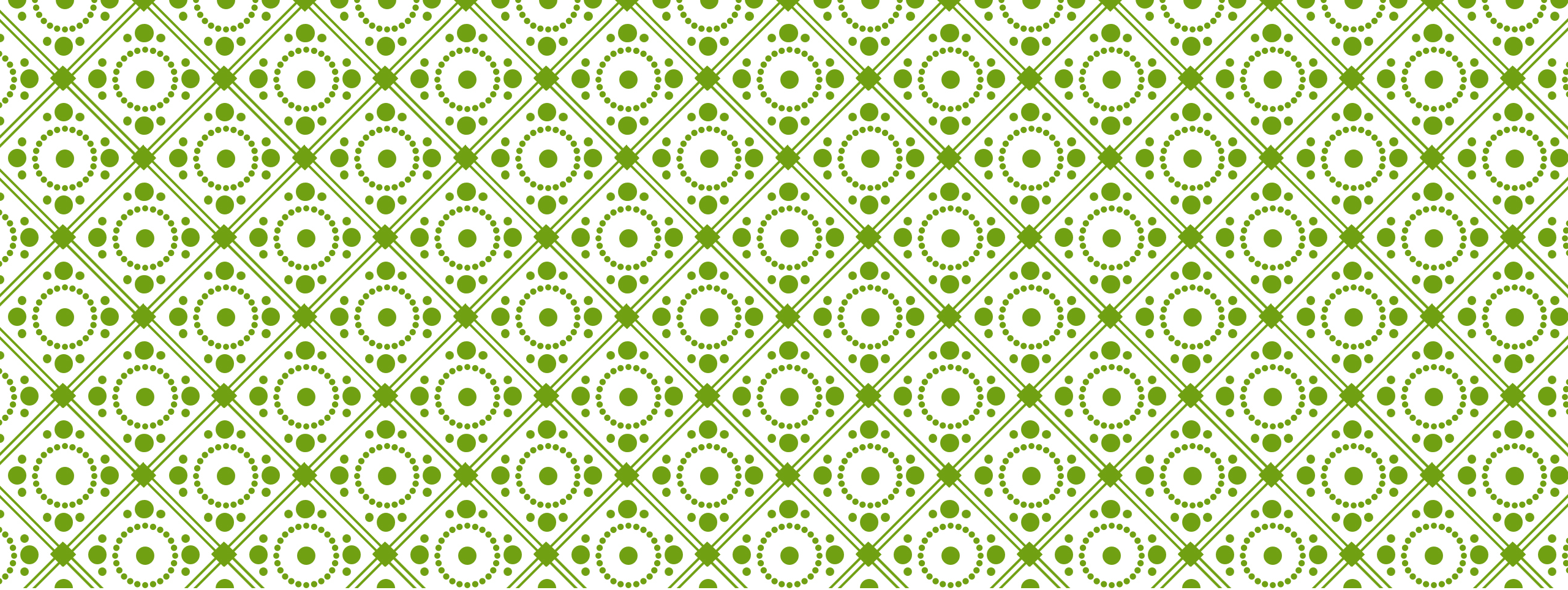


ПРОГРАМИРАЊЕ У РТ ЛИНУКСУ

Програмирање у реалном времену
Вјежба 9

УВОД

Потребно – печиран Линукс

Како смањити латентност закључавањем меморије?

Утицај дијељених ресурса програмских нити на приоритете процеса у реалном времену.

ЛАТЕНТНОСТ

Можда и највећи губитак времена у програмирању у реалном времену настаје због чињенице да је меморија процеса, који је управо добио процесорско вријеме, НА ДИСКУ!

- Постоје двије врсте *pagefault-a*:
  - *Major*, приликом приступа улазно-излазној периферији. Изазива ОГРОМНУ латентност.
  - *Minor*, без I/O приступа. Не изазива велику латентност.

Закључати меморију у радној меморији у довољној количини да се пејџфолтови више не дешавају. Ово се може овезбиједити ако:

- У `main()` функцији позивамо `mlockall()`
- Не креирамо програмске нити динамички.
- Резервишемо довољну количину меморије и за стек програмских нити.

ЗАКЉУЧАВАЊЕ МЕМОРИЈЕ

`mlockall(MCL_CURRENT | MCL_FUTURE )`

Гура у радну меморију све што се тренутно налази у

- код, дата и стек сегменту.
- Дијељене библиотеке
- Кориснички простор који комуницира са кернелом
- Дијељену меморију
- Мапиране фајлове

Све остаје до даљњег у RAM меморији!

`MCL_FUTURE` – значи да ће и све будуће алоциране ствари бити у радној меморији

`staticlock.c`

ДИНАМИЧКО АЛОЦИРАЊЕ ПРИ ЗАКЉУЧАНОЈ МЕМОРИЈИ

Да би могли динамички да алоцирамо меморију, морамо да контролишемо системске позиве, а то су `sbrk()` и `mmap()`.

Пошто се `mmap()` не може контролисати, било би добро га онемогућити га. Функција `sbrk()` подиже и спушта своју адресу по меморији. Било би добро да не може да спушта!

Кључна фукнција:

- **`malopt()`**
- http://man7.org/linux/man-pages/man3/mallopt.3.html
- **`mallopt (M_TRIM_THRESHOLD, -1); - онемогућавање спуштања sbrk()`**
- **`mallopt (M_MMAP_MAX, 0); - онемогућавање позива mmap()`**

`dynamiclock.c`

ПРОГРАМСКЕ НИТИ У РЕАЛНОМ ВРЕМЕНУ

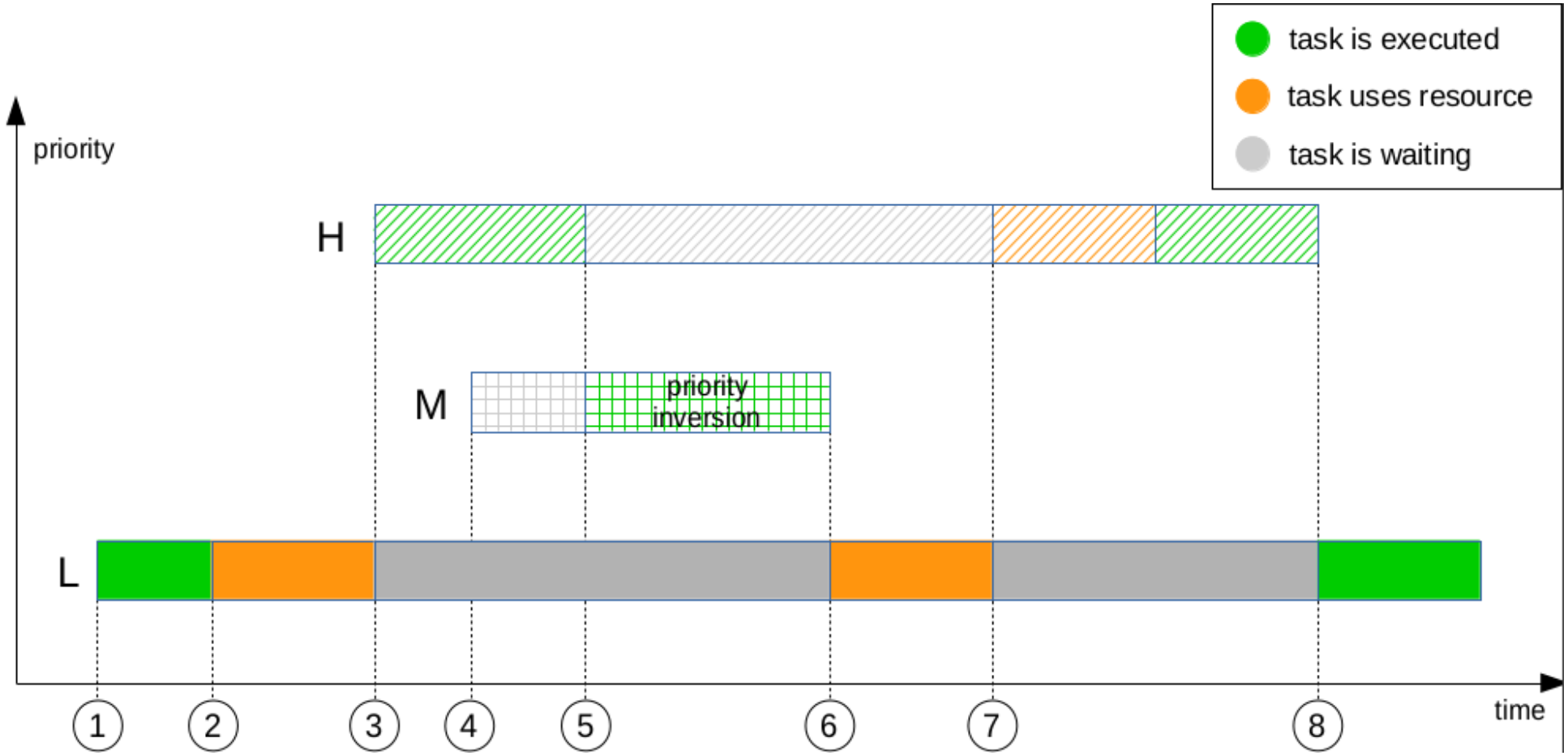
Ствар код програмских нити јесте да имају свој посебан стек, тако да морамо:

- Креирати све потребне програмске нити прије него што радимо нешто што је рад у реалном времену.
- Због оног `MCL_FUTURE` параметра све што се алоцира у програмској нити биће у радној меморији, па и стек програмске нити, уобичајене величине 8 мегабајта.
- Али за разлику од нормалне ситуације у којој стек програмске нити може да се шири, ми смо то све забранили, па морамо да УРАЧУНАМО колико ће нам још требати за стек!

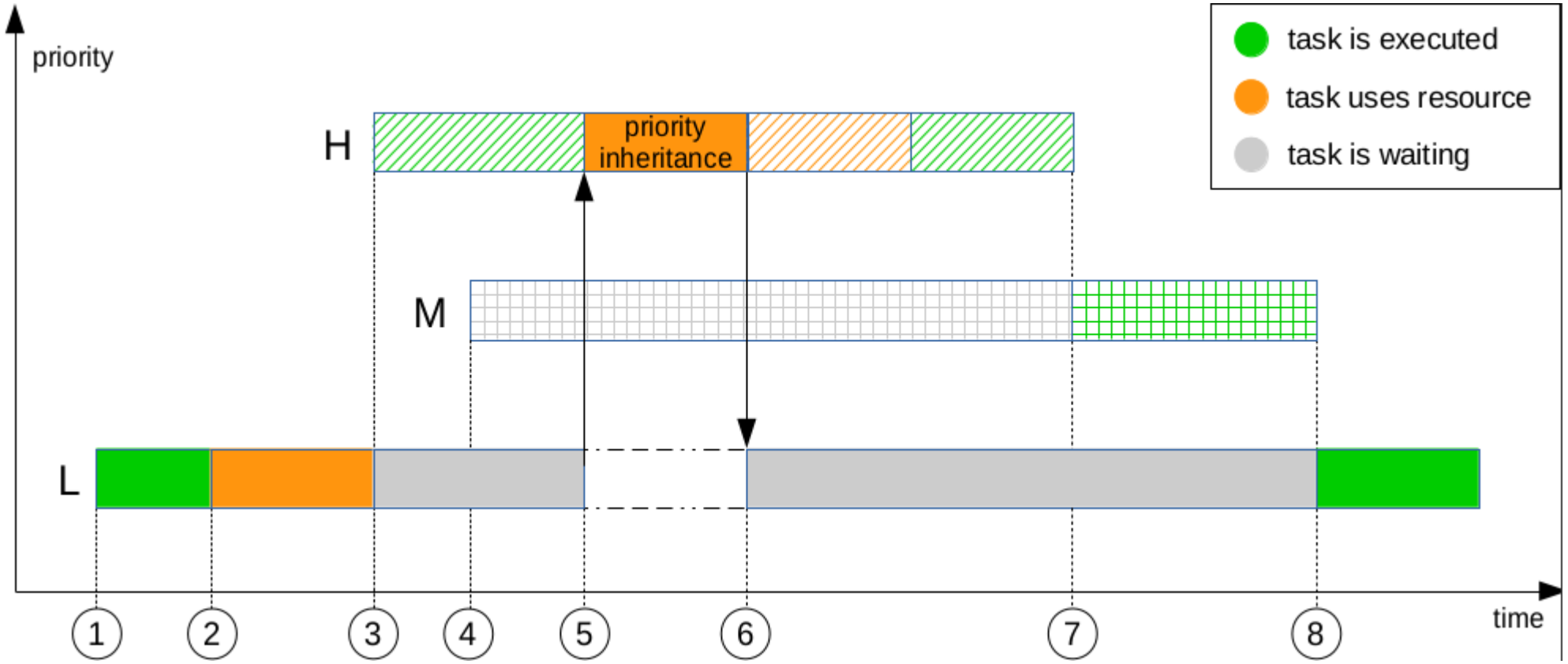
`thread_template.c`

Међутим, проблеми са програмским нитима не престају ту!

ИНВЕРЗИЈА ПРИОРИТЕТА



НАСЉЕЂИВАЊЕ ПРИОРИТЕТА



ЗАДАТАК

Прерадити `thread_template.c` тако да:

- Постоје двије функције за програмску нит, направљене на основу постојеће `my_rt_thread`:
  - `resource_thread_fn()` са промјењивим параметрима
    - Секунди колико ће спавати
    - Приоритета који ће бити
    - Додатне величине стека
    - Параметра који говори да ли ће се постојећи птринови на екрану исписивати или не
    - Цијелог броја којим ће увећати дијељени ресурс (`static int shared_val = 0;`)
      - Наравно да ће овдје бити мутекс!
  - `non_res_thread_fn()` која има све исто осим броја којим увећава дијељени ресурс, јер ова нит не користи ништа дијељено
- Функција `start_rt_thread` враћа креирану програмску нит, а прима параметре на основу којих може да креира програмске нити које користе горе наведене функције.

Направити три програмске нити које имају проблем са инверзијом приоритета

АКО СЕ НИШТА НЕ ПРИМЈЕТИ

Или нису добро постављене програмске нити

Или је мутекс иницијално постављен да поштује насљеђивање приоритета

PTHREAD_PRIO_INHERIT

```
static pthread_mutexattr_t mtx_attr;
....
// u main()
pthread_mutexattr_init(&mtx_attr);
pthread_mutexattr_setprotocol(&mtx_attr, PTHREAD_PRIO_PROTECT);
pthread_mutex_init(&mtx, &mtx_attr);
....
pthread_mutexattr_destroy(&mtx_attr); // prije unistavanja mutexa
```