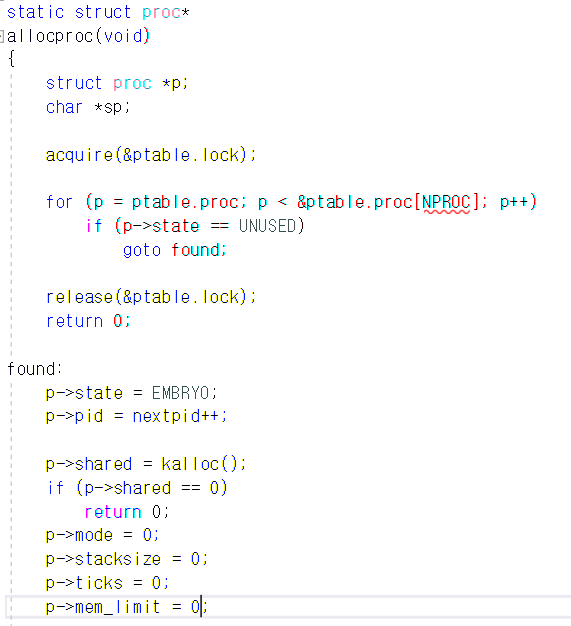


그리고 exec2 함수에서 (1+stacksize) \* PGSIZE만큼을 할당해주어서 stacksize 만큼의 스택용 페이지와 1개의 가드 페이지를 할당해주었습니다.

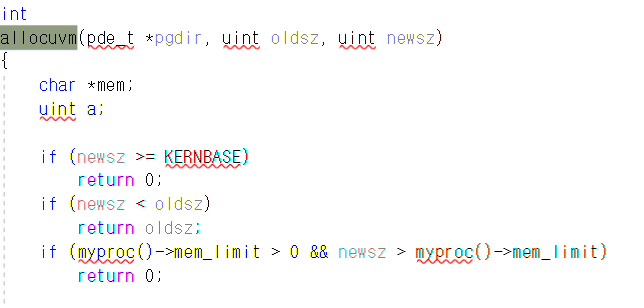
1. Memory limit



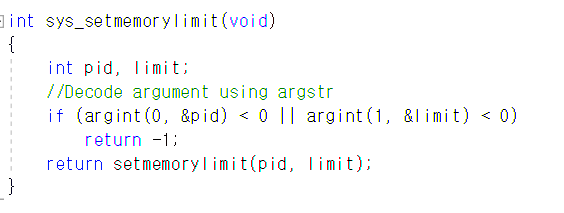
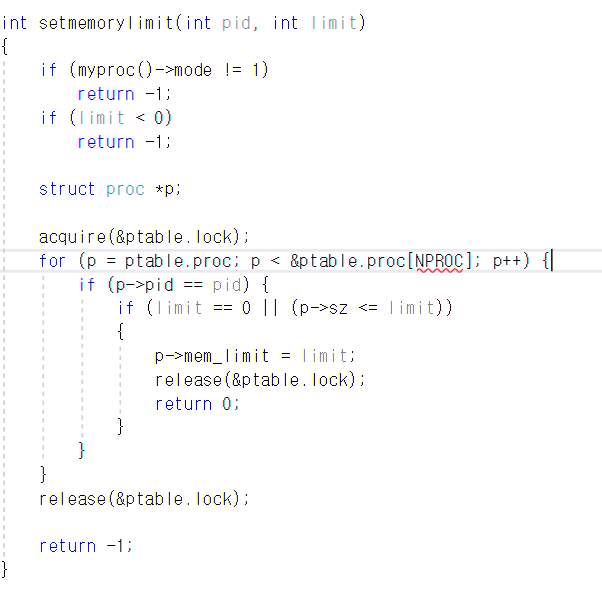
먼저 proc struct에 mem\_limit 항목을 추가해주었습니다. 이 값으로 메모리의 최대치를 지정할 수 있게 해줍니다.



그리고 allocproc에서 mem\_limit 값을 0으로 설정해주어서 초기 생성 시 메모리 제한이 없도록 해줍니다.

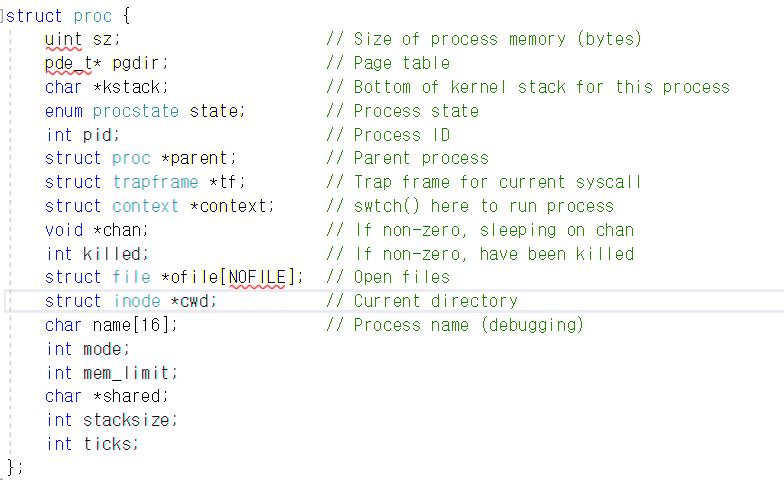


그리고 allocuvm에서 새로운 페이지를 할당할 때, newsz가 myproc()->mem\_limit을 넘지 않는지 체크해주고, 만약 넘는다면, 0을 return해서 변경하지 않도록 합니다.

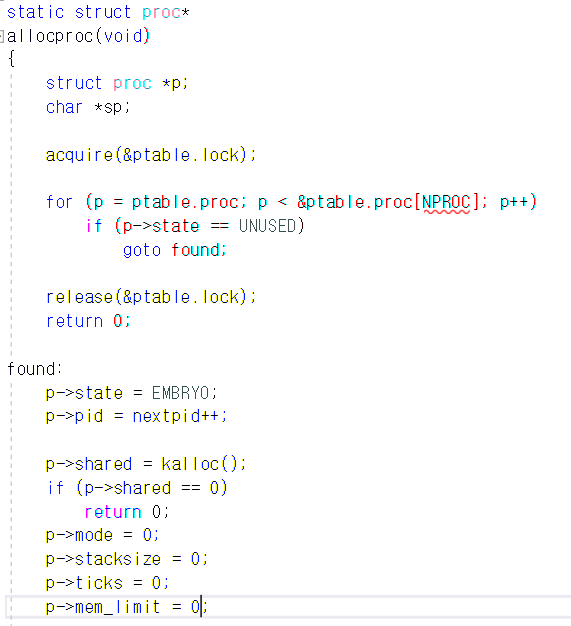


그리고 setmemorylimit, sys\_setmemorylimit 함수를 proc.c에 만들어서 시스템 콜을 구현했습니다. 먼저 myproc()->mode가 1인지를 체크함으로써 관리자 권한을 얻었는지 확인하고, 만약 user mode라면 -1을 return해줍니다. 그리고 limit 값이 음수인 경우도 -1을 return 해줍니다. 그리고 ptable을 돌면서 해당 pid의 프로세스가 있고, p->sz보다 limit가 크다면, p->mem\_limit 값을 limit으로 지정해주고, 0을 return해줍니다. 만약, 해당하는 pid의 프로세스를 찾지 못했거나, p->sz가 limit보다 크다면 -1을 return 해줍니다.

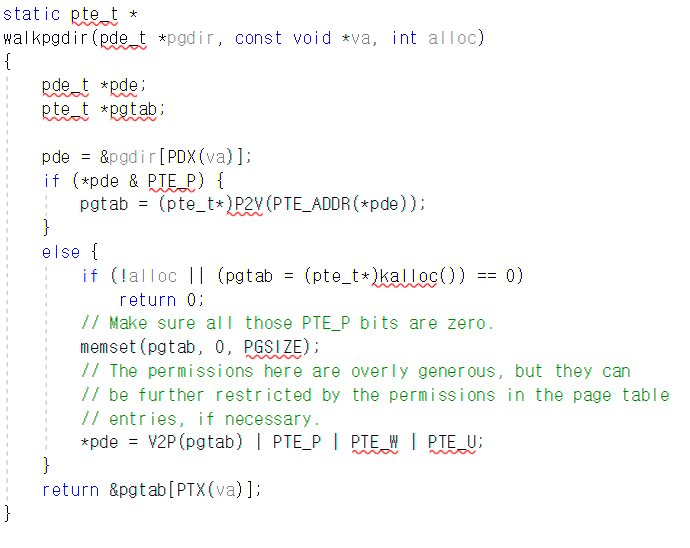
1. Shared memory

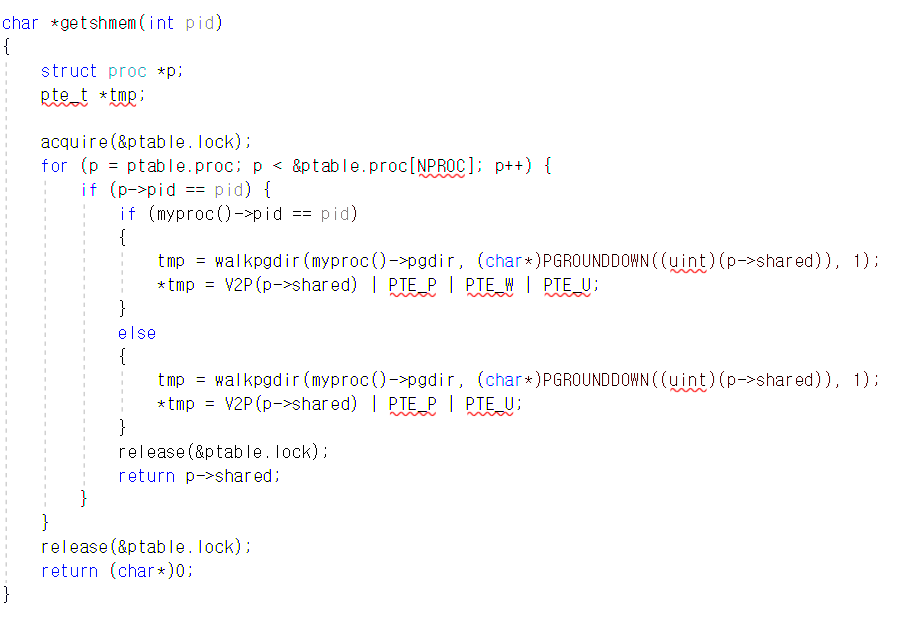


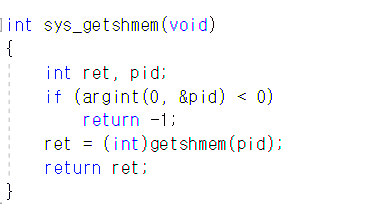
먼저 proc struct에 shared 항목을 추가해주었습니다. 여기 이 프로세스의 shared memory 주소를 저장해놓습니다.



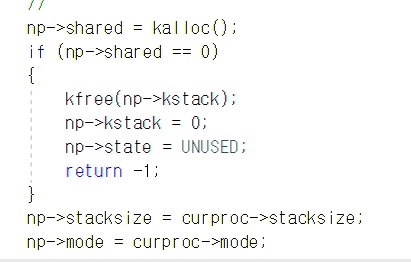
그리고 allocproc에서 프로세스가 생성될 때 kalloc을 통해 shared memory를 할당해줍니다.





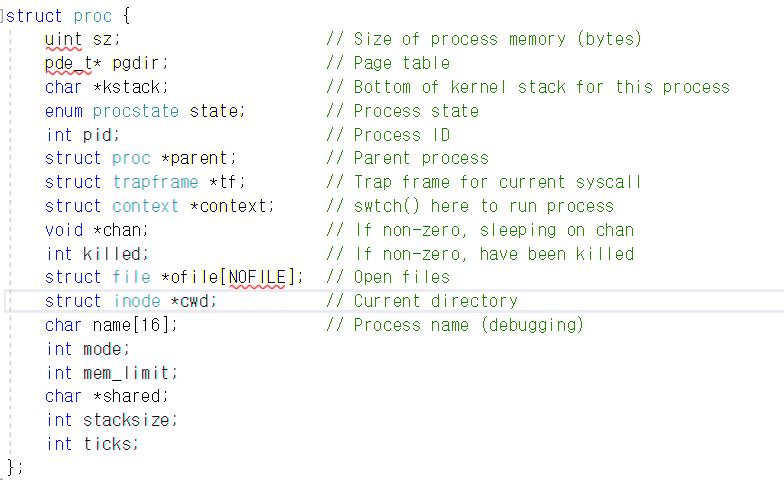


그리고 proc.c에 vm.c에 있던 walkpgdir 함수를 가져오고, getshmem, sys\_getshmem 함수를 통해 시스템 콜을 구현했습니다. Getshmem 함수에서는 ptable을 돌며 pid와 일치하는 프로세스를 찾아서 이 시스템 콜을 호출한 프로세스의 pid와 얻고자하는 shared memory의 pid가 일치하면 write 권한을 주고, 그렇지 않다면, write 권한을 주지 않았습니다. 그리고 pid가 일치하는 프로세스가 있다면 그 프로세스의 shared memory의 주소를 return 해주고, 그렇지 않다면 0을 return 해주었습니다.

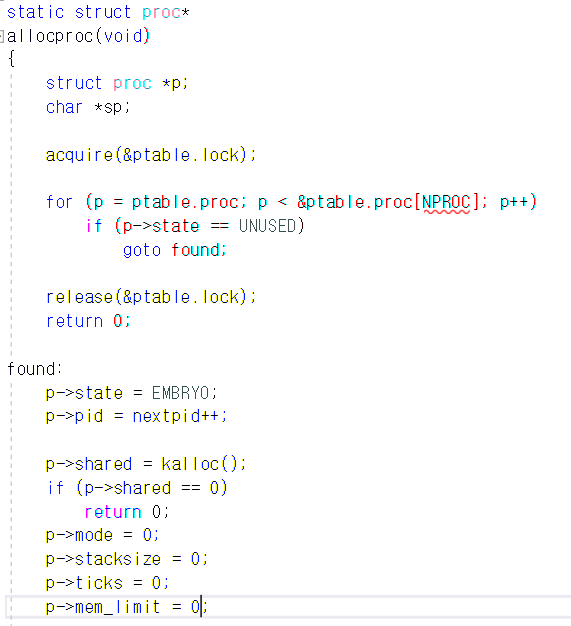


그리고 proc.c 안에 있는 fork 함수에서 fork된 프로세스에게 새로운 shared memory를 할당해주었고, stacksize와 mode를 그대로 넘겨주었습니다.

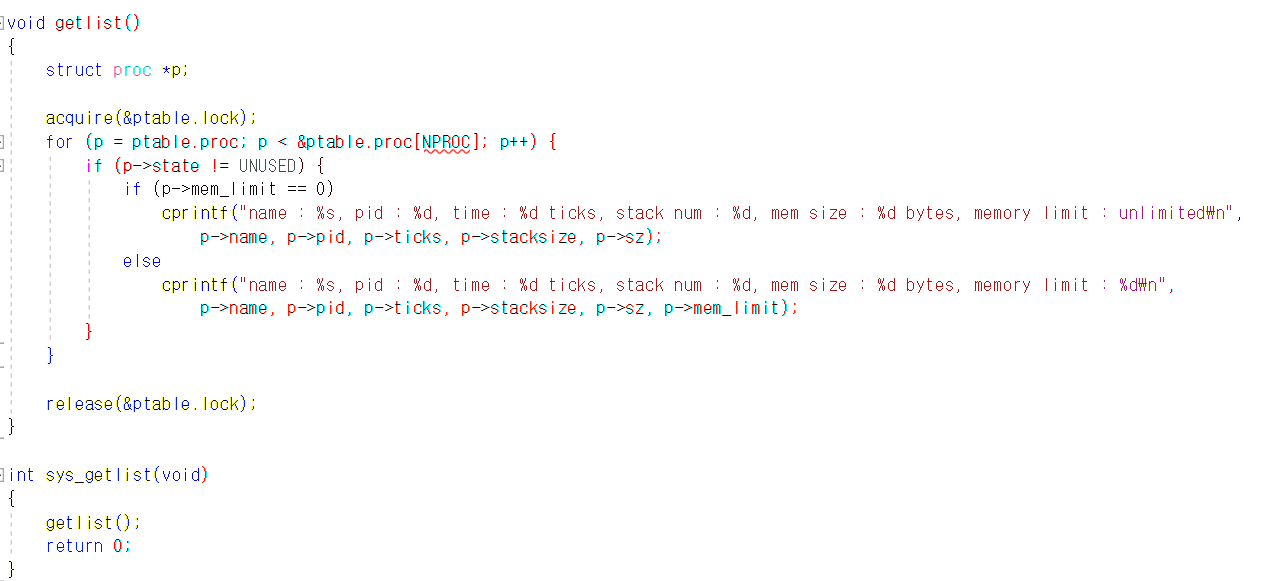
1. Process Manager



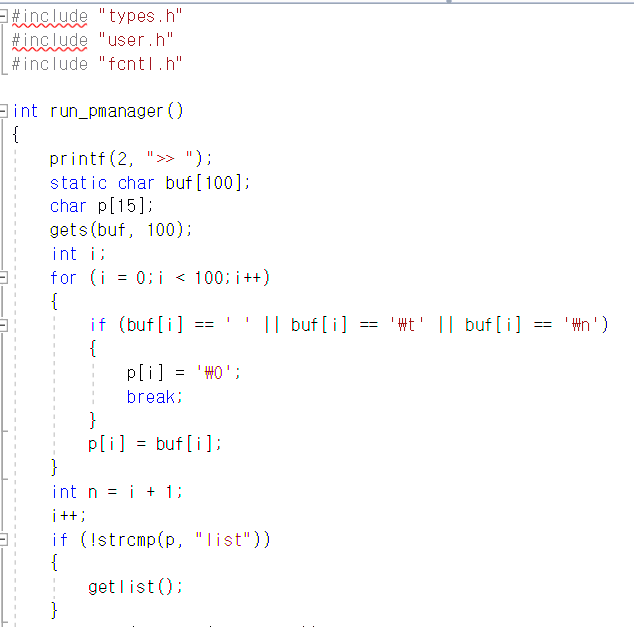
먼저 proc struct에 ticks 항목을 추가해주었습니다.

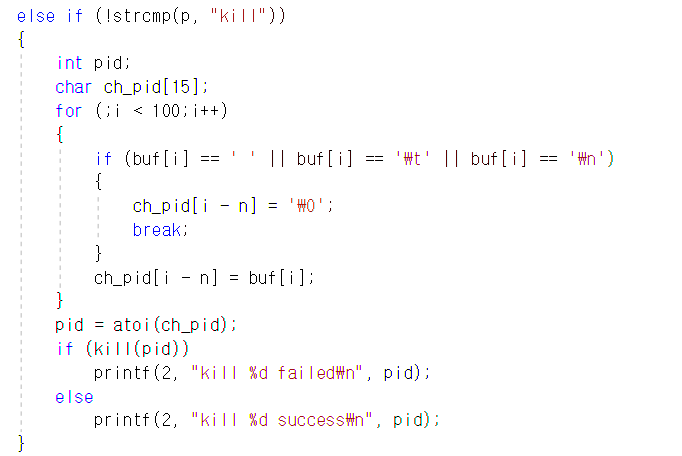


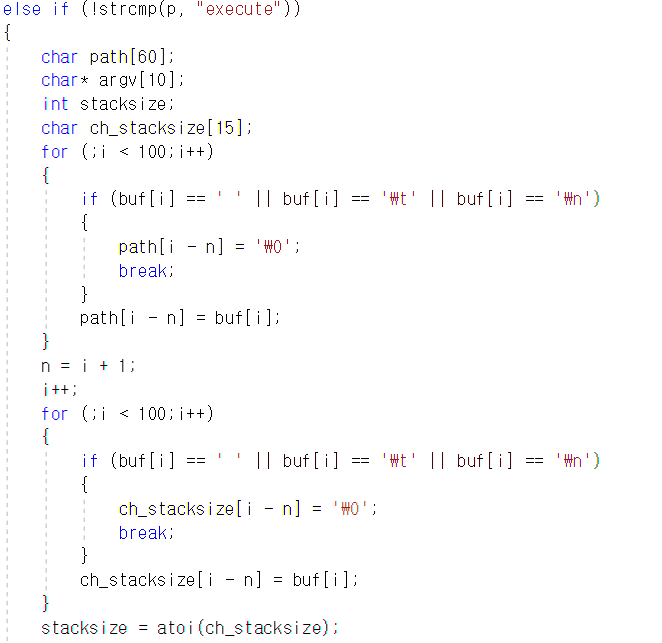
그리고 allocproc에서 프로세스가 생성될 때 ticks 값을 0으로 초기화해주고, 1 tick이 지날 때마다 이 tick 값을 1씩 증가시켜주었습니다.

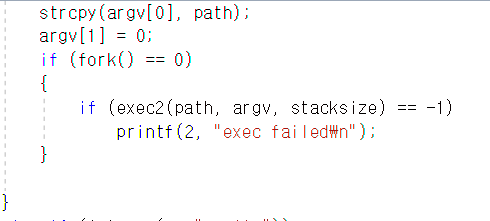


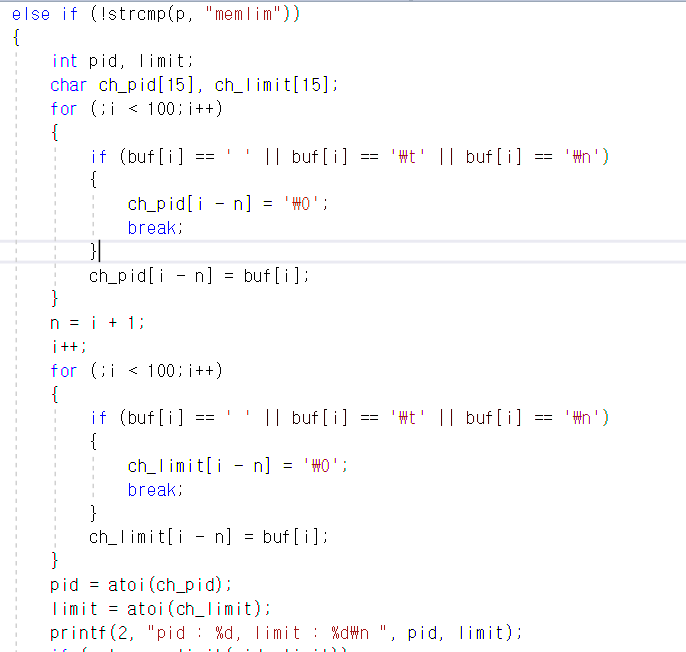
그리고 proc.c에 getlist, sys\_getlist 함수를 통해 시스템콜을 구현했습니다. 이를 통해 pmanager에서 list 명령어를 받았을 때 이 시스템 콜을 통해 UNUSED 상태가 아닌 모든 프로세스의 상태를 출력해주도록 했습니다. Mem\_limit이 0일 때에는 memory limit을 unlimited로 출력하도록 했습니다.

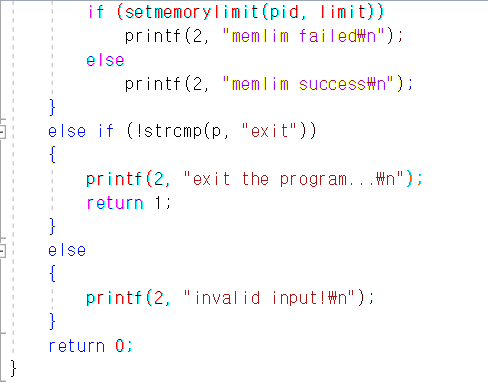


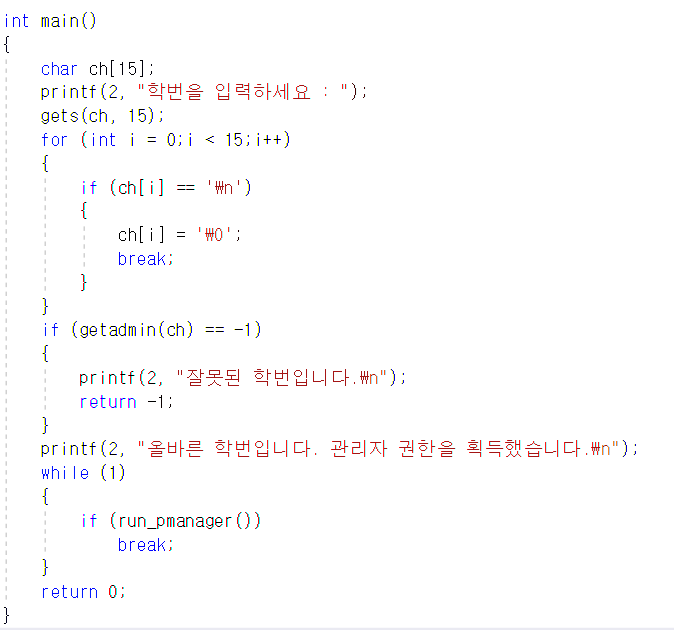








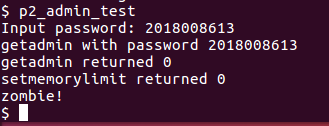


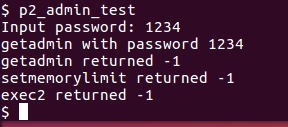


그리고 pmanager.c를 만들어서 pmanager를 구현했습니다. 먼저 main에서 getadmin 시스템 콜을 통해 학번을 입력받아서 일치하면 관리자 권한을 얻도록 해 주었습니다. Main의 while문을 돌면서 계속 run\_pmanager()를 호출하도록 했고, 여기서 getline을 통해 명령어를 입력받은 후 parsing해주었습니다. List 명령이 들어왔을 때 getlist() 시스템 콜을 통해 list를 출력해주었습니다. Kill 명령이 들어왔을 때에는 kill 시스템 콜을 통해 처리를 해주었고, 성공 여부를 출력해 주었습니다. Execute 명령이 들어왔을 때에는 exec2 시스템 콜을 통해 처리를 해주었고, fork를 통해 프로세스가 실행되고 있는 동안에도 pmanager를 실행할 수 있도록 해 주었습니다. 그리고 argv[0]에는 path를 넣어주었고, argv[1]에는 0을 넣어주었습니다. Return 값을 통해 실패 시 실패 메시지를 출력하도록 해 주었습니다. Memlim 명령이 들어왔을 때에는 setmemorylimit 시스템 콜을 통해 처리해 주었고, 성공여부를 출력해주었습니다. Exit 명령이 들어왔을 때에는 프로그램을 종료시켜주었습니다.

1. 실행 결과

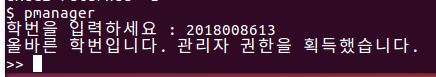
동영상을 따라보면서 같은 테스트를 진행했습니다.



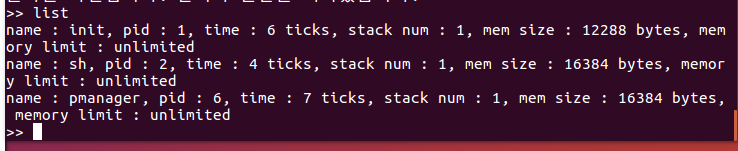


먼저 p2\_admin\_test를 했습니다. 먼저, 2018008613이라는 올바른 학번을 입력했을 때에는 getadmin, setmemorylimit 함수는 0을 return하면서 정상적으로 동작했고, exec2를 실행했을 때에도 zombie! 프로그램이 정상동작했습니다.

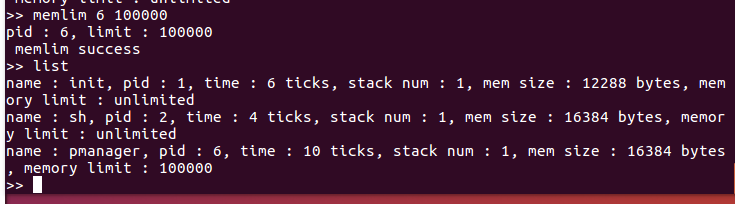
그런데, 1234라는 잘못된 학번을 입력했을 때에는 getadmin, setmemorylimit, exec2 함수가 -1을 return 했습니다.



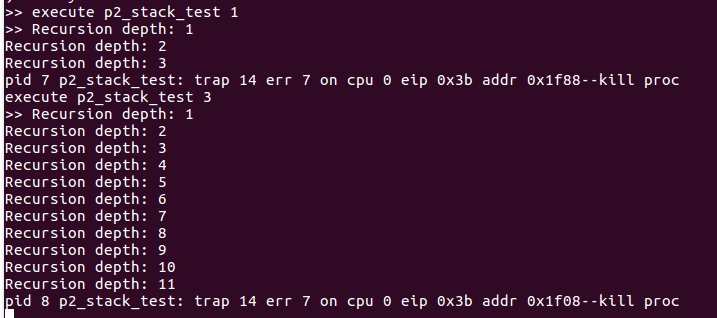
그리고 pmanager를 실행한 뒤 올바른 학번을 입력해 관리자 권한을 획득했습니다.



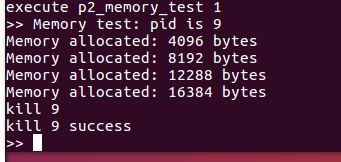
List를 실행했을 때 정상적으로 모든 프로세스들의 상태를 출력했습니다.



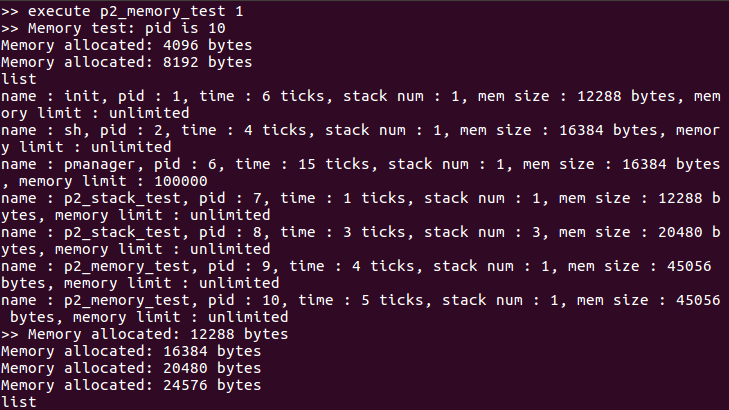
그리고 memlim을 통해 pid가 6인 프로세스의 memory limit을 100000으로 변경해준 뒤 list를 다시 실행시켜주면 pid가 6인 프로세스의 memory limit이 100000으로 출력됨을 확인했습니다.

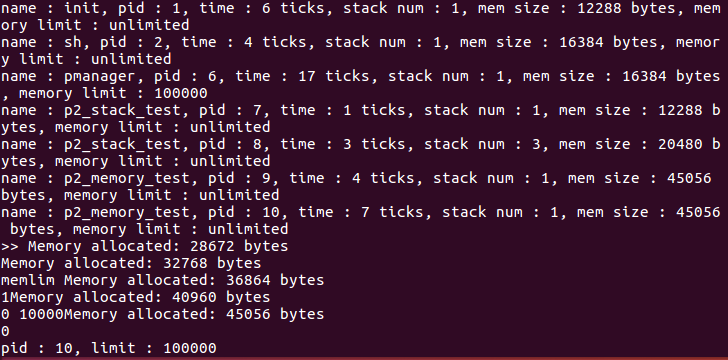


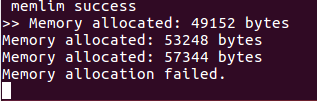
그리고 execute 명령을 통해 p2\_stack test를 실행한 결과 정상적으로 동작했습니다.



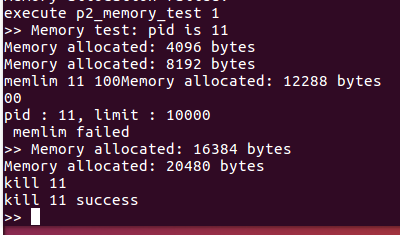
그리고 p2\_memory\_test 도중 kill 명령어를 통해 해당 프로세스를 정상적으로 종료시켰습니다.



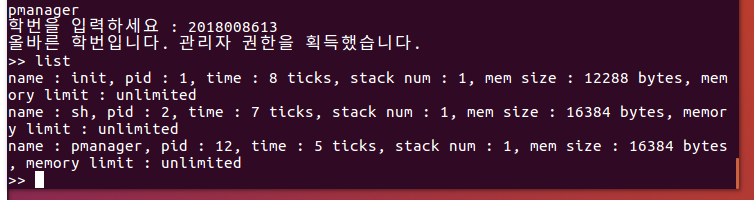




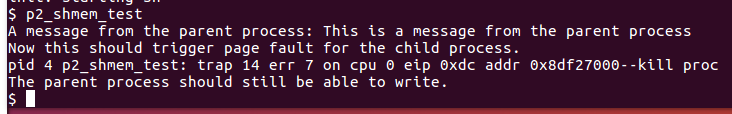
P2\_memory\_test를 한번 더 실행해 보았는데, 이 프로세스가 실행중일 때에도 list, memlim 등의 명령을 통해 pmanager를 실행시킬 수 있었습니다. 그리고, 중간에 memlim을 통해 memory limit을 100000으로 설정했는데, 이보다 메모리를 더 많이 할당하려고 하는 겨우 Memory allocation failed라는 메시지가 나오면서 프로그램이 종료되었습니다.



그리고 p2\_memory\_test를 한번 더 실행해서 할당되어 있는 메모리보다 더 작은 크기로 memory limit을 설정해 보았는데, memlim failed라는 메시지가 뜨면서 정상적으로 설정에 오류가 생기는 것을 볼 수 있습니다.



종료 후 pmanager를 재실행했을 때 기존의 process들은 나타나지 않는 것을 확인할 수 있었습니다.



그리고 shared memory test 결과 다음과 동영상과 같은 결과가 나왔습니다.

1. 트러블 슈팅

처음 작업을 수행할 때 fork에서 관리자 권한을 자식 프로세스에 양도하지 않아서 문제가 발생했는데, 이를 찾아서 잘 수정했습니다. 그리고 pmanager와 shared memory를 구현할 때 어려움이 있었는데 계속 고민한 결과 답을 찾아낼 수 있었습니다.