Operating System 실습
[5주차]
Process, Thread

한양대학교
HANYANG UNIVERSITY

Data-Centric Systems Lab.

Process and Thread

- Program
  - 실행 가능한 형태의 파일

```
os2013502346@os2014-VirtualBox:~/test$ ls
test  test.c
```

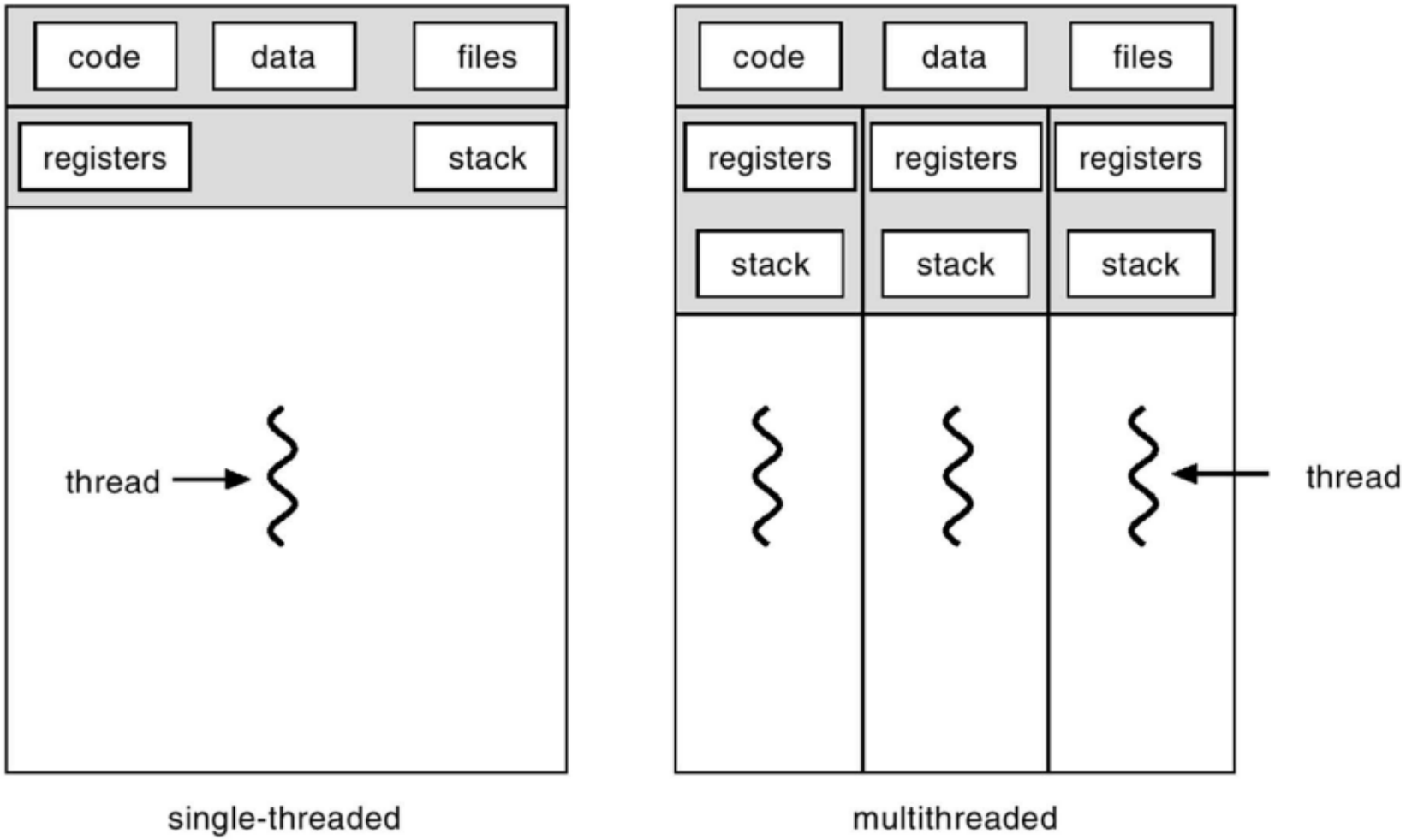
- Process (자원 소유권 단위)
  - 동작중인 프로그램 (Running or runnable program)
  - 자신만의 고유 공간과 자원 할당 받음
    * 다른 프로세스와 자원을 공유하기 위해서
      (Share address space, file system resources, file descriptors and signal handlers)

```
os2013502346@os2014-VirtualBox:~$ ps -t pts/0
  PID TTY          TIME CMD
 4413 pts/0    00:00:00 bash
 4544 pts/0    00:00:00 test
```

- Thread (수행의 단위)
  - 프로세스 내에서 동작되는 실행 흐름
  - 프로세스 내에서 다른 쓰레드와 공간과 자원을 공유

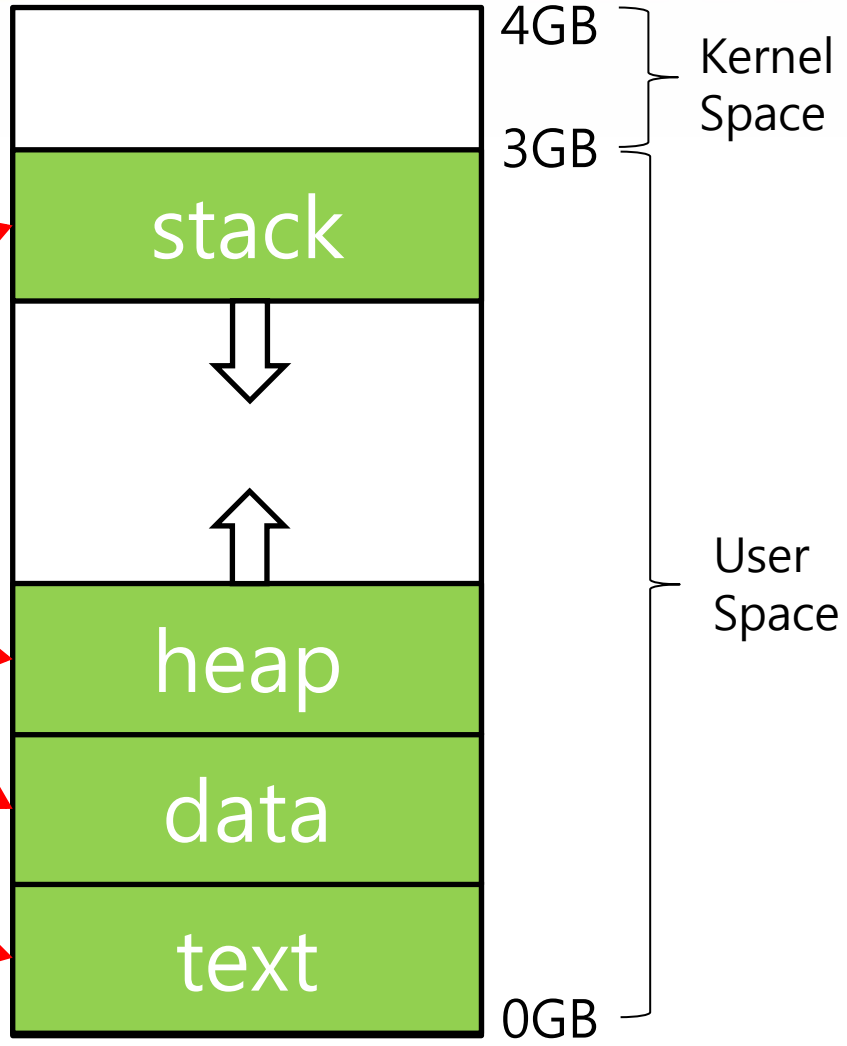
- **리눅스에서는 Process, Thread 모두 Task로 관리**

Process and Thread



Process 구조

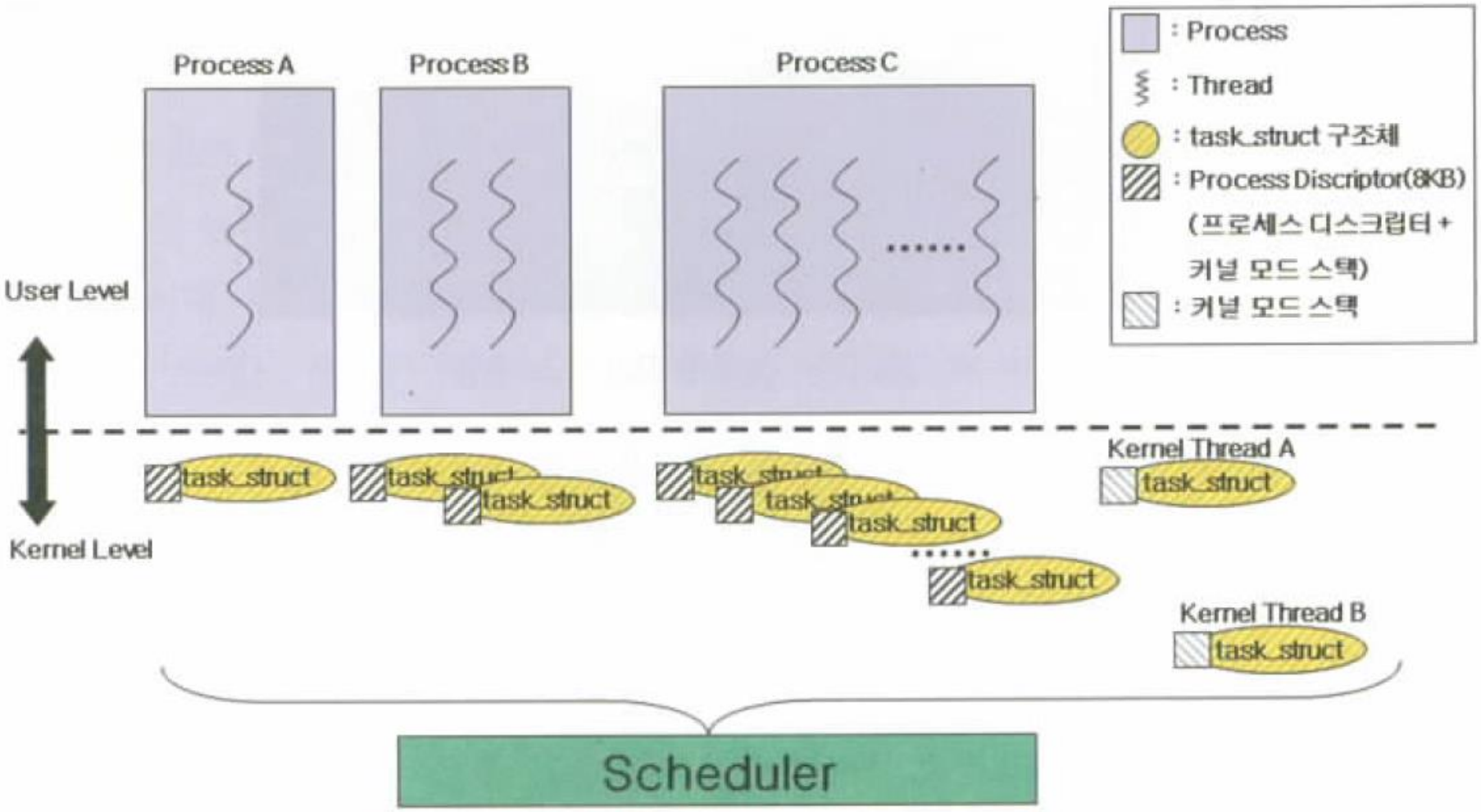
```c
#include <stdio.h>
#include <stdlib.h>
int glob;
int main(void)
{
    int local, *dynamic;
    dynamic = malloc(1383);

    printf("Local = %p\n", &local);
    printf("Dynamic = %p\n", dynamic);
    printf("Global = %p\n", &glob);
    printf("main = %p\n", main);
}
```



Task

- 실행 중인 프로그램의 수행 단위
- 스케쥴링되는 타켓
- 자신의 메모리 공간(코드, 변수, 스택)과 하드웨어 레지스터
- 태스크 계층구조
- 상태와 전이

Task Management Structrue



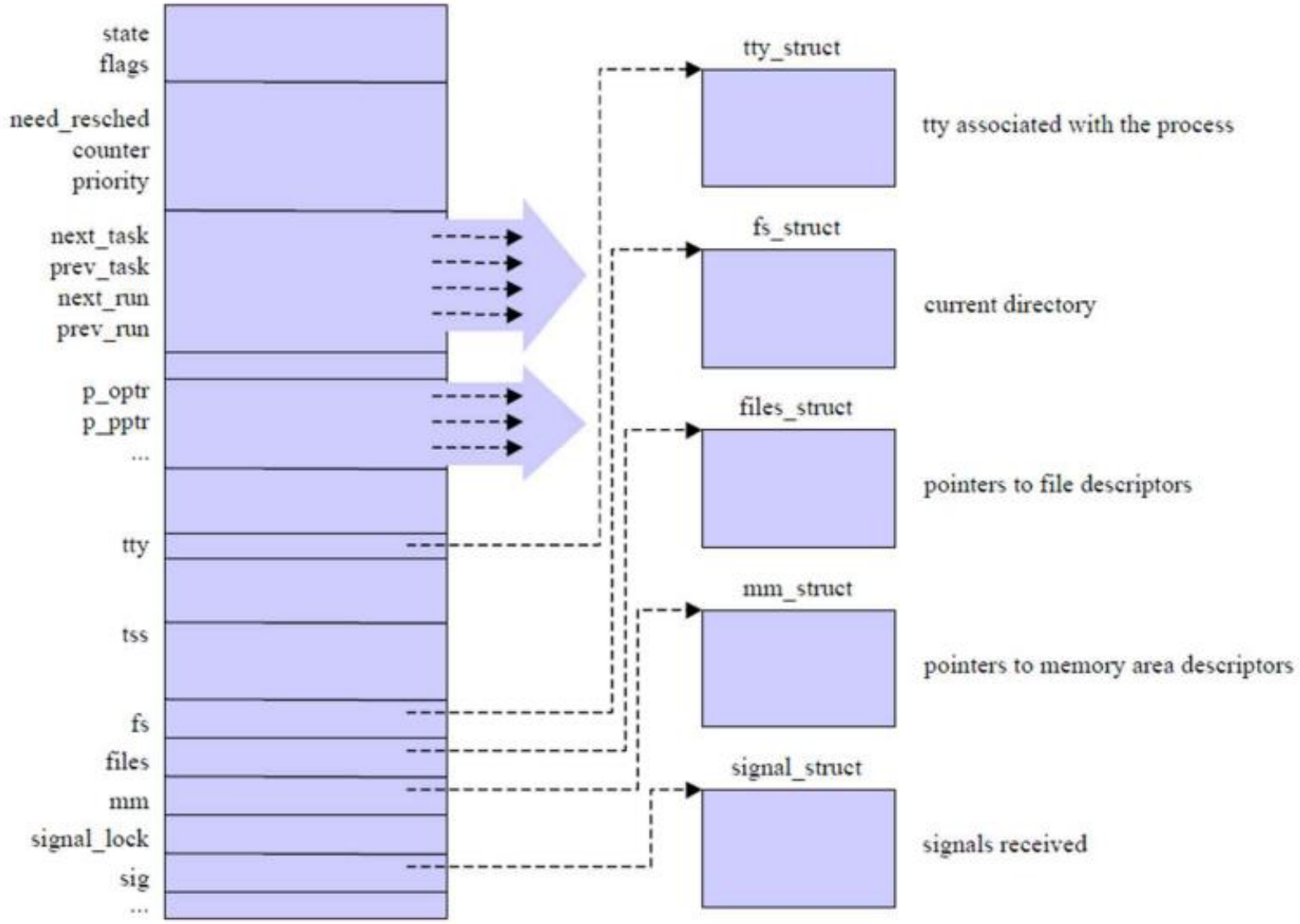
Process descriptor

- task_struct - process의 모든 정보를 담고있는 자료구조
  - Identifier
  - State
  - Process relationships
  - Scheduling information
    - prio, policy, cpus_allowd, time_slice, etc.
  - Virtual memory
    - segment, page
  - Virtual file system
    - file descriptor
  - Signal information
    - signal_struct, sighand, blocked, pending, etc.
  - Thread structure
  - Resource limits
  - Time information
  - Executable format
  - Process synchronization

Details of task_struct

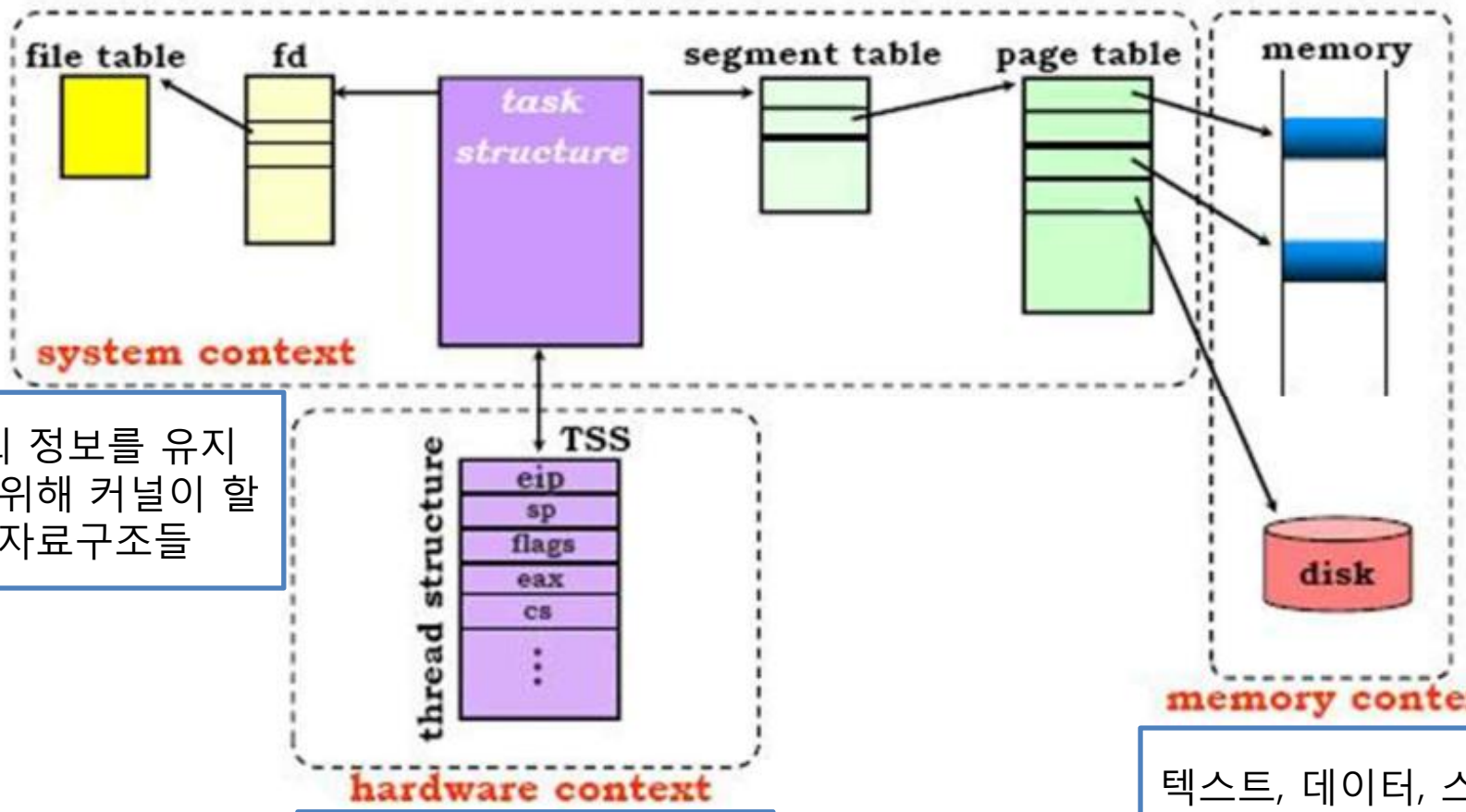
Field	Descript
thread_info	프로세스에 관한 저수준(low-level) 정보와 커널용 스택 관리
run_list	프로세스 스케줄링에 사용
array	실행 큐의 우선순위를 갖고 있는 큐
mm	프로세스 메모리 관리에 사용
pid	프로세스 ID
group_info	그룹 ID 관리
user	사용자
comm[]	실행한 명령 이름
thread	CPU 상태 저장
fs	작업 디렉토리나 루트 디렉토리에 관한 정보 관리
files	파일 디스크립터 관리
namespace	이름 공간에 관한 정보 관리
signal	시그널에 관한 정보 관리
sighand	시그널 핸들러에 관한 정보 관리

Process descriptor



Process descriptor

- context



task의 정보를 유지하기 위해 커널이 할당한 자료구조들

context switch 할 때 task의 현재 실행 위치에 대한 정보 유지

텍스트, 데이터, 스택, heap 영역, 스왑공간

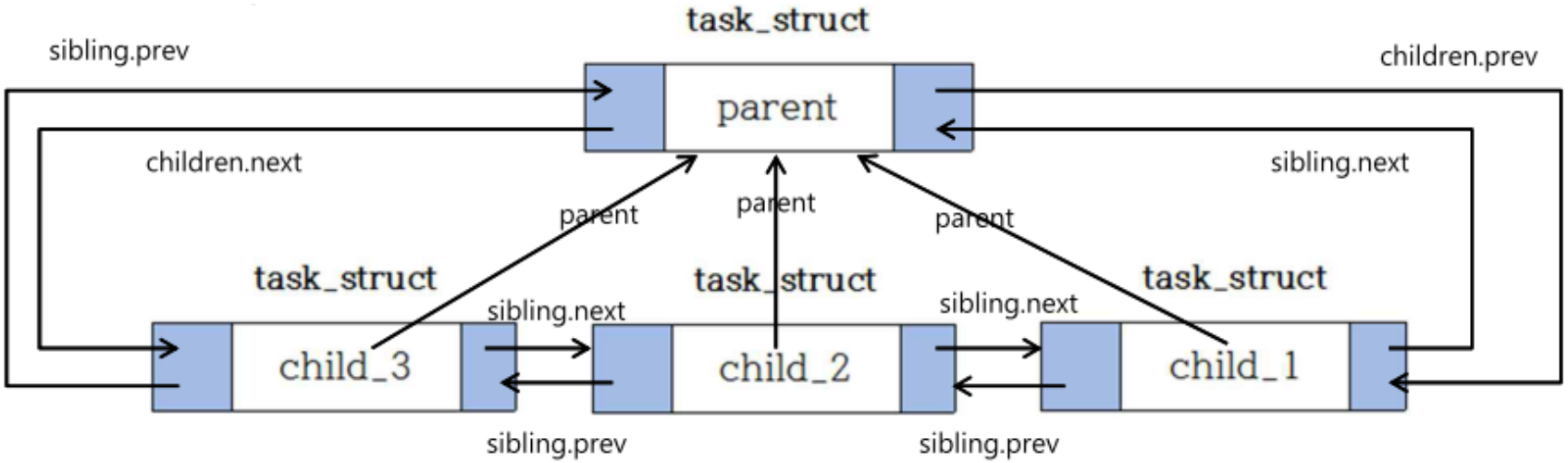
Process descriptor

- task identification
  - pid
    - task의 ID
  - tgid
    - task가 속해있는 쓰레드 그룹 ID
- task state
  - state

```
#define TASK_RUNNING            0
#define TASK_INTERRUPTIBLE      1
#define TASK_UNINTERRUPTIBLE    2
#define __TASK_STOPPED          4
#define __TASK_TRACED           8
/* in tsk->exit_state */
#define EXIT_ZOMBIE             16
#define EXIT_DEAD               32
/* in tsk->state again */
#define TASK_DEAD               64
#define TASK_WAKEKILL           128
#define TASK_WAKING             256
#define TASK_STATE_MAX          512
```

Process descriptor

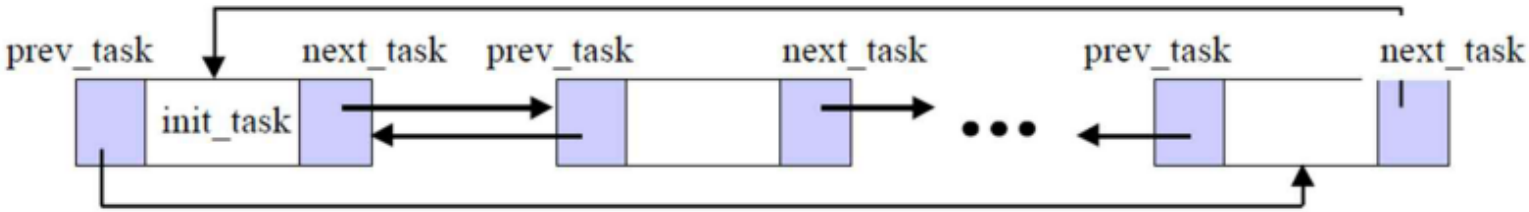
- task relationship()
  - real_parent
  - parent
  - children
  - sibling



Process descriptor

- task list
  - Circular doubly linked list
  - The head of the list is the init_task
  - list_entry : get entry

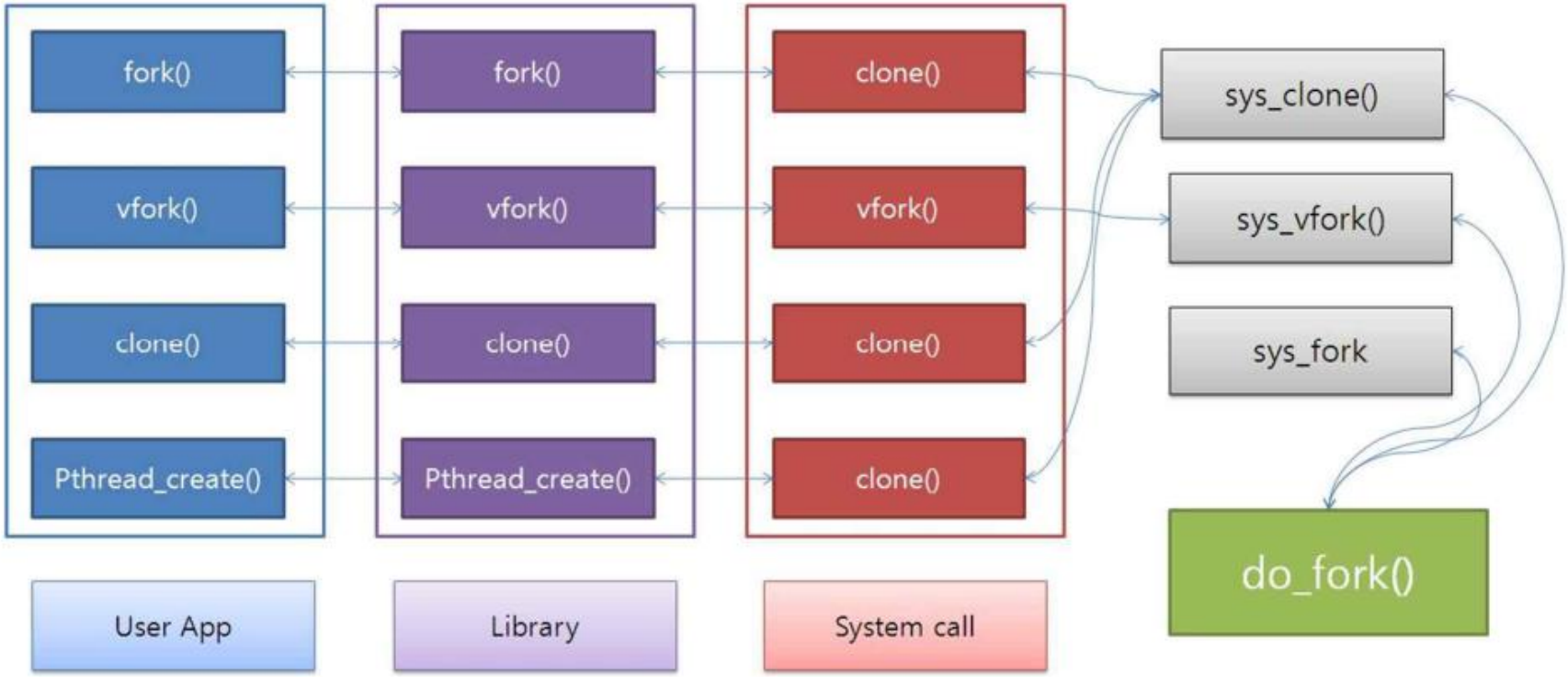
```
search = &init_task;

while(search->pid != id)
{
        search = list_entry((search)->tasks.next, struct task_struct, tasks);
        if(search->pid == init_task.pid)
        {
                printk(KERN_NOTICE "init_task\n");
                return -1;
        }
}
```



Process creation

- Process creation
  - fork()
- Executing new program
  - exec()
- Finishing program
  - exit()

Process creation



Fork()

- Create a child process

```c
#include <stdio.h>
#include <unistd.h>

int glob = 6;
char buf[] = "a write to stdout\n";

int main(int argc, char *argv[])
{
    int var;
    pid_t pid;

    var = 99;
    write(1, buf, sizeof(buf)-1);   /* stdout fd[1] */
    printf("before fork\n");

    if ((pid = fork()) == 0)    /* child */
    {
        glob++;
        var++;

        printf("child\t");
    }
    else                        /* parent, get child pid */
    {
        wait();

        printf("parent\t");
    }

    printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);

    return 0;
}
```

Exec()

```c
#include <stdio.h>
#include <unistd.h>

int glob = 6;
char buf[] = "a write to stdout\n";

int main(int argc, char *argv[])
{
    int var;
    pid_t pid;

    var = 99;
    write(1, buf, sizeof(buf)-1);   /* stdout fd[1] */
    printf("before fork\n");

    if ((pid = fork()) == 0)    /* child */
    {
        execl("/bin/ls", "ls", "-l", (char *) 0);
        glob++;
        var++;

        printf("child\t");
    }
    else                        /* parent, get child pid */
    {
        wait();

        printf("parent\t");
    }

    printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);

    return 0;
}
```

Process creation

- Create threads

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/unistd.h>

#define NUM_THREADS 2

void *print_hello(void *thread_id)
{
    int tid = (int)thread_id;

    sleep(2);

    printf("Thread [%d]'s tgid[%d] pid[%d]\n", tid, getpid(), syscall(__NR_gettid));

    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc;
    int i;
    int status;
```

Process creation

```
    for (i=0; i<NUM_THREADS; i++)
    {
        printf("Creating thread [%d]\n", i);
        rc = pthread_create(&threads[i], NULL, print_hello, (void *)i);

        if (rc)
        {
            printf(" Error : pthread_create()'s return code is %d\n", rc);
            exit(0);
        }
    }


    for (i=0; i<NUM_THREADS; i++)
    {
        rc = pthread_join(threads[i], (void **)&status);

        printf("completed join with thread [%d] status [%d]\n", i, (int)status);
    }

    printf("Exiting with completed program.\n");

    return 0;
}
```

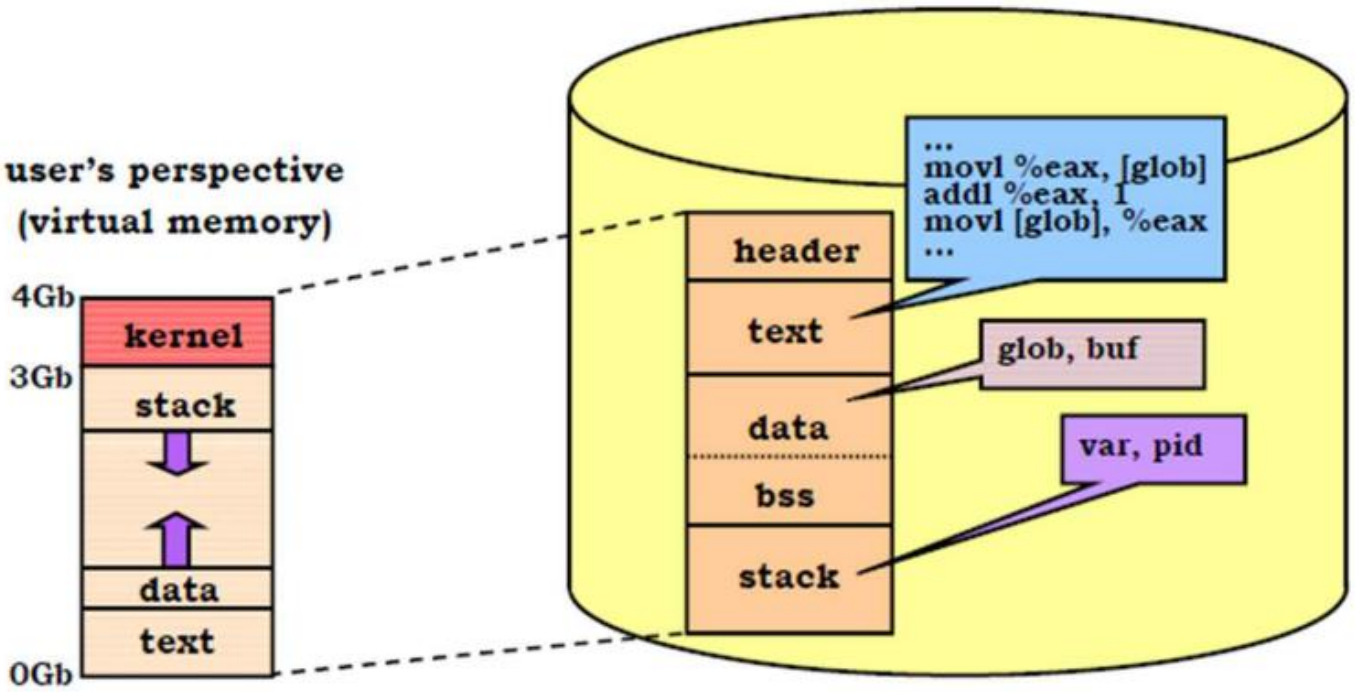
- 컴파일 시 GCC에 -lpthread 옵션 추가

  (ex : gcc -o test test.c -lpthread)

Creating Process

- 프로세스의 생성
  - 고전적인 유닉스 환경
    * 부모 프로세스의 모든 정보를 그대로 복사
    * 새로운 프로세스에 맞게 필요한 데이터들을 수정
  - 현재 유닉스 환경
    * Copy On Write (COW)를 이용한 복사의 최소화
    * Lightweight Process를 통한 커널자료구조의 공유
    * vfork()를 이용한 메모리 주소 공간 공유

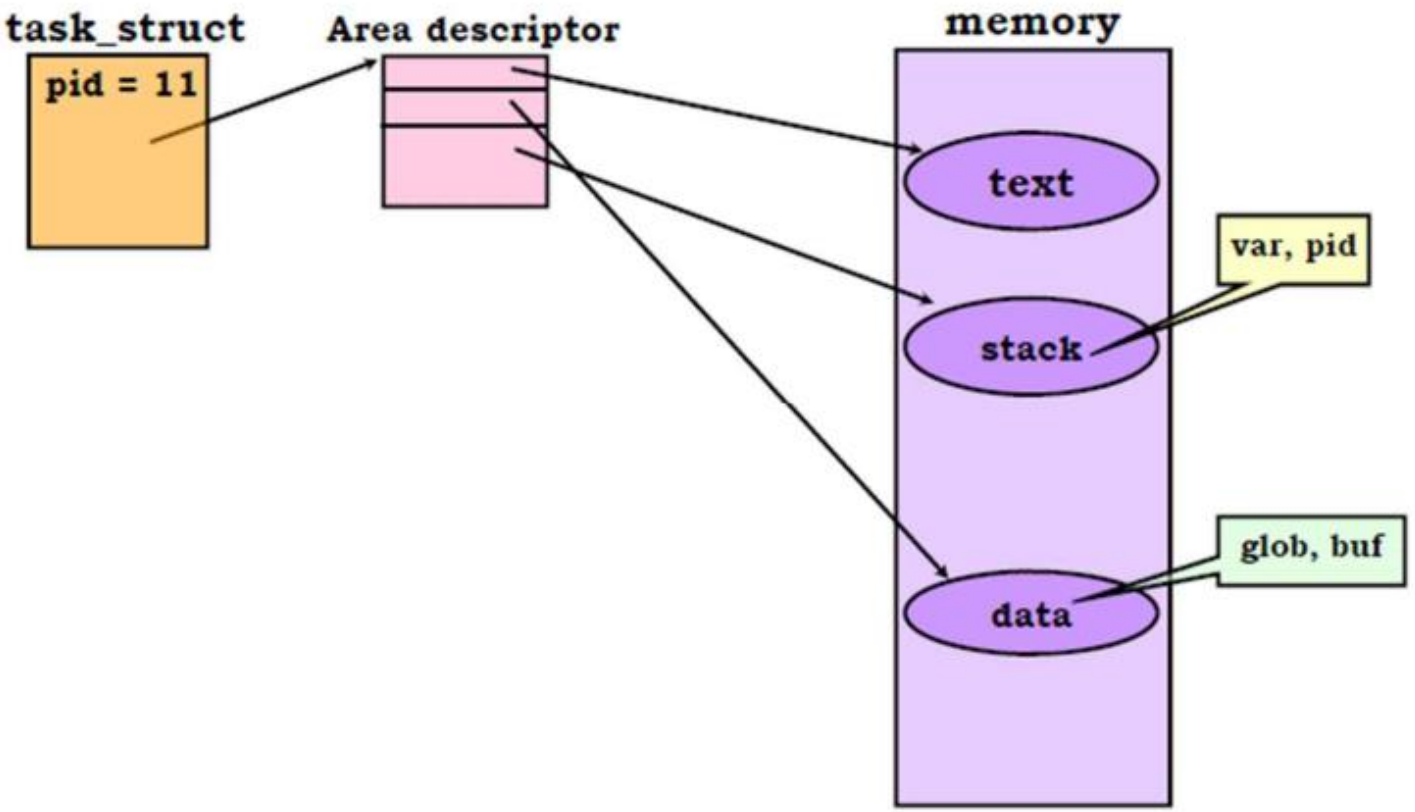
Process creation



Linux One © 2002

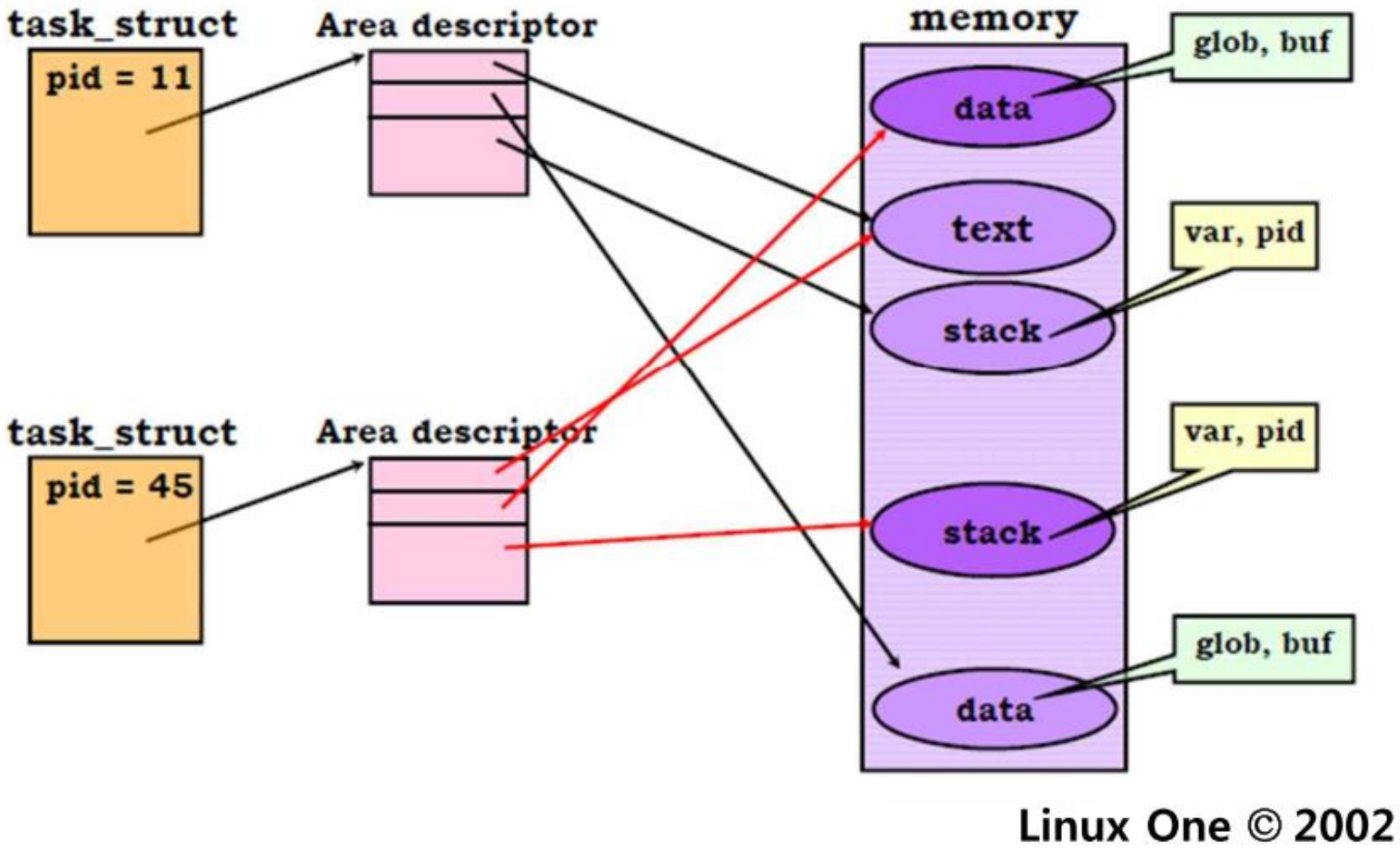
Process creation

- Before fork()



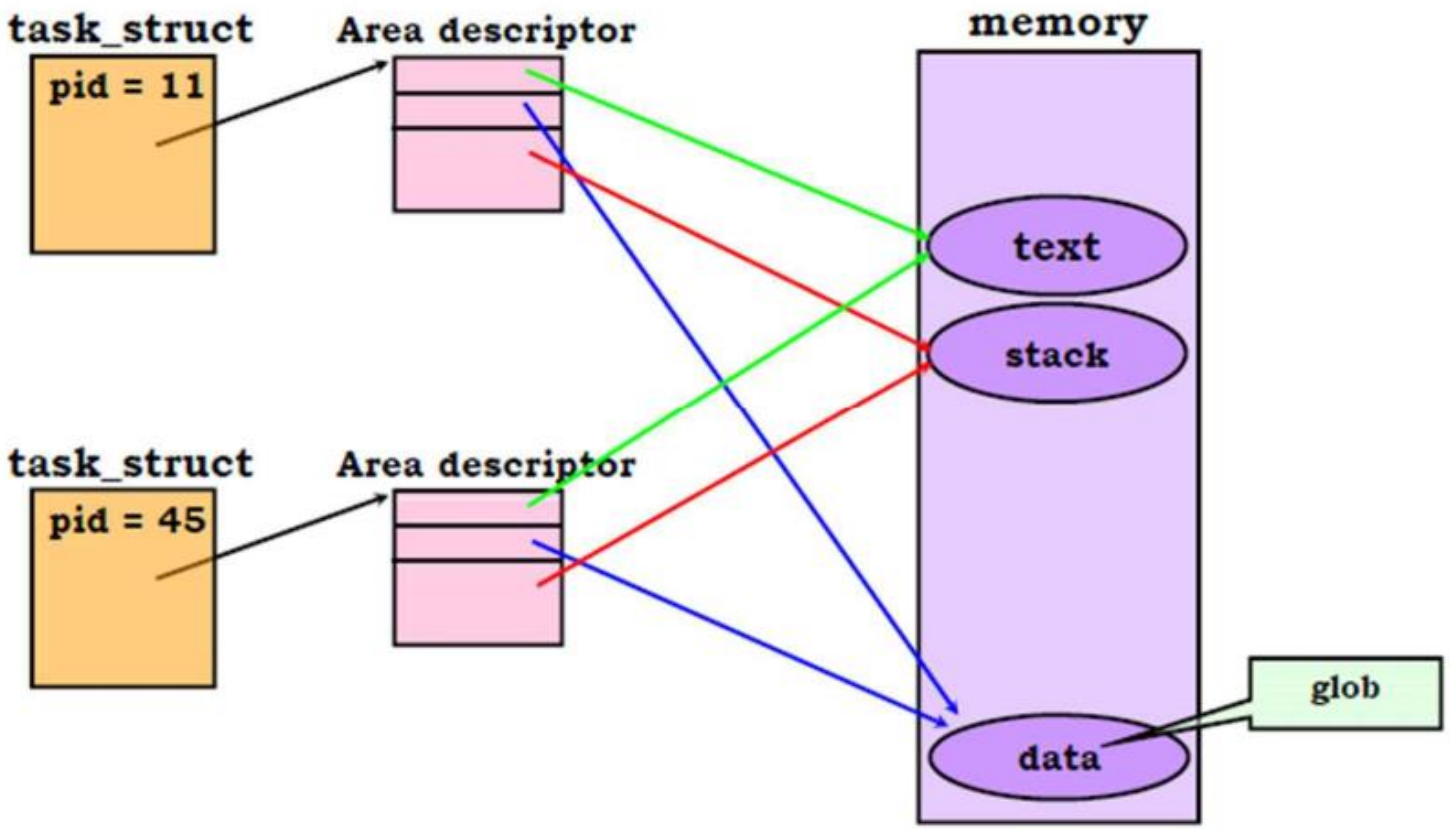
Process creation

- After fork() – prev version



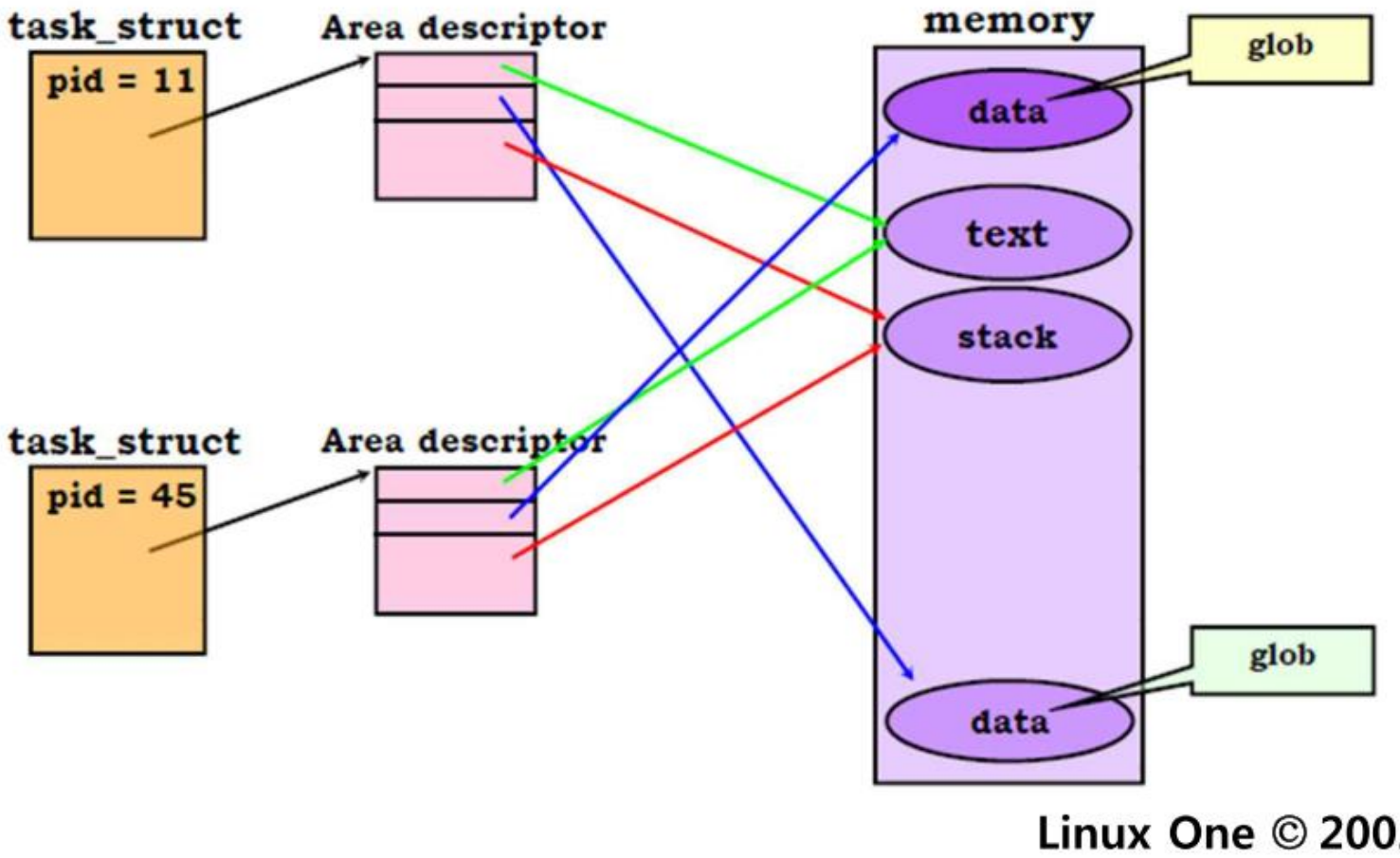
Process creation

- After fork() – COW 1



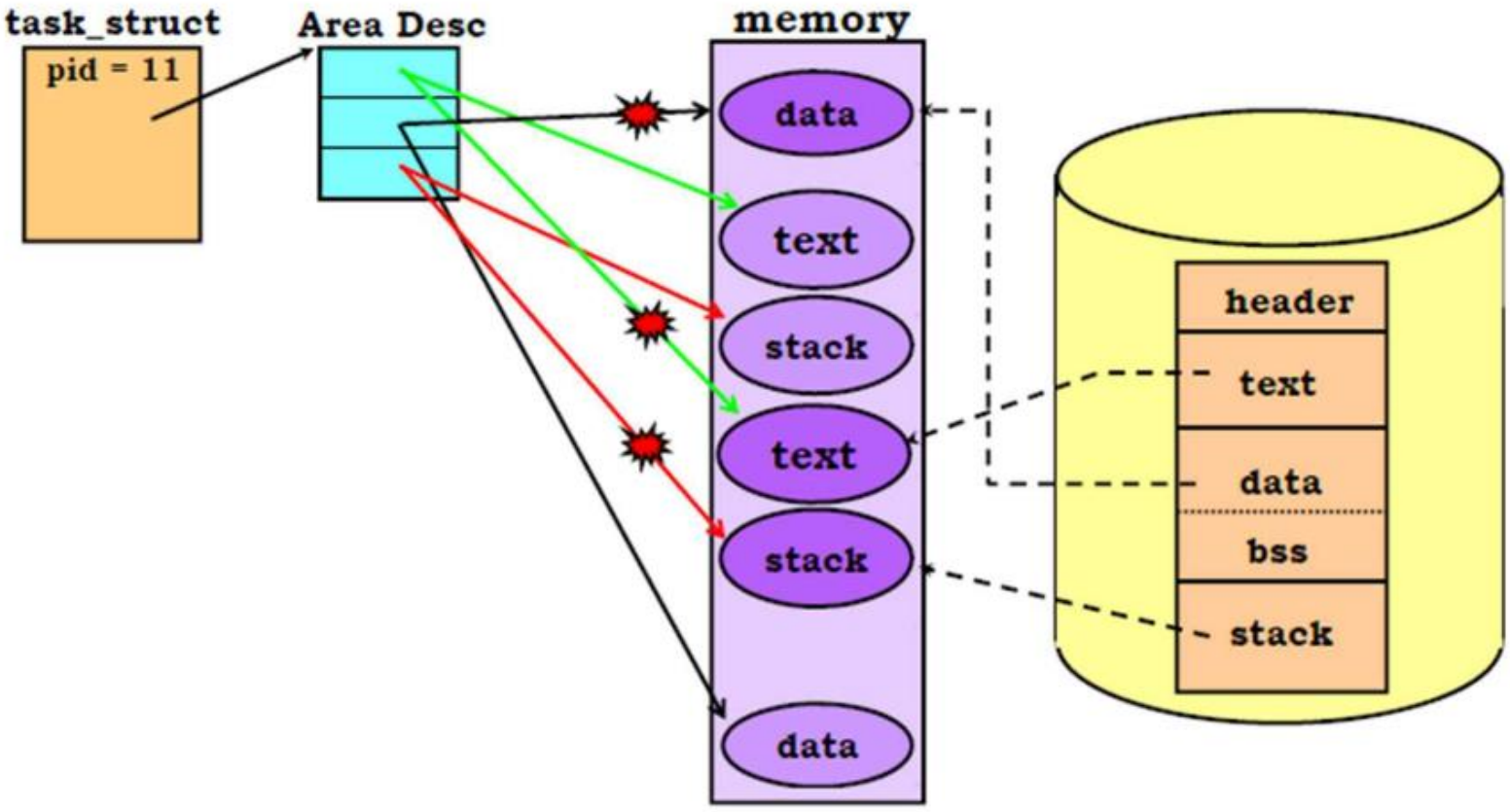
Process creation

- After fork() – COW 2



Process creation

- After execve()



Linux One © 2002