

1. Compiling Linux kernel

커널 빌드 과정의 오류들

```
make[3]: *** No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509_certificate_list'. Stop.  
make[3]: *** Waiting for unfinished jobs....  
CC      certs/system_keyring.o
```

```
make[2]: *** [scripts/Makefile.build:442: certs] Error 2  
make[2]: *** Waiting for unfinished jobs....  
HOSTCC  security/selinux/genheaders  
CC      inc/util.o
```

< 서명 인증 오류로 인한 오류 >

오류 해결을 위한 명령어

```
janginh@test2:/usr/src/linux-6.13.11$ sudo scripts/config --disable SYSTEM_TRUSTED_KEYS  
janginh@test2:/usr/src/linux-6.13.11$ sudo scripts/config --disable REVOCATION_KEYS  
janginh@test2:/usr/src/linux-6.13.11$ MODULE_SIG_ALL  
MODULE_SIG_ALL: command not found  
janginh@test2:/usr/src/linux-6.13.11$ sudo scripts/config --disable MODULE_SIG_ALL  
janginh@test2:/usr/src/linux-6.13.11$ sudo scripts/config --disable MODULE_SIG_KEY  
janginh@test2:/usr/src/linux-6.13.11$ sudo scripts/config --disable CONFIG_DEBUG_INFO_BTF
```

SYSTEM_TRUSTED_KEYS : 모듈 서명 검사 사용 X

REVOCATION_KEYS : 키 관련 오류 방지

MODULE_SIG_ALL : 자동 서명 X, 인증서 오류 방지

MODULE_SIG_KEY : 서명 키 설정 비활성화

CONFIG_DEBUG_INFO_BTF : 디버깅 정보 포함 X

커널 빌드 완료

```
janginh@test1: ~/linux-6.13.11
LD [M] net/nfc/nci/nci_spi.ko
LD [M] net/nfc/nci/nci_uart.ko
LD [M] net/nfc/hci/hci.ko
LD [M] net/nfc/nfc_digital.ko
LD [M] net/psample/psample.ko
LD [M] net/ife/ife.ko
LD [M] net/openvswitch/openvswitch.ko
LD [M] net/openvswitch/vport-vxlan.ko
LD [M] net/openvswitch/vport-geneve.ko
LD [M] net/openvswitch/vport-gre.ko
LD [M] net/vmw_vsock/vsock.ko
LD [M] net/vmw_vsock/vsock_diag.ko
LD [M] net/vmw_vsock/vmw_vsock_vmci_transport.ko
LD [M] net/vmw_vsock/vmw_vsock_virtio_transport.ko
LD [M] net/vmw_vsock/vmw_vsock_virtio_transport_common.ko
LD [M] net/vmw_vsock/hv_sock.ko
LD [M] net/vmw_vsock/vsock_loopback.ko
LD [M] net/nsh/nsh.ko
LD [M] net/hsr/hsr.ko
LD [M] net/qrtr/qrtr.ko
LD [M] net/qrtr/qrtr-smd.ko
LD [M] net/qrtr/qrtr-tun.ko
LD [M] net/qrtr/qrtr-mhi.ko
janginh@test1:~/linux-6.13.11$
```

리붓 진행

```
Ubuntu, with Linux 6.13.11-061311-generic
Ubuntu, with Linux 6.13.11-061311-generic (recovery mode)
Ubuntu, with Linux 6.13.11
*Ubuntu, with Linux 6.13.11 (recovery mode)
Ubuntu, with Linux 6.13.11.old
Ubuntu, with Linux 6.13.11.old (recovery mode)
Ubuntu, with Linux 6.11.0-25-generic
Ubuntu, with Linux 6.11.0-25-generic (recovery mode)
```

명령어 실행

```
janginh@test1:~$ uname -a
Linux test1 6.13.11-061311-generic #202504101401 SMP PREEMPT_DYNAMIC Thu Apr 10
20:20:36 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
janginh@test1:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.2 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
```

결론 :

Virtual Box을 이용해 가상환경을 만들어서 우분투 리눅스를 다운 받았으며, 디스크는 넉넉하게 120GB, 코어 6개, 메모리는 10GB를 통해 진행하였다.

리눅스 커널 컴파일 과정 중 생긴 오류들을 해결하기 위해 초기화를 여러번 진행하였다. 대부분 서명, 키 와 관련된 오류이기때문에 .config 파일을 변경하여 오류를 해결하였다.

커널 소스를 다운 받아 컴파일 하는 과정은 사용자가 기존의 커널에 접근해 수정, 확장을 위해 컴파일하는 것이다.

2. Adding my first system calls

코드생성 : helloworld.c 파일을 생성하여 코드를 생성함

```
GNU nano 7.2 helloworld.c
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(helloworld) {
    printk("Hello, Janginhwan!\n");
    printk("214683");
    return 0;
}
```

테스트 코드 생성

```
#include <unistd.h>
#include <sys/syscall.h>

#define MY_SYS_HELLO 548

int main(int argc, char*argv[]) {
    int ret = syscall (MY_SYS_HELLO);

    return 0;
}
```

Makefile에 시스템콜 추가

```
GNU nano 7.2 helloworld/Makefile *
obj-y:=helloworld.o
```



```

GNU nano 7.2                                Makefile *
endif

export KBUILD_MODULES KBUILD_BUILTIN

ifdef need-config
include $(objtree)/include/config/auto.conf
endif

ifeq ($(KBUILD_EXTMOD),)
# Objects we will link into vmlinux / subdirs we need to visit
core-y      := kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ helloworld/
drivers-y    :=
libs-y       := lib/
endif # KBUILD_EXTMOD

# The all: target is the default when no target is given on the
# command line.
# This allow a user to issue only 'make' to build a kernel including modules
# Defaults to vmlinux, but the arch makefile usually adds further targets
all: vmlinux

```

시스템콜 추가 : linux-*/arch/x86/entry/syscalls/syscall_64.tbl 파일에 추가

```

janganh@test1: ~/linux-6.13.11/arch/x86/entry/syscalls
GNU nano 7.2                                syscall_64.tbl *
532  x32    vmsplce             sys_vmsplce
533  x32    move_pages         sys_move_pages
534  x32    preadv             compat_sys_preadv64
535  x32    pwritev            compat_sys_pwritev64
536  x32    rt_tgsigqueueinfo  compat_sys_rt_tgsigqueueinfo
537  x32    recvmmsg          compat_sys_recvmmsg_time64
538  x32    sendmmsg          compat_sys_sendmmsg
539  x32    process_vm_readv   sys_process_vm_readv
540  x32    process_vm_writev  sys_process_vm_writev
541  x32    setsockopt         sys_setsockopt
542  x32    getsockopt         sys_getsockopt
543  x32    io_setup           compat_sys_io_setup
544  x32    io_submit          compat_sys_io_submit
545  x32    execveat           compat_sys_execveat
546  x32    preadv2            compat_sys_preadv64v2
547  x32    pwritev2           compat_sys_pwritev64v2
# This is the end of the legacy x32 range. Numbers 548 and above are
# not special and are not to be used for x32-specific syscalls.
548  common hello             _x64_sys_hello

```

[^]G Help [^]O Write Out [^]W Where Is [^]K Cut [^]T Execute [^]C Location
[^]X Exit [^]R Read File [^]\ Replace [^]U Paste [^]J Justify [^]/ Go To Line

헤더파일에 추가 : linux-*/include/linux/syscalls.h 파일에 어셈블리어 언어 함수를 호출 키워드 추가

```
janginh@test1: ~/linux-6.13.11/include/linux
GNU nano 7.2 syscalls.h *
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmdt(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
                            unsigned int nsops,
                            const struct old_timespec32 __user *timeout);
long __do_semtimedop(int semid, struct sembuf *tsems, unsigned int nsops,
                    const struct timespec64 *timeout,
                    struct ipc_namespace *ns);

int __sys_getsockopt(int fd, int level, int optname, char __user *optval,
                    int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                    int optlen);
#endif
asmlinkage long sys_hello(void);

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute  ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify  ^_ Go To Line
```

재빌드 : 컴파일 과정으로 재부팅까지 연결됨

```
janginh@test1: ~/linux-6.13.11$ make -j2
SYNC    include/config/auto.conf.cmd
SYSHDR  arch/x86/include/generated/uapi/asm/unistd_64.h
SYSHDR  arch/x86/include/generated/uapi/asm/unistd_x32.h
SYSHDR  arch/x86/include/generated/asm/unistd_64_x32.h
SYSTBL  arch/x86/include/generated/asm/syscalls_64.h
DESCEND objtool
INSTALL libsubcmd_headers
CC      arch/x86/kernel/asm-offsets.s
CALL    scripts/checksyscalls.sh
CC      init/main.o
CC      arch/x86/coco/tdx/tdx.o
AR      arch/x86/coco/tdx/built-in.a
CC      arch/x86/coco/sev/core.o
CC      init/do_mounts.o
CC      init/do_mounts_initrd.o
AR      arch/x86/coco/sev/built-in.a
AR      arch/x86/coco/built-in.a
AS      arch/x86/entry/entry_64.o
CC      init/initramfs.o
CC      arch/x86/entry/syscall_64.o
CC      arch/x86/entry/common.o
CC      init/init_task.o
```

리붓

```
Ubuntu, with Linux 6.13.11-061311-generic
Ubuntu, with Linux 6.13.11-061311-generic (recovery mode)
Ubuntu, with Linux 6.13.11
*Ubuntu, with Linux 6.13.11 (recovery mode)
Ubuntu, with Linux 6.13.11.old
Ubuntu, with Linux 6.13.11.old (recovery mode)
Ubuntu, with Linux 6.11.0-25-generic
Ubuntu, with Linux 6.11.0-25-generic (recovery mode)
```

결과

```
janginh@test1:~/linux-6.13.11$ sudo dmesg | grep -i hello
[ 301.500485] Hello, janginhwan!
                214683
```

결론

코드작성 - Makefile 수정 - syscall에 추가 - 헤더파일 추가 과정을 통해 컴파일 과정을 위한 재부팅 과정을 거쳐 결과가 도출되었다.

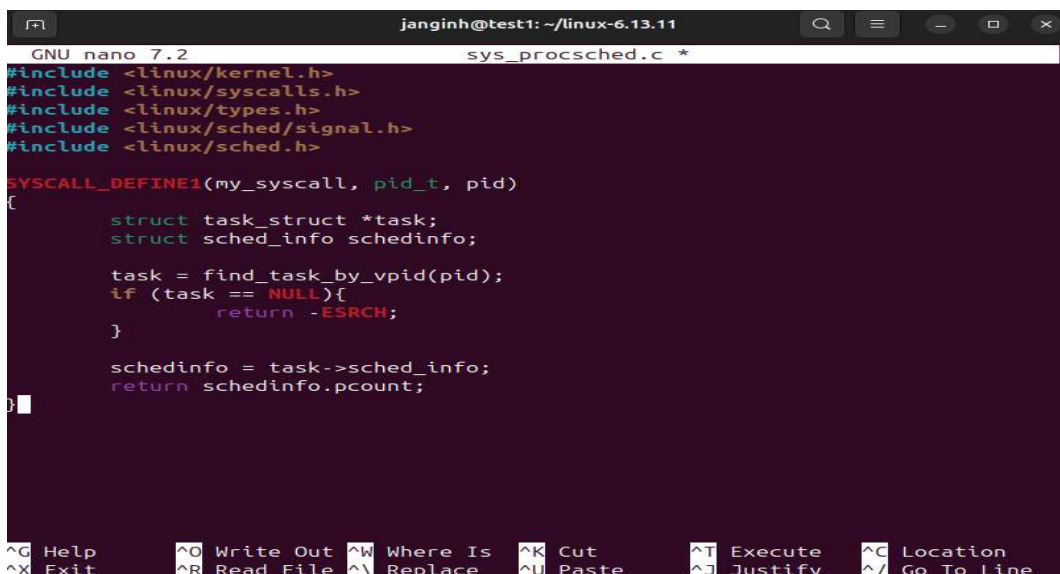
리눅스에서 시스템콜을 추가하기 위해 어셈블리어 추가

3. Taking a glance at PCB via Syscalls

조건 :

- 특정 프로세스의 PID를 인자로 입력받고
- 해당 프로세스에 대한 PCB, 즉 리눅스의 경우 'task_struct' 구조체를 읽은 후
- 해당 구조체 내부에 있는 'sched_info'를 가져와
- 내부의 pcount 정보를 반환

코드 생성



```
GNU nano 7.2 sys_procsched.c *
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/types.h>
#include <linux/sched/signal.h>
#include <linux/sched.h>

SYSCALL_DEFINE1(my_syscall, pid_t, pid)
{
    struct task_struct *task;
    struct sched_info schedinfo;

    task = find_task_by_vpid(pid);
    if (task == NULL){
        return -ESRCH;
    }

    schedinfo = task->sched_info;
    return schedinfo.pcount;
}
```

테스트 코드 생성


```

GNU nano 7.2 sys_syscall_pro.c
#include<unistd.h>
#include<stdio.h>
#include<sys.syscall.h>

#define MY_SYS_PROCSCHED 549

int main(int argc, char *argv[]){
    int ret = syscall( MY_SYS_PROCSCHED, 1234);
    print("pcount of 1234 = %d \n", ret);
    return 0;
}

```

Makefile 수정

```

GNU nano 7.2 Makefile
obj-y:=my_syscall.o

```

```

GNU nano 7.2 Makefile *
endif

export KBUILD_MODULES KBUILD_BUILTIN

ifdef need-config
include $(objtree)/include/config/auto.conf
endif

ifeq ($(KBUILD_EXTMOD),)
# Objects we will link into vmlinux / subdirs we need to visit
core-y      := kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ my_syscall/
drivers-y    :=
libs-y       := lib/
endif # KBUILD_EXTMOD

# The all: target is the default when no target is given on the
# command line.
# This allow a user to issue only 'make' to build a kernel including modules

```

Syscall 추가

```
janginh@test1: ~/linux-6.13.11/arch/x86/entry/syscalls
GNU nano 7.2 syscall_64.tbl *
527 x32 mq_notify compat_sys_mq_notify
528 x32 kexec_load compat_sys_kexec_load
529 x32 waitid compat_sys_waitid
530 x32 set_robust_list compat_sys_set_robust_list
531 x32 get_robust_list compat_sys_get_robust_list
532 x32 vmsplice sys_vmsplice
533 x32 move_pages sys_move_pages
534 x32 preadv compat_sys_preadv64
535 x32 pwritev compat_sys_pwritev64
536 x32 rt_tsigqueueinfo compat_sys_rt_tsigqueueinfo
537 x32 recvmmsg compat_sys_recvmmsg_time64
538 x32 sendmmsg compat_sys_sendmmsg
539 x32 process_vm_readv sys_process_vm_readv
540 x32 process_vm_writev sys_process_vm_writev
541 x32 setsockopt sys_setsockopt
542 x32 getsockopt sys_getsockopt
543 x32 io_setup compat_sys_io_setup
544 x32 io_submit compat_sys_io_submit
545 x32 execveat compat_sys_execveat
546 x32 preadv2 compat_sys_preadv64v2
547 x32 pwritev2 compat_sys_pwritev64v2
# This is the end of the legacy x32 range. Numbers 548 and above are
# not special and are not to be used for x32-specific syscalls.
548 common hello _x64_sys_hello
549 common procsched _x64_sys_procsched
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line
```

헤더파일 추가


```
janginh@test1: ~/linux-6.13.11/include/linux
GNU nano 7.2 syscalls.h *
long ksys_semget(key_t key, int nsems, int semflg);
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msgget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmdt(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
                            unsigned int nsops,
                            const struct old_timespec32 __user *timeout);
long __do_semtimedop(int semid, struct sembuf *tsems, unsigned int nsops,
                    const struct timespec64 *timeout,
                    struct ipc_namespace *ns);

int __sys_getsockopt(int fd, int level, int optname, char __user *optval,
                    int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                    int optlen);
#endif
asmlinkage long sys_hello(void);
asmlinkage long sys_procsched(pid_t pid);
```

재빌드

```
janginh@test1: ~/linux-6.13.11
CC      arch/x86/entry/syscall_32.o
CC      kernel/exit.o
AR      arch/x86/entry/built-in.a
CC      arch/x86/events/utils.o
CC      arch/x86/events/amd/core.o
CC      kernel/softirq.o
CC      arch/x86/events/amd/lbr.o
CC      arch/x86/events/amd/brs.o
CC      kernel/resource.o
CC      arch/x86/events/amd/ibs.o
CC      kernel/sysctl.o
CC      arch/x86/events/amd/iommu.o
CC [M]  arch/x86/events/amd/uncore.o
CC      kernel/capability.o
AR      arch/x86/events/amd/built-in.a
LD [M]  arch/x86/events/amd/amd-uncore.o
CC      arch/x86/events/msr.o
CC      kernel/ptrace.o
CC      arch/x86/events/intel/core.o
CC      kernel/user.o
CC      kernel/signal.o
CC      arch/x86/events/intel/bts.o
CC      arch/x86/events/intel/ds.o
CC      kernel/sys.o
CC      arch/x86/events/intel/knc.o
CC      arch/x86/events/intel/lbr.o
CC      kernel/umh.o
CC      arch/x86/events/intel/p4.o
```

결과

```
janginh@test1:~/virtual_Shared/helloworld$ gcc sys_procsched.c
janginh@test1:~/virtual_Shared/helloworld$ ./a.out
pcount of 1234 = 56107
janginh@test1:~/virtual_Shared/helloworld$

janginh@test1:~/linux-6.13.11$ ps -f 1172
UID      PID    PPID  C  STIME TTY          STAT      TIME CMD
janginh  1234    1033   1  02:06 ?          Ssl       0:56 /usr/bin/gnome-shell
```

결론

2번 과정과 유사한 과정으로 코드작성 - Makefile 수정 - syscall에 추가 - 헤더파일 추가
- 컴파일 및 재부팅 과정을 거쳐 진행 하였다.

Pcount는 현재 실행중인 프로세스의 cpu 사용 시간을 나타내는 값으로 ps -f를 통해 TIME의 0:56을 통해 확인이 가능하다.