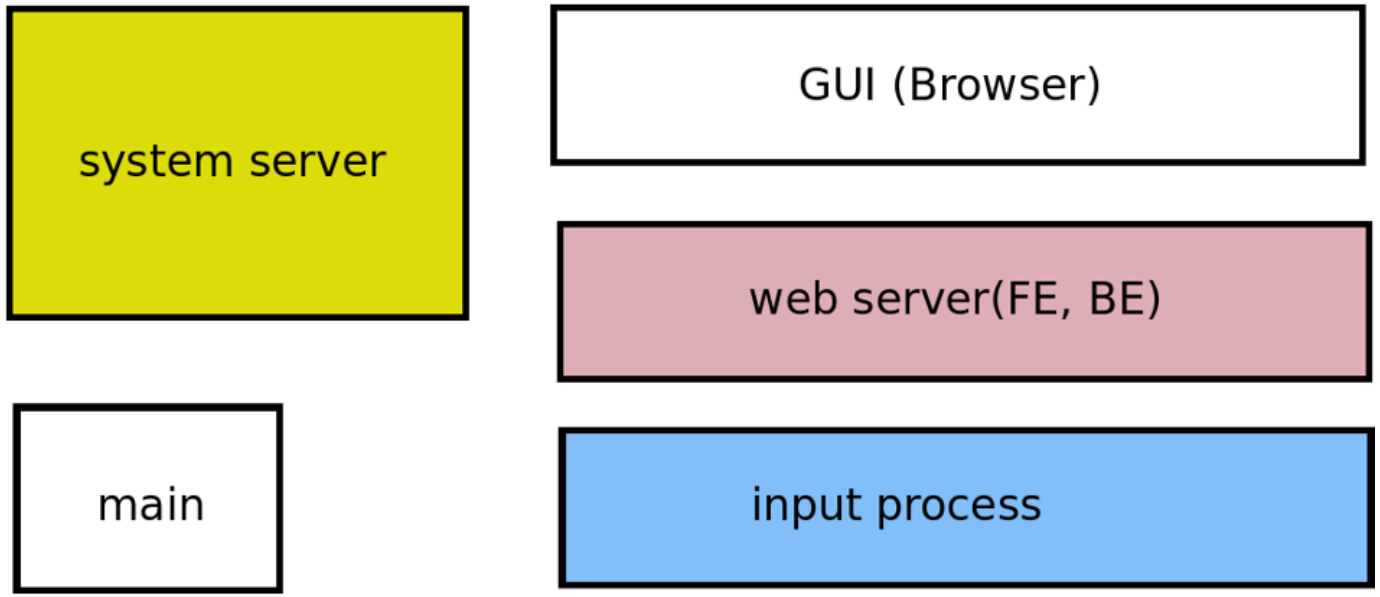
스레드

지난 토이 프로젝트 과제(타이머)



리눅스 OS를 공부하면서..

- **OS가 없다면?**
- **리눅스가 어떻게 문제를 해결?**
  - **격리(Isolation)**
  - 하드웨어 추상화
  - 다중화(Multiplex)
  - 공유(Sharing)
  - 보안(Security)
  - 성능(Performance)

- **SW 복잡성을 어떻게 해결?**
  - **모듈화, 추상화, 레이어링, 계층화(Hierachial)**

프로세스 복습

- 프로세스
  - 격리(isolation)를 위해 만들어짐 , 일부 OS는 프로세스가 없음.
    - 예: RTOS
  - 추상화된 가상 머신 (하지만, 실제 가상 머신 아님)
    - 더 확장된 가상 머신은 리눅스 컨테이너
  - 실행중인 프로그램
  - 마치 자신이 CPU와 메모리를 모두 소유한 것 처럼 보임
  - 다른 프로세스의 영향을 받지 않음
    - 다른 프로세스가 죽어도 영향 없음

- API: fork exec exit wait kill getpid

프로세스 - CPU 격리(Isolation)

- 어떻게?
  - H/W 주기적으로 "clock interrupt"를 제공
    - 강제적으로 현재 프로세스를 중단
    - 커널로 진입
    - 커널은 다른 프로세스로 변경
- 왜?
  - 긴 연산하는 프로세스를 막기 위해
  - 또는, 이상한 무한 루프 코드 방지
- 커널은 반드시 현재의 프로세스 상태(registers)를 저장/복구
  - 문맥교환(Context switch)

프로세스 - 메모리 격리(Isolation)

- 주소 공간(Address Space)
  - code, variables, heap, stack
- 커널 또는 다른 프로세스가 접근하는 것 방지
- 이건 또 어떻게?
  - "페이징 하드웨어"를 통해
  - 또는 MMU라고 함.
    - memory management unit
- 자세한 내용은 메모리 수업 시간에..

스레드란?

- CPU 코어에서 실제로 실행되는 코드
- 1개의 프로세스는 적어도 1개 이상(main 스레드 + 알파)의 스레드를 가짐
- 그럼 실행되지 않은 스레드는?
  - 스택에 저장됨
  - 레지스터
- 특징은 스레드 간에는 메모리를 공유한다.
  - 프로세스 주소 공간을 공유

스레드 특징

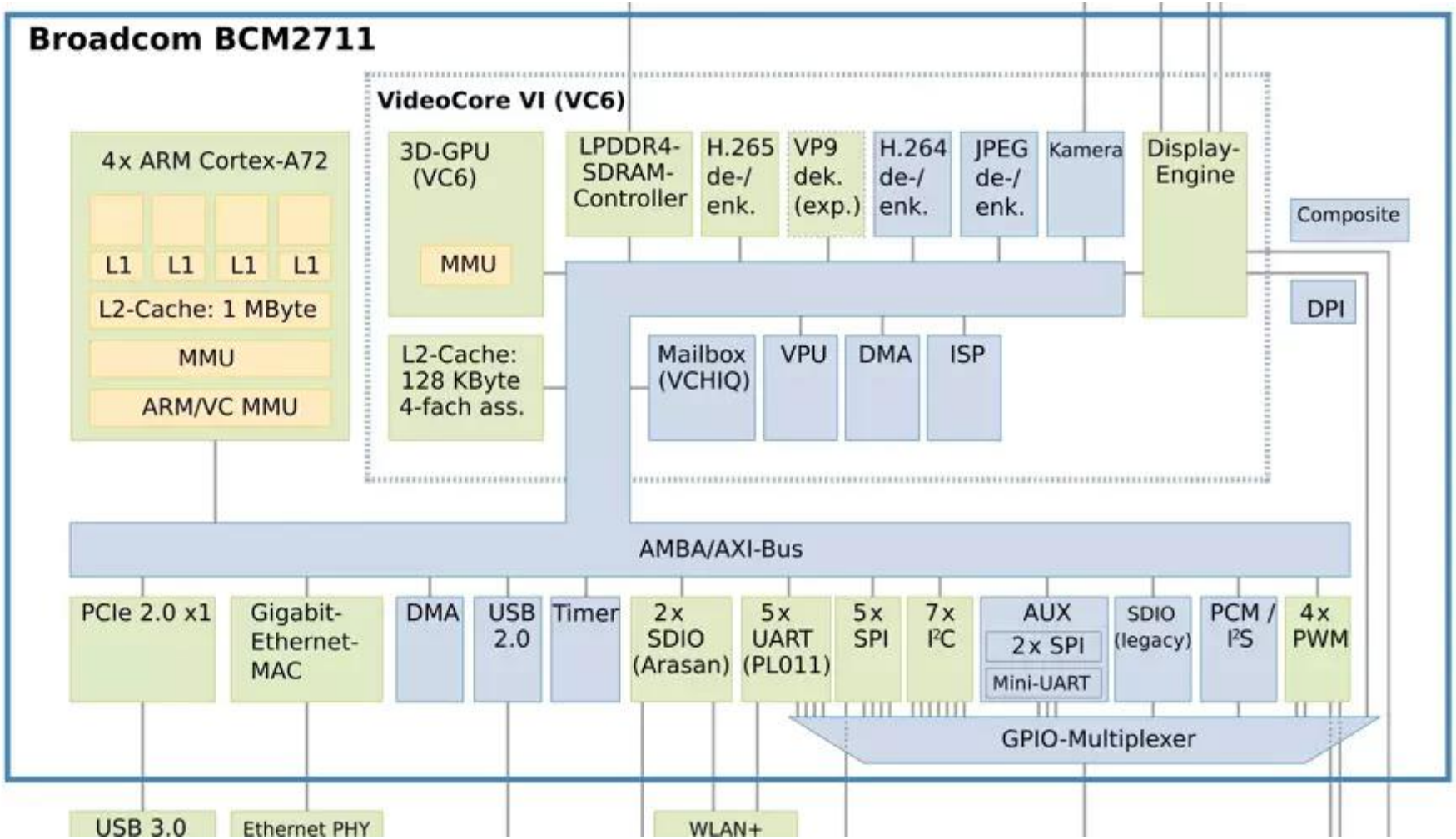
- 스레드는 실행하거나 실행하지 않거나..

- 만약 실행 중이면
  - 스레드는 CPU를 사용한다.
    - 레지스터들 이용(스택 포인터를 이용해 스택 메모리 사용)

- 실행 중이 아니면
  - 용어가 다양함
  - 어디는 "suspended"
  - 누구는 "blocked" 또는 "waiting"
  - 레지스터들은 메모리 어딘가 저장됨

- 스레드가 대기 상태에서 실행되면 저장된 레지스터들이 복구

참고: 멀티 코어 라즈베리파이4 SoC(BCM2711)



프로세스와 스레드

- 프로세스 = 독립적인 주소 공간 + 여러 스레드
  - 독립적인 주소 공간(page tables 관리)

- 스레드 = 독립적으로 실행
  - 레지스터, PC(Program Counter), 스택
  - 일반적으로 CPU보다 많은 스레드들이 동작

리눅스 vs RTOS

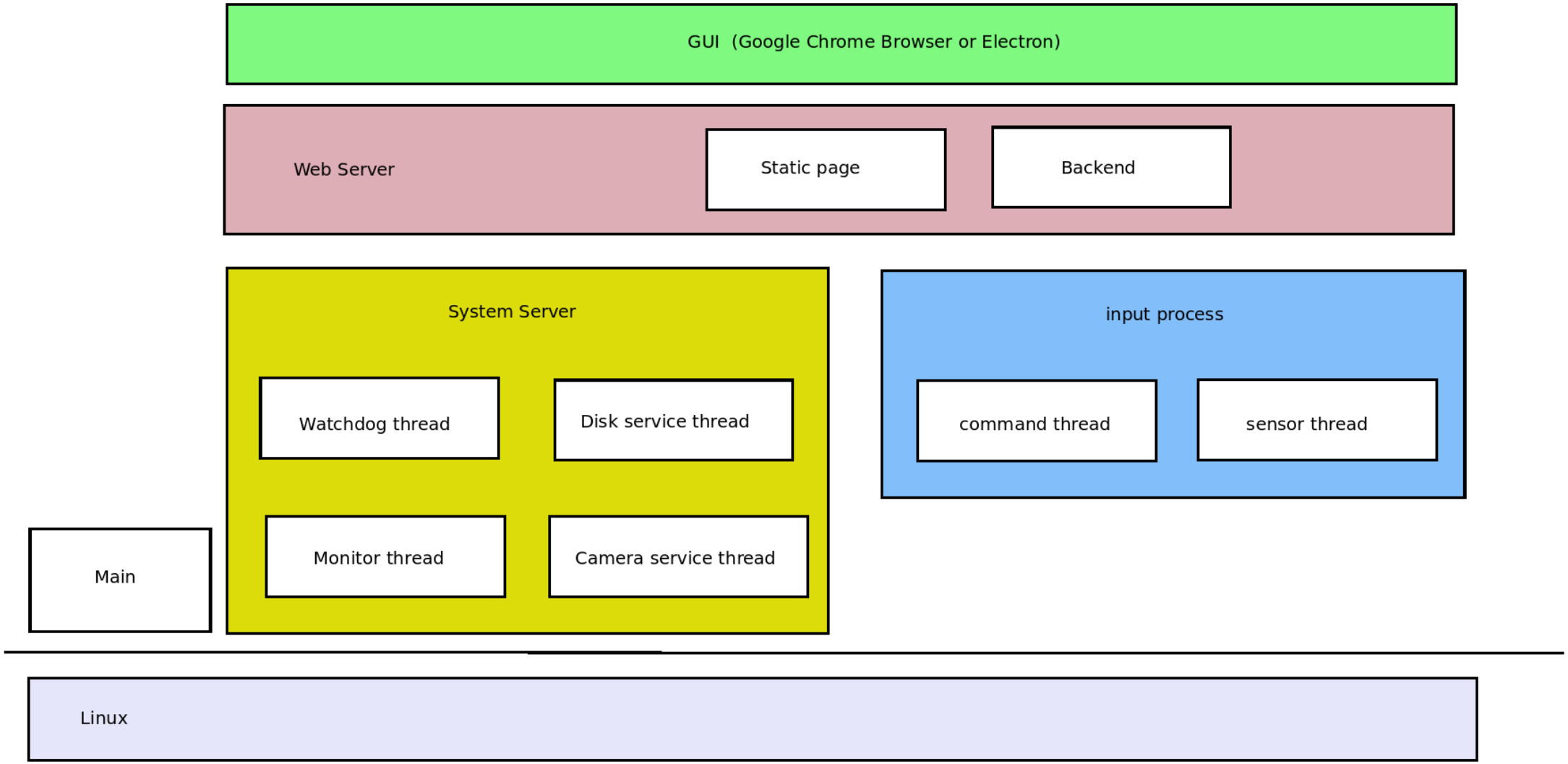
- 리눅스
  - 프로세스 기반
  - 프로세스간 메모리 침범 X

- RTOS(Real-time OS)
  - 스레드로만 동작
  - 모든 메모리 공유

스레드 스위칭 흘끔 보기

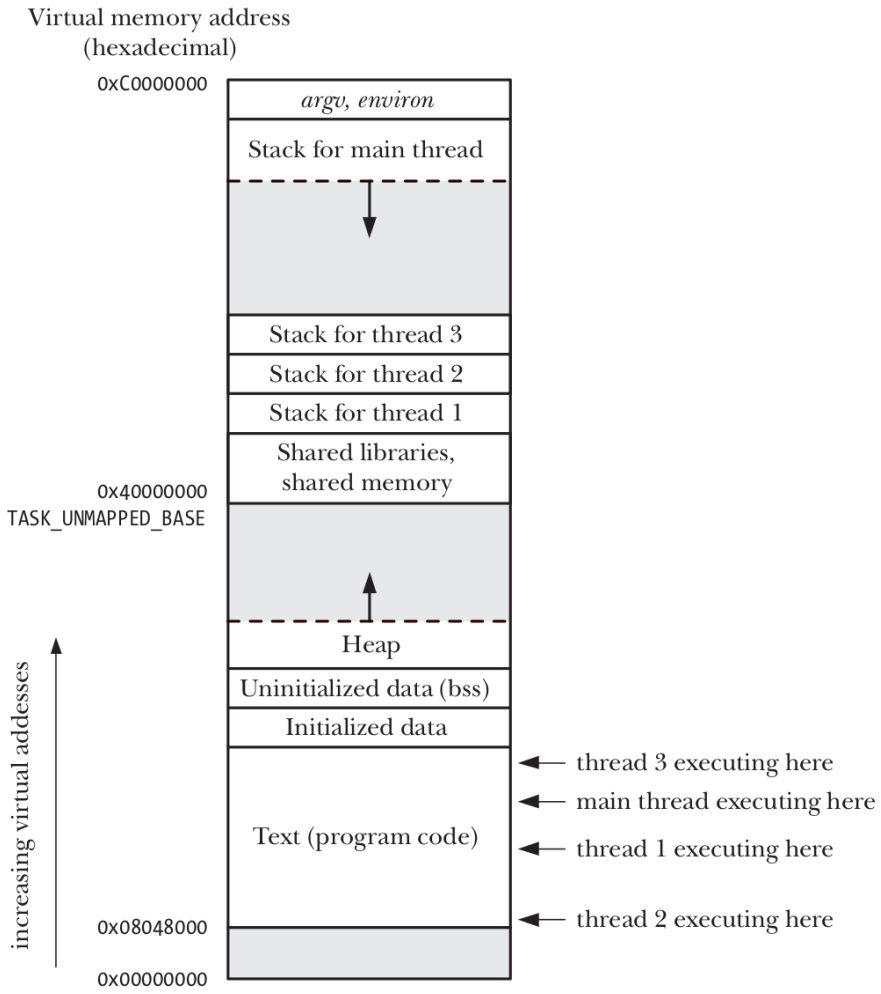
- user -> kernel (시스템 콜 호출 또는 타이머 인터럽트)
- kernel -> scheduler
- scheduler는 사용 가능한 스레드 찾음
- scheduler -> kernel
- kernel -> user
- 자세한 내용 -> 리눅스 커널 수업

멀티 스레드를 추가한 토이 프로젝트



프로세스 메모리 맵 - 스레드 포함

- 4 스레드 메모리 맵
- /proc/<pid>/map



Pthreads API

Table 29-1: Pthreads data types

Data type	Description
pthread_t	Thread identifier
pthread_mutex_t	Mutex
pthread_mutexattr_t	Mutex attributes object
pthread_cond_t	Condition variable
pthread_condattr_t	Condition variable attributes object
pthread_key_t	Key for thread-specific data
pthread_once_t	One-time initialization control context
pthread_attr_t	Thread attributes object

스레드 생성

The *pthread_create()* function creates a new thread.

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start)(void *), void *arg);
```

Returns 0 on success, or a positive error number on error

스레드 종료

```
include <pthread.h>

void pthread_exit(void *retval);
```

종료된 스레드 조인

```
include <pthread.h>

int pthread_join(pthread_t thread, void **retval);
                                    Returns 0 on success, or a positive error number on error
```

simple_thread

Listing 29-1: A simple program using Pthreads

threads/simple_thread.c

```
#include <pthread.h>
#include "tlpi_hdr.h"

static void *
threadFunc(void *arg)
{
    char *s = (char *) arg;

    printf("%s", s);

    return (void *) strlen(s);
}

int
main(int argc, char *argv[])
{
    pthread_t t1;
    void *res;
    int s;

    s = pthread_create(&t1, NULL, threadFunc, "Hello world\n");
    if (s != 0)
        errExitEN(s, "pthread_create");

    printf("Message from main()\n");
    s = pthread_join(t1, &res);
    if (s != 0)
        errExitEN(s, "pthread_join");

    printf("Thread returned %ld\n", (long) res);

    exit(EXIT_SUCCESS);
}
```

threads/simple_thread.c

```
$ ./simple_thread
Message from main()
Hello world
Thread returned 12
```

Detaching a Thread

- 언제 사용?

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

Returns 0 on success, or a positive error number on error

```
pthread_detach(pthread_self());
```

스레드 속성

Listing 29-2: Creating a thread with the detached attribute

from **threads/detached_attrib.c**

```
pthread_t thr;
pthread_attr_t attr;
int s;

s = pthread_attr_init(&attr);               /* Assigns default values */
if (s != 0)
    errExitEN(s, "pthread_attr_init");

s = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
if (s != 0)
    errExitEN(s, "pthread_attr_setdetachstate");

s = pthread_create(&thr, &attr, threadFunc, (void *) 1);
if (s != 0)
    errExitEN(s, "pthread_create");

s = pthread_attr_destroy(&attr);            /* No longer needed */
if (s != 0)
    errExitEN(s, "pthread_attr_destroy");
```

from **threads/detached_attrib.c**

멀티 스레드 디버깅

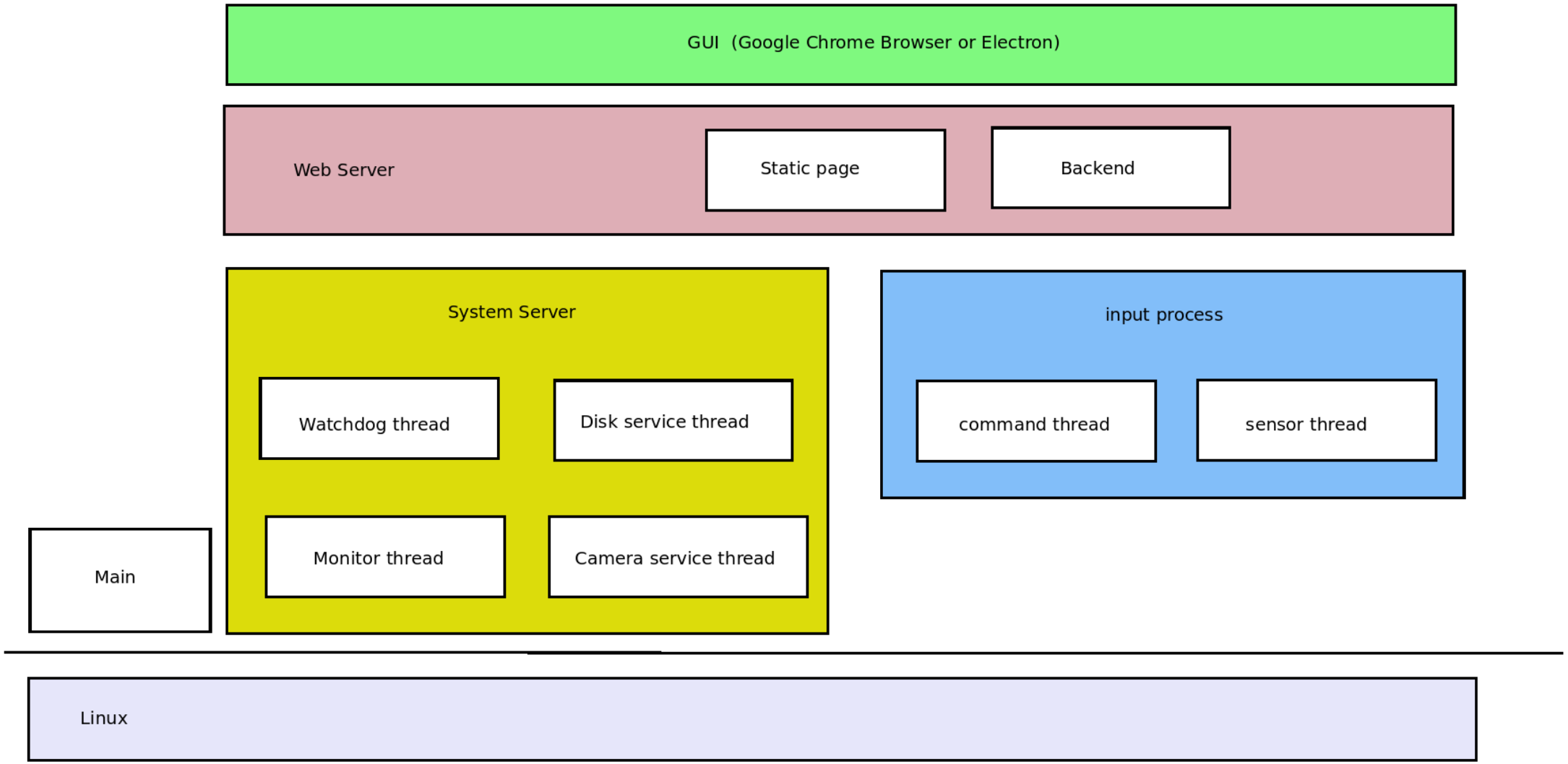
- GDB
  - Info thread
  - thread <number>

- VSCODE
  - 좌측 thread 디버그 창 참조

실습 코드 분석

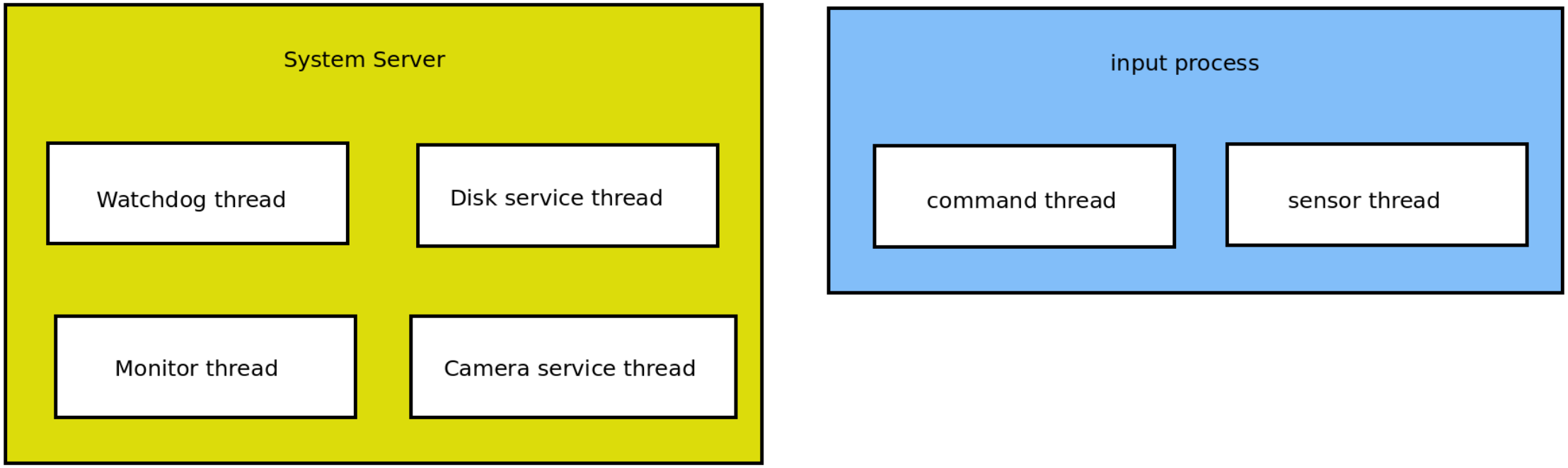
- vscode debugger로 디버깅
- tlpi-dist/threads/simple_thread.c
  - 코드 분석 및 실행
- tlpi-dist/threads/detached_attrib.c
  - 코드 분석 및 실행

토이 프로젝트 - 멀티 스레드 생성 과제



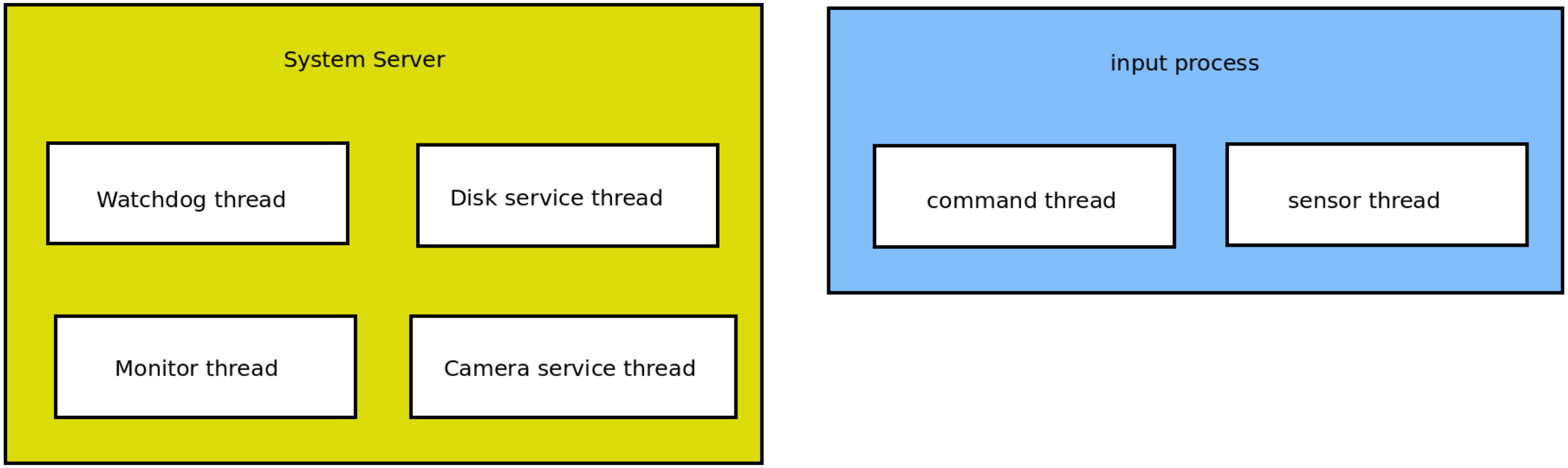
System server와 input process 확장

- 6개 thread 추가
  - Watchdog, disk service, monitor, camera service, command, sensor



Command thread

- 키보드 입력을 받아서 등록된 함수 수행하도록 수정.
- 향후 메시지 테스트 용도로 활용.



토이 프로젝트 - 스레드 생성

- 다른 스레드는 아무일 안하고 loop 대기

```
while (1)
    sleep(1);
```

- Command thread는 toy_loop 함수 호출