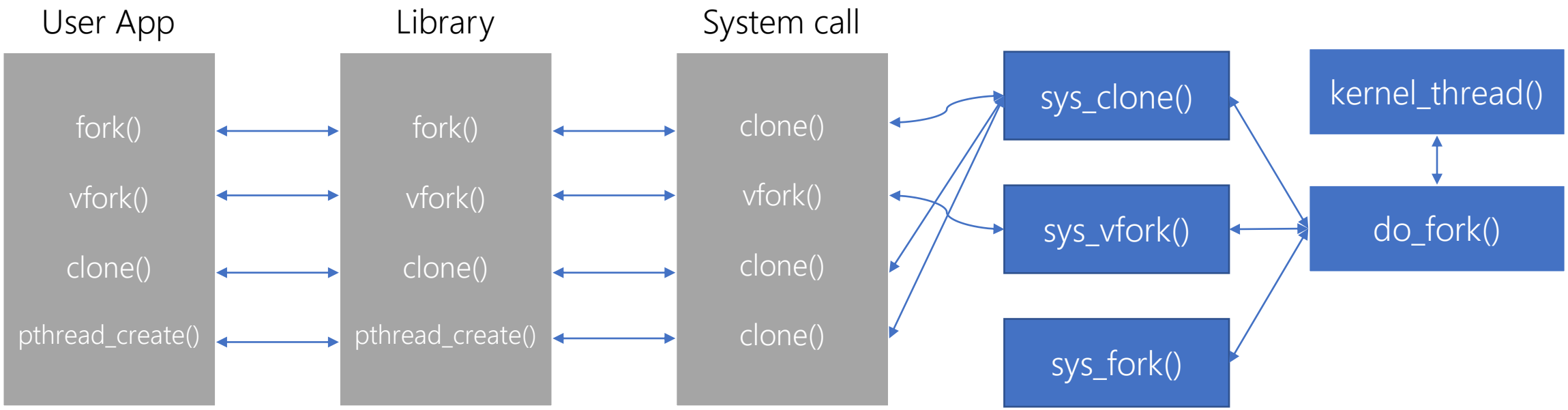
Linux Task Model

task_struct

- 리눅스에서 각 프로세스마다 생성하는 자료구조
- 리눅스에서는 프로세스 / 쓰레드 관계 없이 이 자료구조를 생성하여 관리
- fork() / vfork() / clone() / pthread_create()


User App
Library
System call
fork()
vfork()
clone()
pthread_create()
fork()
vfork()
clone()
pthread_create()
clone()
vfork()
clone()
clone()
sys_clone()
sys_vfork()
sys_fork()
kernel_thread()
do_fork()

fork() = 프로세스 생성함수

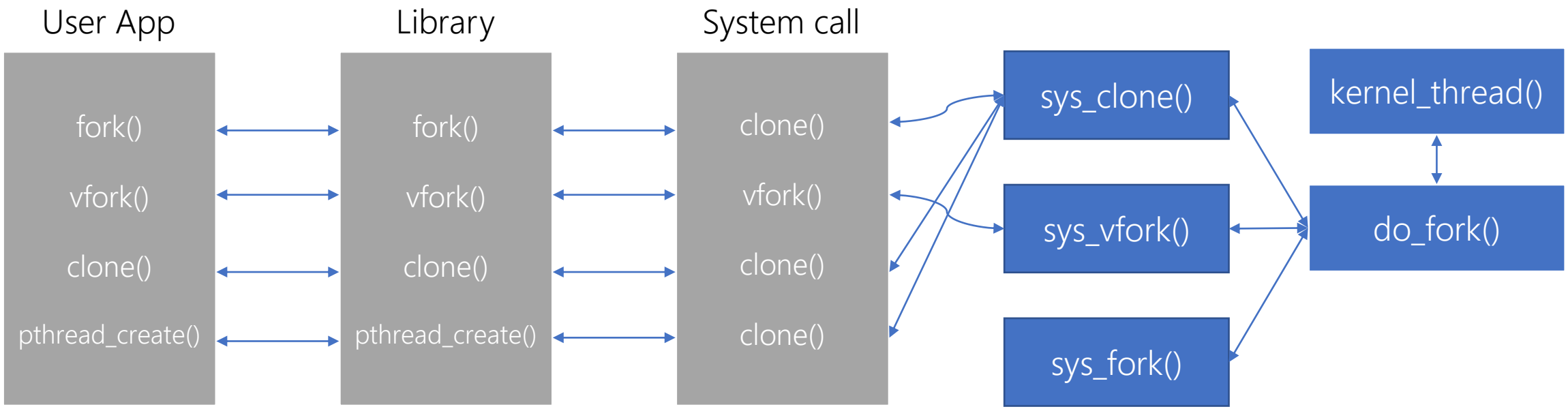
clone() = 쓰레드 생성함수

마지막은 공통적으로 do_fork() 호출?

리눅스 입장에서는 둘 다 태스크이기 때문.

fork는 부모와 덜 공유하는 태스크

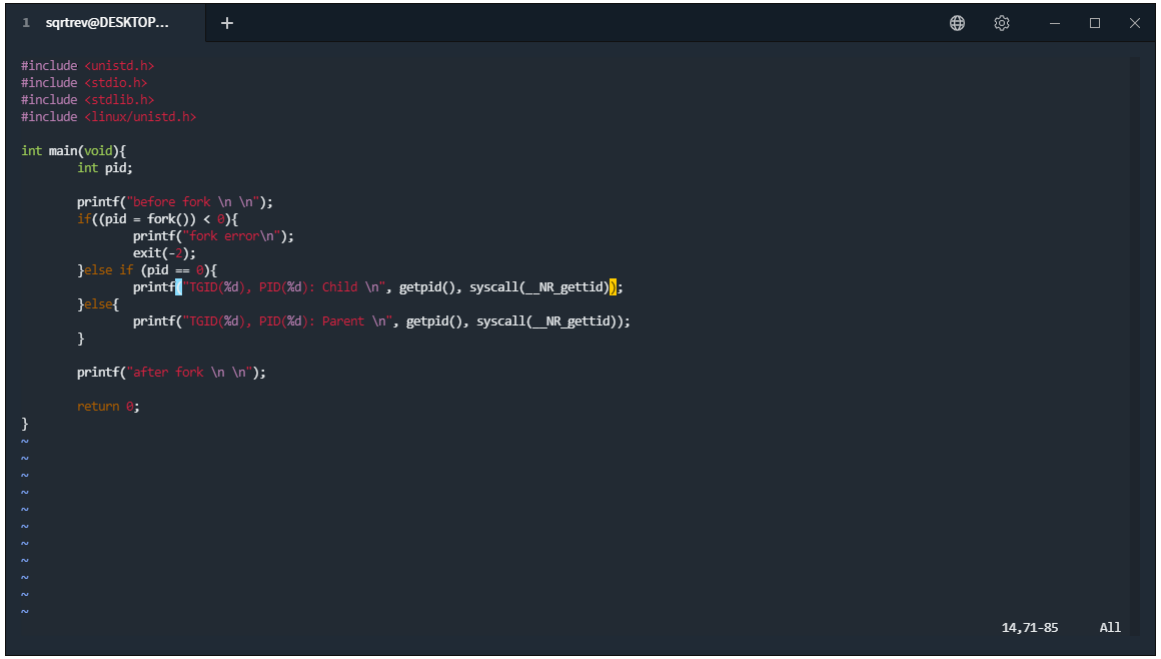
clone은 부모와 많이 공유하는 태스크



태스크 구분

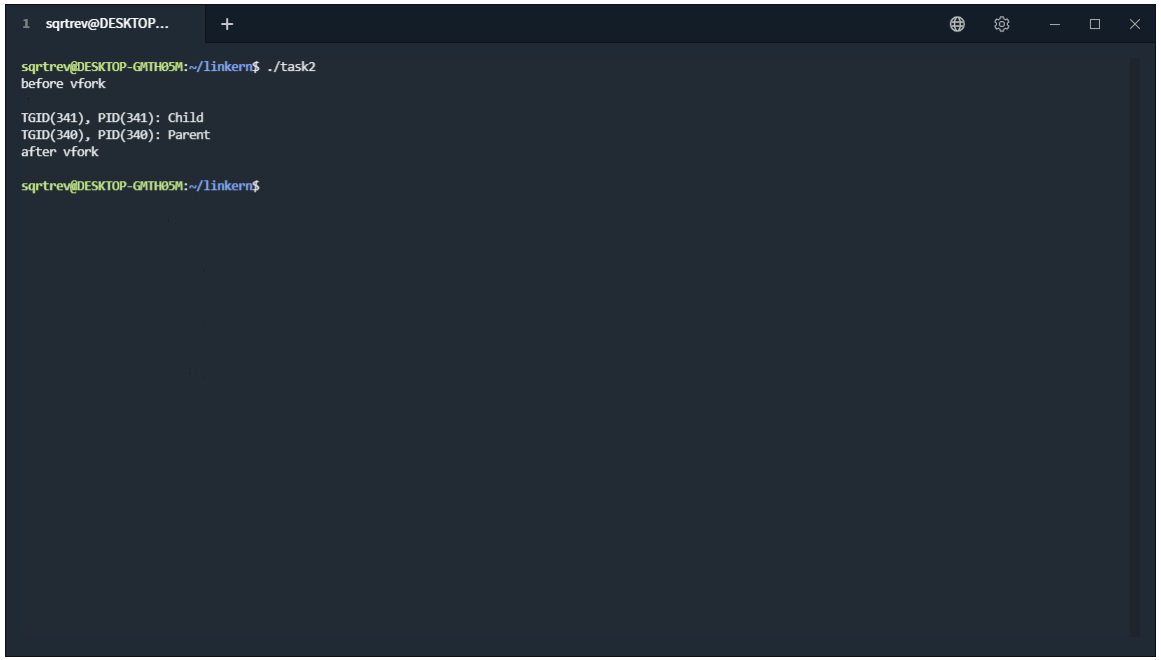
- task_struct의 pid 필드로 구분
- POSIX 표준 “한 프로세스 내의 쓰레드는 동일한 PID를 공유“
- tgid(Thread Group ID) 개념 도입
- 부모, 자식 태스크는 동일한 tgid를 가짐
- 즉, tgid로 동일한 프로세스에 속한 것인지 해석

fork();



태스크의 pid와 tgid가 부모 태스크와 자식 태스크 간에 서로 다름

vfork();



태스크의 pid와 tgid가 부모 태스크와 자식 태스크 간에 서로 다름

pthread_create()

```
1  sqrtrev@DESKTOP...
```

```c
#include <linux/unistd.h>
#include <pthread.h>
void *t_function(void *data){
        int id;
        int i = 0;
        pthread_t t_id;
        id = *((int *)data);
        printf("TGID(%d), PID(%d), pthread_self(%d): Child \n", getpid(), syscall(__NR_gettid), pthread_self());

        sleep(2);
}

int main(void){
        int pid, status;
        int a = 1;
        int b = 2;
        pthread_t p_thread[2];
        printf("before pthread_create \n\n");
        if((pid = pthread_create(&p_thread[0], NULL, t_function, (void*)&a)) < 0){
                perror("thread create error : ");
                exit(1);
        }
        if((pid = pthread_create(&p_thread[1], NULL, t_function, (void*)&b)) < 0){
                perror("thread create error : ");
                exit(2);
        }
        pthread_join(p_thread[0], (void **)&status);
        printf("pthread_join(%d) \n", status);
        pthread_join(p_thread[1], (void **)&status);
        printf("pthread_join(%d) \n", status);
        printf("TGID(%d), PID(%d) : Parent \n", getpid(), syscall(__NR_gettid));
        return 0;
}
```

```
33,1-8        Bot
```

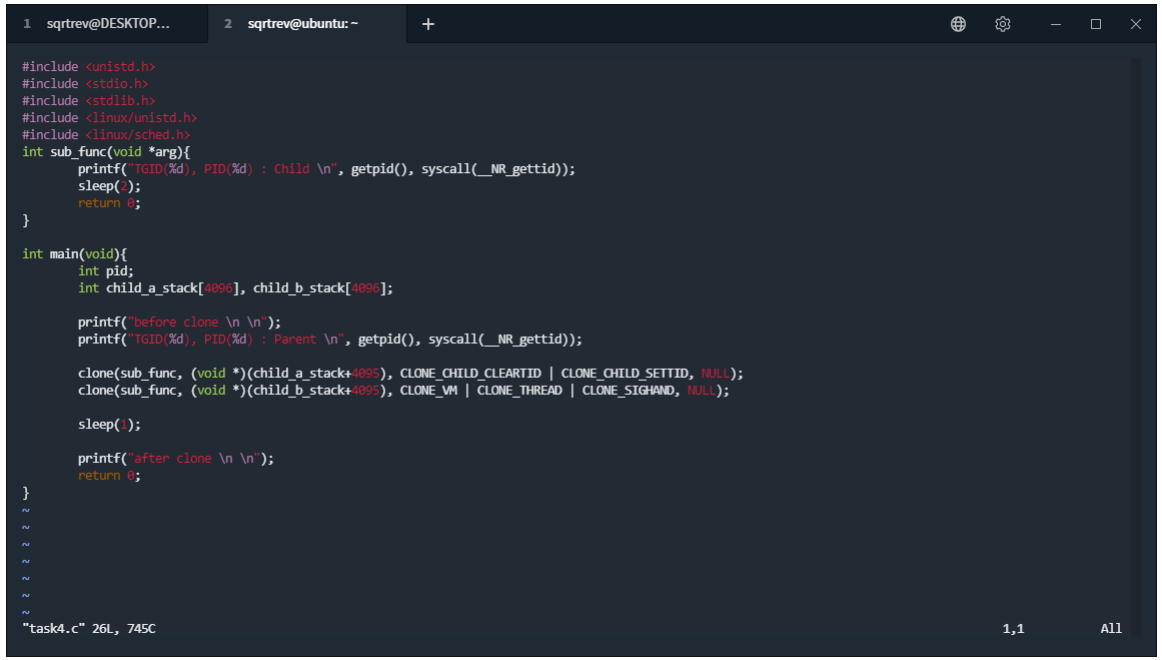
```
1  sqrtrev@DESKTOP...
```

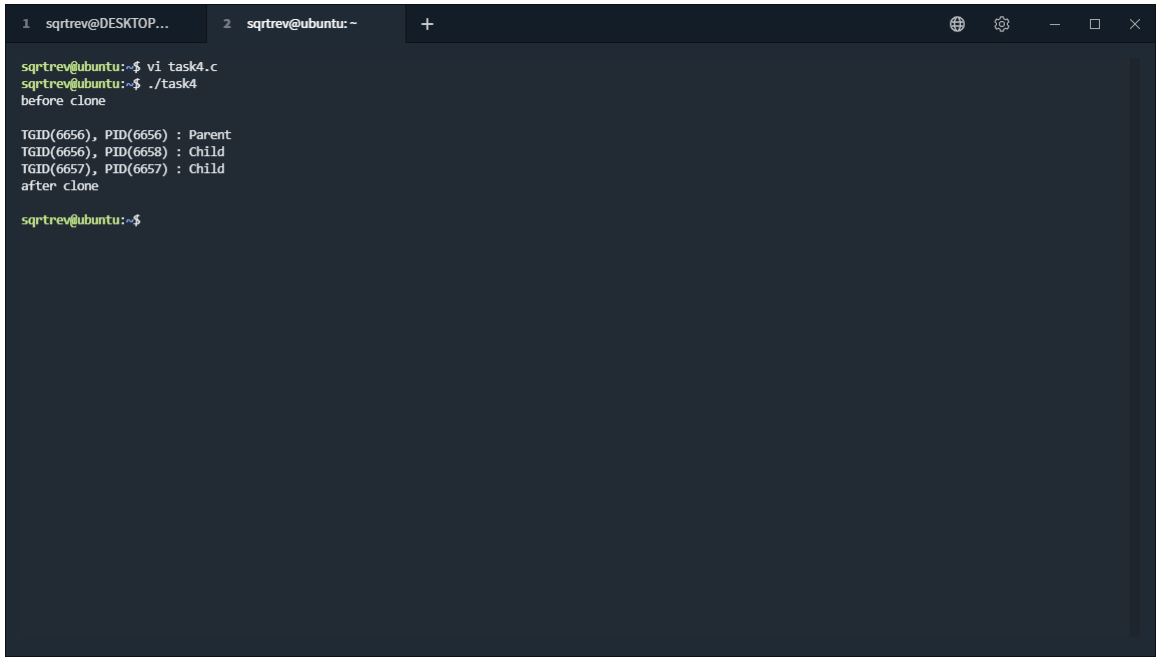
```
sqrtrev@DESKTOP-GMTH05M:~/linkern$ gcc -pthread -o task3 ./task3.c
./task3.c: In function ‘t_function’:
./task3.c:11:25: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘long int’ [-Wformat=]
  printf("TGID(%d), PID(%d), pthread_self(%d): Child \n", getpid(), syscall(__NR_gettid), pthread_self());
                        ~^                                 ~~~~~~~~~~~~~~~~~~~~
                        %ld
./task3.c:11:43: warning: format ‘%d’ expects argument of type ‘int’, but argument 4 has type ‘pthread_t {aka long unsigned int}’ [-Wformat=]
  printf("TGID(%d), PID(%d), pthread_self(%d): Child \n", getpid(), syscall(__NR_gettid), pthread_self());
                                          ~^                                              ~~~~~~~~~~~~~~
                                          %ld
./task3.c: In function ‘main’:
./task3.c:34:25: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘long int’ [-Wformat=]
  printf("TGID(%d), PID(%d) : Parent \n", getpid(), syscall(__NR_gettid));
                        ~^                           ~~~~~~~~~~~~~~~~~~~~
                        %ld
sqrtrev@DESKTOP-GMTH05M:~/linkern$ ./task3
before pthread_create

TGID(437), PID(438), pthread_self(-880998656): Child
TGID(437), PID(439), pthread_self(-889452800): Child
pthread_join(0)
pthread_join(0)
TGID(437), PID(437) : Parent
sqrtrev@DESKTOP-GMTH05M:~/linkern$
```

태스크의 pid는 다르지만 tgid는 같음

clone()





CLONE_CHILD_CLEARID = 자식 메모리에서 쓰레드 ID지움
CLONE_VM = 자식 프로세스와 메모리를 공유함
CLONE_SIGHAND = 부모와 자식간 같은 시그널 테이블 공유

CLONE_CHILD_SETTID = 자식 메모리에 쓰레드 ID 저장
CLONE_THREAD = 부모 프로세스와 같은 쓰레드 그룹으로 처리

CLONE_CHILD_CLEARID / CLONE_CHILD_SETTID -> 프로세스 해석(자원 공유 x)
CLONE_THREAD -> 쓰레드 해석(자원 공유 o)

Task Context

Task Context

- 1. System Context
  - 정보를 유지하기위해 커널이 할당한 자료구조들이다.
  - (task_struct,fd,file table, segment table,page table, etc)
- 2. Memory Context
  - text, data, stack, heap, swap
- 3. Hardware Context
  - Context switc할때 task의 현재 실행위치 정보 유지

Task_struct

- ~/include/linux/sched.h에 정의되어있음

1. Task identification
2. State
3. Task relationship
4. Scheduling information
5. Signal information
6. Memory information
7. File information
8. Thread structure
9. Time information
10. Format
11. Resource limits

Task identification

- 쓰임새: Task 인식하기 위한 변수

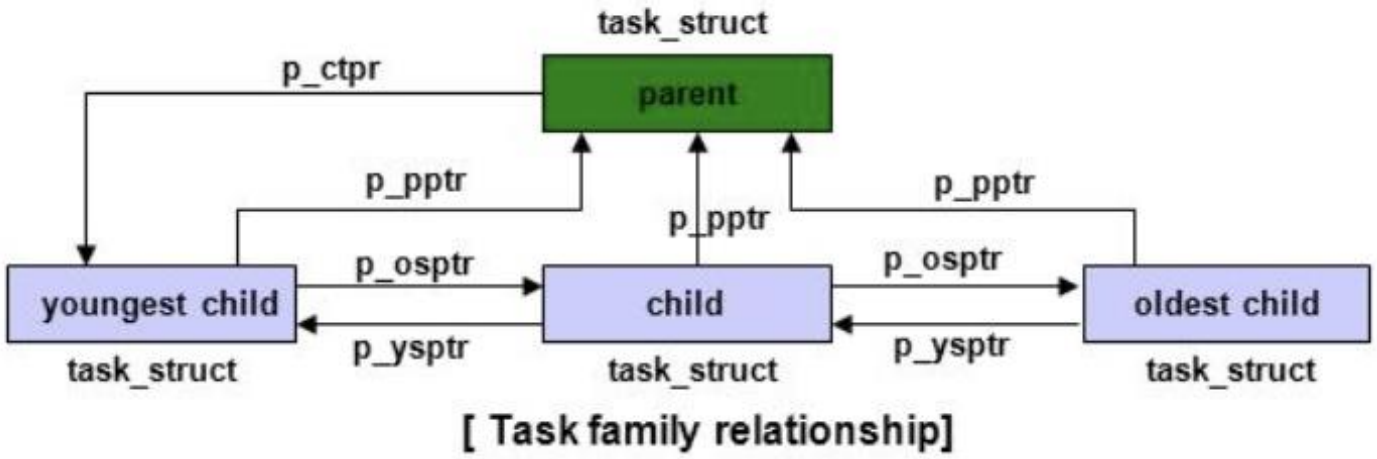
- 대표적, Pid,tgid,hash 관련필드 변수
- 사용자에 대한 접근 제어권한
  - uid,euid,suid,fsuid
- 사용자 그룹에 대한 접근제어
  - Gid,egid,sgid,fssgid

State

- 생성에서 소멸까지 상태관리를 위한 변수
  - RUNNING(0),INTERRUPTIBLE(1),UN INTERRUPTIBLE(2),STOPPED(4)
  - TRACED(8),EXIT_DEAD(16),EXIT_ZOMBIE(32)

Task relationship

Task는 생성시 가족관계를 갖음



Information

- Scheduling
  - prio, policy, cpus_allowd,time_slice, rt_priority와 같은 스케쥴링변수가 존재
- Singal
  - 비동기적인 사건발생을 알림
    - Signal,sighand,blocked,pending 등이 있음
- Memory
  - Task_struct에는 메모리영역 공간에 대한 위치,크기,접근제어정보를 관리하는 변수 존재
  - 가상주소->물리주소 변환하기 위한 페이지 디렉터리와 테이블 등 정보도 존재
  - Task_struct의 mm_struct라는 변수로 접근가능
- File
  - Task가 오픈한 파일을 files_struct 구조체형태인 files변수로 접근가능
  - Ffs변수로 루트디렉토리, 현재디렉토리의 inode 접근가능
- Time
  - Task의 시간정보, start_time,real_start_time 등이 존재
  - 사용CPU 시간의 통계를 담는 필드도 있음

Thread Structure

- Context Switch할때 Task가 현재어디까지 실행되었는지 저장하는공간

Resource limits

- Task가 사용가능한 자원의 한계
  - rlim_max : 최대 허용자원 수
  - rlim_cur : 현재 설정된 허용 자원의 수

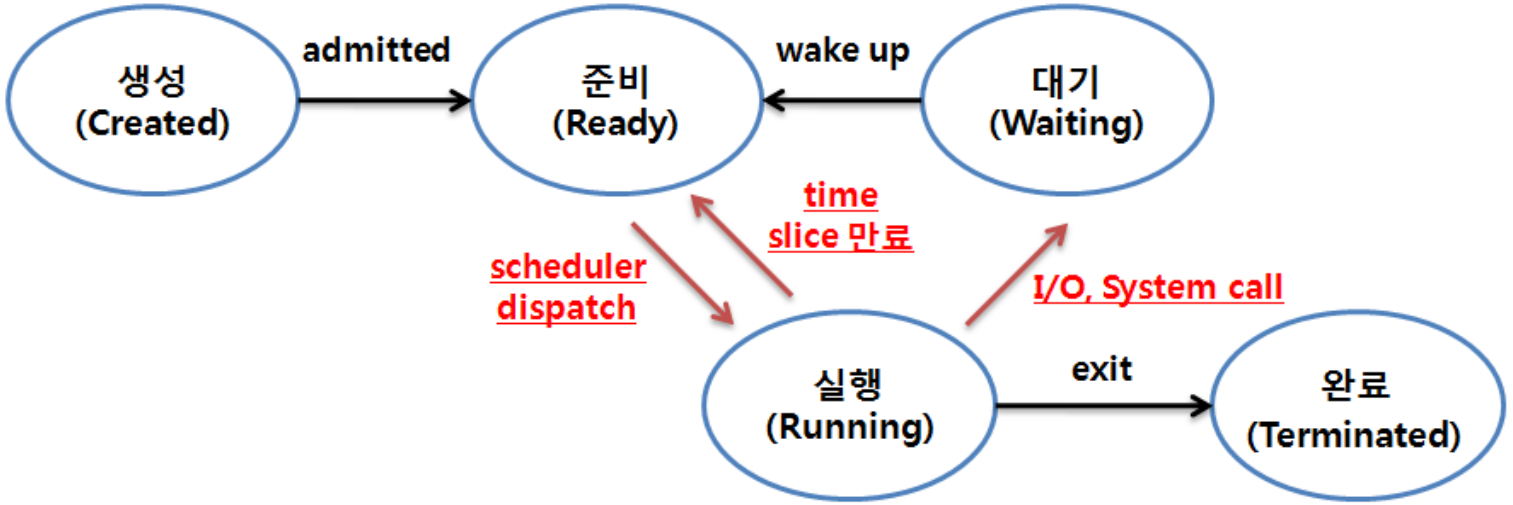
- 현 린커널은 최대 16개의 자원에 대한 한계 설정 가능

문맥 교환

문맥 교환에는 총 4번의 CPU 레지스터 정보 저장/복원이 이루어진다.

문맥 교환

- 현재 진행중인 task 상태 저장 & 다음 task 수행할 준비
- CPU의 멀티 테스킹을 위해 필요
- 오버헤드 발생 -> 최적화된 스케줄링 필요



기본 구조체

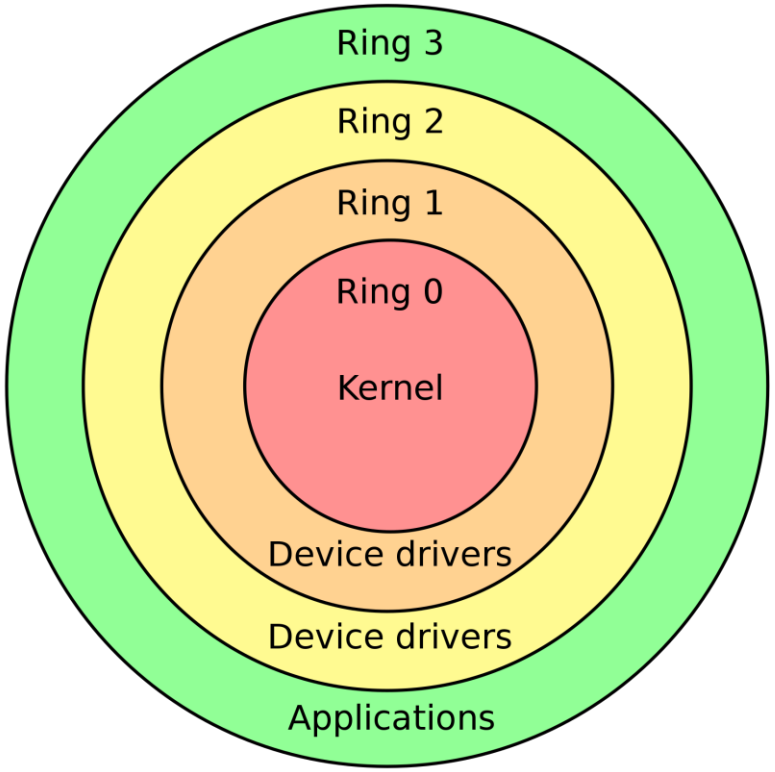
```
struct task_struct {
        volatile long state;/* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;/* per process flags, defined below */
        unsigned int ptrace;
                .
                .
                .
        /* CPU-specific state of this task */
        struct thread_struct thread;  <-여기에 context 저장
                .
                .
                .
}
```

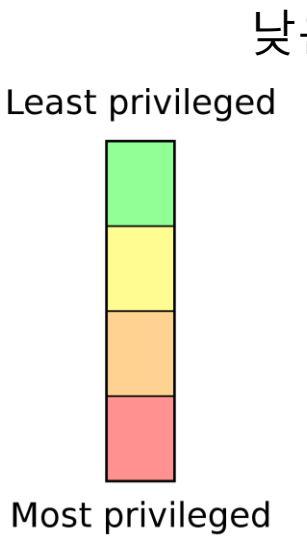
```
struct thread_struct {
        /* Cached TLS descriptors: */
        struct desc_struct      tls_array[GDT_ENTRY_TLS_ENTRIES];
        unsigned long           sp0;
        unsigned long           sp;
#ifdef CONFIG_X86_32
        unsigned long           sysenter_cs;
#else
        unsigned long           usersp; /* Copy from PDA */
        unsigned short          es;
        unsigned short          ds;
        unsigned short          fsindex;
        unsigned short          gsindex;
#endif
#ifdef CONFIG_X86_32
        unsigned long           ip;
#endif
#ifdef CONFIG_X86_64
        unsigned long           fs;
#endif
        unsigned long           gs;
        /* Save middle states of ptrace breakpoints */
        struct perf_event       *ptrace_bps[HBP_NUM];
        /* Debug status used for traps, single steps, etc... */
        unsigned long           debugreg6;
        /* Keep track of the exact dr7 value set by the user */
        unsigned long           ptrace_dr7;
        /* Fault info: */
        unsigned long           cr2;
        unsigned long           trap no;
        unsigned long           error_code;
        /* floating point and extended processor state */
        struct fpu              fpu;
#ifdef CONFIG_X86_32
        /* Virtual 86 mode info */
        struct vm86_struct __user *vm86_info;
        unsigned long           screen_bitmap;
        unsigned long           v86flags;
        unsigned long           v86mask;
        unsigned long           saved_sp0;
        unsigned int            saved_fs;
        unsigned int            saved_gs;
#endif
        /* IO permissions: */
        unsigned long           *io_bitmap_ptr;
        unsigned long           iopl;
        /* Max allowed port in the bitmap, in bytes: */
        unsigned                io_bitmap_max;
};
```

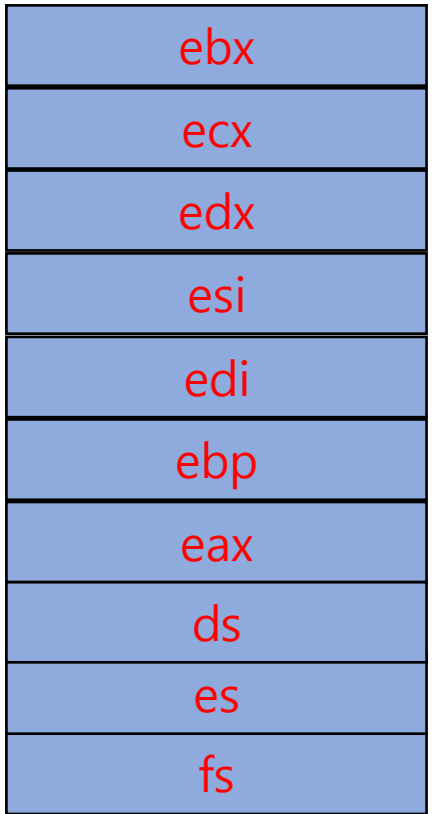
```
struct vm86_struct {
        struct vm86_regs regs;
        unsigned long flags;
        unsigned long screen_bitmap;
        unsigned long cpu_type;
        struct revectored_struct int_revectored;
        struct revectored_struct int21_revectored;
};
```

과정 1 - 상태전이

사용자 모드 -> 커널 모드 (상태 전이)로 전환되는 과정에서 커널 스택에 cpu 레지스터 정보 저장








```
.macro SAVE_ALL
        cld
        PUSH_GS
        pushl_cfi %fs
        /*CFI_REL_OFFSET fs, 0;*/
        pushl_cfi %es
        /*CFI_REL_OFFSET es, 0;*/
        pushl_cfi %ds
        /*CFI_REL_OFFSET ds, 0;*/
        pushl cfi %eax
        CFI_REL_OFFSET eax, 0
        pushl cfi %ebp
        CFI_REL_OFFSET ebp, 0
        pushl cfi %edi
        CFI_REL_OFFSET edi, 0
        pushl cfi %esi
        CFI_REL_OFFSET esi, 0
        pushl cfi %edx
        CFI_REL_OFFSET edx, 0
        pushl cfi %ecx
        CFI_REL_OFFSET ecx, 0
        pushl cfi %ebx
        CFI_REL_OFFSET ebx, 0
        movl $(__USER_DS), %edx
        movl %edx, %ds
        movl %edx, %es
        movl $(__KERNEL_PERCPU), %edx
        movl %edx, %fs
        SET_KERNEL_GS %edx
.endm
```

과정 2, 3

```
unsigned long ebx, ecx, edx, esi, edi;                                  \
                                                                        \
asm volatile("pushfl\n\t"                /* save    flags */           \
             "pushl %%ebp\n\t"           /* save    EBP   */           \
             "movl %%esp,%[prev_sp]\n\t"         /* save    ESP */      \
             "movl %[next_sp],%%esp\n\t"         /* restore ESP */      \
             "movl $1f,%[prev_ip]\n\t"   /* save    EIP   */           \
             "pushl %[next_ip]\n\t"      /* restore EIP   */           \
             __switch_canary                                            \
             "jmp __switch_to\n"         /* regparm call  */           \
             "1:\t"                                                     \
             "popl %%ebp\n\t"            /* restore EBP   */           \
             "popfl\n"                   /* restore flags */           \
                                                                        \
             /* output parameters */                                    \
             : [prev_sp] "=m" (prev->thread.sp),                        \
               [prev_ip] "=m" (prev->thread.ip),                        \
               "=a" (last),                                             \
                                                                        \
               /* clobbered output registers: */                        \
               "=b" (ebx), "=c" (ecx), "=d" (edx),                      \
               "=S" (esi), "=D" (edi)                                   \
                                                                        \
               __switch_canary_oparam                                   \
                                                                        \
               /* input parameters: */                                  \
             : [next_sp]  "m" (next->thread.sp),                        \
               [next_ip]  "m" (next->thread.ip),                        \
                                                                        \
               /* regparm parameters for __switch_to(): */              \
               [prev]     "a" (prev),                                   \
               [next]     "d" (next)                                    \
                                                                        \
               __switch_canary_iparam                                   \
                                                                        \
             : /* reloaded segment registers */                         \
              "memory");                                                \
```

```
/*
 * Reload esp0.
 */
load_sp0(tss, next);

/*
 * Save away %gs. No need to save %fs, as it was saved on the
 * stack on entry.  No need to save %es and %ds, as those are
 * always kernel segments while inside the kernel.  Doing this
 * before setting the new TLS descriptors avoids the situation
 * where we temporarily have non-reloadable segments in %fs
 * and %gs.  This could be an issue if the NMI handler ever
 * used %fs or %gs (it does not today), or if the kernel is
 * running inside of a hypervisor layer.
 */
lazy_save_gs(prev->gs);

/*
 * Load the per-thread Thread-Local Storage descriptor.
 */
load_TLS(next, cpu);
```

Task struct에 CPU레지스터 정보를 저장 & 복원

과정4

커널 모드 -> 사용자 모드로 전환 -> 커널 스택에서 CPU 정보 복

```
.macro RESTORE_REGS pop=0
        RESTORE_INT_REGS
1:      popl_cfi %ds
        /*CFI_RESTORE ds;*/
2:      popl_cfi %es
        /*CFI_RESTORE es;*/
3:      popl_cfi %fs
        /*CFI_RESTORE fs;*/
        POP_GS \pop
.pushsection .fixup, "ax"
4:      movl $0, (%esp)
        jmp 1b
5:      movl $0, (%esp)
        jmp 2b
6:      movl $0, (%esp)
        jmp 3b
```

```
.macro RESTORE_INT_REGS
        popl cfi %ebx
        CFI_RESTORE ebx
        popl cfi %ecx
        CFI_RESTORE ecx
        popl cfi %edx
        CFI_RESTORE edx
        popl cfi %esi
        CFI_RESTORE esi
        popl cfi %edi
        CFI_RESTORE edi
        popl cfi %ebp
        CFI_RESTORE ebp
        popl cfi %eax
        CFI_RESTORE eax
.endm
```

Task & signal

Task가 signal을 처리하기 위한 3가지 기능

- 다른 task에게 signal을 보낼 수 있어야 함
- 자신에게 signal이 오면 이를 수신할 수 있어야 함
- signal을 수신 했을 때, 이를 처리할 수 있는 함수를 지정할 수 있어야 함

Signal

- Linux 기본 지원 signal 32개 + posix signal 32개

- https://github.com/torvalds/linux/blob/master/include/linux/signal.h

```
*
*      +--------------------+------------------+
*      |  POSIX signal      |  default action  |
*      +--------------------+------------------+
*      |  SIGHUP            |  terminate       |
*      |  SIGINT            |  terminate       |
*      |  SIGQUIT           |  coredump        |
*      |  SIGILL            |  coredump        |
*      |  SIGTRAP           |  coredump        |
*      |  SIGABRT/SIGIOT    |  coredump        |
*      |  SIGBUS            |  coredump        |
*      |  SIGFPE            |  coredump        |
*      |  SIGKILL           |  terminate(+)    |
*      |  SIGUSR1           |  terminate       |
*      |  SIGSEGV           |  coredump        |
*      |  SIGUSR2           |  terminate       |
*      |  SIGPIPE           |  terminate       |
*      |  SIGALRM           |  terminate       |
*      |  SIGTERM           |  terminate       |
*      |  SIGCHLD           |  ignore          |
*      |  SIGCONT           |  ignore(*)       |
*      |  SIGSTOP           |  stop(*)(+)      |
*      |  SIGTSTP           |  stop(*)         |
*      |  SIGTTIN           |  stop(*)         |
*      |  SIGTTOU           |  stop(*)         |
*      |  SIGURG            |  ignore          |
*      |  SIGXCPU           |  coredump        |
*      |  SIGXFSZ           |  coredump        |
*      |  SIGVTALRM         |  terminate       |
*      |  SIGPROF           |  terminate       |
*      |  SIGPOLL/SIGIO     |  terminate       |
*      |  SIGSYS/SIGUNUSED  |  coredump        |
*      |  SIGSTKFLT         |  terminate       |
*      |  SIGWINCH          |  ignore          |
*      |  SIGPWR            |  terminate       |
*      |  SIGRTMIN-SIGRTMAX |  terminate       |
*      +--------------------+------------------+
*      |  non-POSIX signal  |  default action  |
*      +--------------------+------------------+
*      |  SIGEMT            |  coredump        |
*      +--------------------+------------------+
*
```

Task_struct

```c
struct task_struct{
  ...
  struct signal_struct    *signal; // 전체가 작동하는 signal 저장
  struct sighand_struct __rcu   *sighand; // 특정 signal을 받으면 이를 실행할 함수 저장
  sigset_t      blocked; // signal을 받지않게끔 설정, SIGKILL, SIGSTOP 예외
  sigset_t      real_blocked;
  /* Restored if set_restore_sigmask() was used: */
  sigset_t      saved_sigmask;
  struct sigpending   pending; // 특정 task에서만 실행되는 signal 저장
}
```

Signal 받을 때

해당 task의 task_struct를 찾음

-> siginfo 구조를 초기화 한 다음 해당 signal에 따라서 singal 혹은 pending 에 저장함.

-> 이 과정에서 block 체크

signal의 처리는 kernel → user 로 이동할 때 이루어짐. (pending, signal의 개수를 통해 시그널 여부 확인 가능)

signal이 있으면 sighand 의 action 배열을 찾아서 해당 시그널에 맞는 함수 실행.
만약 없으면 무시, 종료, 중지 등과 같은 기본적인 행동 실행

Signal 받을 때

https://github.com/torvalds/linux/blob/master/kernel/signal.c#L1070

do_send_specific -> do_send_sig_info -> __send_signal

Signal 받을 때

```
switch ((unsigned long) info) {
case (unsigned long) SEND_SIG_NOINFO:
        clear_siginfo(&q->info);
        q->info.si_signo = sig;
        q->info.si_errno = 0;
        q->info.si_code = SI_USER;
        q->info.si_pid = task_tgid_nr_ns(current,
                                         task_active_pid_ns(t));
        rcu_read_lock();
        q->info.si_uid =
                from_kuid_munged(task_cred_xxx(t, user_ns),
                                 current_uid());
        rcu_read_unlock();
        break;
case (unsigned long) SEND_SIG_PRIV:
        clear_siginfo(&q->info);
        q->info.si_signo = sig;
        q->info.si_errno = 0;
        q->info.si_code = SI_KERNEL;
        q->info.si_pid = 0;
        q->info.si_uid = 0;
        break;
default:
        copy_siginfo(&q->info, info);
        break;
}
```

Signal 받을 때

```
it_set:
        signalfd_notify(t, sig);
        sigaddset(&pending->signal, sig);

        /* Let multiprocess signals appear after on-going forks */
        if (type > PIDTYPE_TGID) {
                struct multiprocess_signals *delayed;
                hlist_for_each_entry(delayed, &t->signal->multiprocess, node) {
                        sigset_t *signal = &delayed->signal;
                        /* Can't queue both a stop and a continue signal */
                        if (sig == SIGCONT)
                                sigdelsetmask(signal, SIG_KERNEL_STOP_MASK);
                        else if (sig_kernel_stop(sig))
                                sigdelset(signal, SIGCONT);
                        sigaddset(signal, sig);
                }
        }

        complete_signal(sig, t, type);
+.
```

interrupt , trap vs signal

interrupt, trap 은 사건을 커널한테 알리지만,
signal을 task에 알림.