2. Roosteren

In deze opdracht ga je aan de slag om een preemptive scheduler te maken. Hierbij moet je gebruik gaan maken van de system timer die de Raspberry Pi biedt.

Dit is een grotere opdracht dan de vorige, waarbij de eerste inschatting is dat je er ongeveer 2 tot 3 keer zoveel aan kwijt bent.

Voorbereiding

0. Haal de laatste versie van FramboOS van GitHub. Dit kun je doen door in je bestaande repository `git pull` uit te voeren. Los eventuele merge conflicten op en controleer of je kernel nog steeds compileert en draait.

Opzet

In `task.h` vind je de datastructuur voor een *Task Control Block* (TCB), het equivalent voor een *Process Control Block* (PCB) in FramboOS:

```
typedef size_t tid_t;

typedef struct {
  saved_cpu_state_t saved_state;
  char *task_name;
  tid_t tid;
  bool in_use;
} task_t;
```

Het bevat de opgeslagen CPU toestand, de naam van de taak, een taak identifier en een flag of de taak in gebruik is.

In `scheduler.h` vind je het maximum aantal taken dat FramboOS kan besturen:

```
#define MAX_TASKS 1024
```

Verder bevat het de signaturen voor procedures om taken toe te voegen en te verwijderen bij de scheduler, en een procedure om het TCB van de huidige taak te verkrijgen:

```
tid_t scheduler_task_add(char *task_name, task_main_f f, void *stack_pointer);
void scheduler_task_remove(tid_t tid);
task_t *scheduler_current_task();
```

Tot slot zijn er procedures om de scheduler te initialiseren, te starten en, de belangrijkste, om een task switch uit te voeren:

```
void scheduler_init();
void scheduler_start();
void scheduler_task_switch(saved_cpu_state_t *last_state);
```

Deze procedures gaan we allemaal implementeren.

Hulpprocedures

Neem een kijkje in `scheduler.c`. Daarin vind je de toestand van de scheduler, opgeslagen in variabelen die globaal zijn voor dit bestand. (Dat is wat `static` aangeeft: het zijn globale variablen, maar ze zijn alleen zichtbaar binnen dit `.c` bestand.)

```
// The array containing the information of all running and terminated tasks.
// The whole piece of memory is initialised with 0.
static task_t tasks[MAX_TASKS] = {0};
// The `current_task_index` represents the index into the `tasks` array.
// The init task will be placed at index 0.
```

```
static size_t current_task_index = 0;
// A counter for the next task identifier.
// The init task will have tid #1.
static tid_t next_tid = 1;
```

Allereerst schrijven we drie hulpprocedures die handig zijn bij de volgende onderdelen.

1. Schrijf een implementatie voor `task_t *scheduler_current_task()`, die een pointer naar de TCB van de huidige taak teruggeeft.
2. `size_t find_next_active_task_index()` zoekt in de `tasks` array de eerst volgende taak *na de huidige* die actief is, dat wil zeggen, waarbij het `in_use` veld `true` is. Let er hierbij op dat je verder zoekt bij het begin wanneer je het einde van de array hebt bereikt. Daarnaast moet je niet een oneindige loop bouwen! Geef een `kernel_panic` wanneer er geen actieve taak is.
3. `size_t find_first_inactive_task_slot()` zoekt in de `tasks` array de eerst mogelijke vrije plek om een nieuwe taak in te voegen. Met andere woorden, het `in_use` veld moet `false` zijn. Aangezien we het gehele blok geheugen van de `tasks` array hebben geinitialiseerd op `{0}`, zijn alle slots in eerste instantie inactief. Bij de initialisatie van de scheduler zullen we het eerste slot vullen.
4. `void terminate_current_task()` termineert de huidige taak door zijn `in_use` veld op `false` te zetten. **Belangrijk!** Eindig deze procedure met een oneindige loop!

Maak bij het schrijven van deze procedures goed gebruik van de log functionaliteit uit `kernel/hardware/uart.h`, zodat je via de console informatie kunt krijgen over wat er gebeurt.

Taken toevoegen en verwijderen

Bekijk `tid_t scheduler_task_add(char *, task_main_f, void *)`. Hierin wordt achtereenvolgens:

- het eerst volgende inactieve slot gezocht;
- de volgende task identifier bepaald;
- in het gevonden inactieve slot een nieuw TCB geplaatst met daarin de opgeslagen CPU toestand;
- de index van de taak geretourneerd.

5. Bekijk goed welke informatie van een taak wordt geïnitialiseerd in het TCB. Beschrijf voor de registers SP, PC, en LR waarvoor ze gebruikt worden. (De CPSR mag je overslaan, dit is een ARM implementatie detail). Geef aan waarom deze geïnitialiseerd moeten worden op een niet-nul waarde in het TCB. Beschrijf ook waarom `terminate_current_task` afgesloten moet worden met een oneindige loop.
6. Schrijf nu een implementatie voor `void scheduler_task_remove(tid_t)`. Zoek door de array van taken naar de juiste taak, en zet diens `in_use` veld op `false`.

Taken roosteren

We hebben nu de basisingredienten voor een task scheduler. We kunnen taken opzoeken, verwijderen en toevoegen. Wat nog mist, is de code om ook echt taken uit te gaan voeren.

Wisselen tussen taken implementeren we aan de hand van een timer. Het doel is om na een tijdsquantum over te schakelen naar de volgende, wachtende taak. Het `TIME_QUANTUM` is bovenaan `kernel/scheduler.c` gedefinieerd. Om te weten wanneer we over moeten schakelen naar de volgende taak, gebruiken we de system timers van de RASPBERRY PI. Er zijn er 4 (genummerd 0, 1, 2, 3). Voor onze scheduler gebruiken we 1 De communicatie met de system timer gaat door middel van *hardware interrupts*. Om de scheduler te starten doen we het volgende (zie `void scheduler_start()`):

- We starten met luisteren naar interrupts van system timer 1

- We stellen de timer voor de eerste keer in met het `TIME_QUANTUM`.
- We gaan in een oneindige loop.

Wanneer de timer af gaat ontvangt FramboOS een hardware interrupt. Op het moment dat dit het geval is, willen we een task switch uitvoeren. Maar hoe doen we dat?

De interrupt vector, zoals beschreven in Silberschatz, wordt opgezet in `kernel/exceptions.s`. Hier is ook bepaald welke procedure wordt aangeroepen (lees: naar welk label we moeten springen) om de interrupts af te handelen. De afhandelingsprocedures staan in `kernel/handlers.c`. Er zijn verschillende soorten interrupts, en elke procedure in dit bestand verwerkt zijn eigen soort. In dit geval hebben we de *Interrupt Request Handler* of `irq_handler` nodig. Momenteel is deze geimplemeneerd met een kernel panic.

7. Bekijk de assembly code in het codeblok voor `irq_handler_asm_wrapper` in `kerel/exception.s` en beantwoord de volgende vragen: a. Welke instructies zorgen er voor dat de inhoud van alle registers op de stack wordt gezet? b. Welke instructie springt naar de procedure `void irq_handler(saved_cpu_state_t *)` uit `kernel/handlers.c`? c. Welke instructies zorgen er voor dat de opgeslagen registers weer van de stack teruggeplaatst in de CPU?

   Tip. Gebruik het internet om de betekenis van de assembly instructies te achterhalen mocht je die niet kennen. Het gaat er om dat je een *globaal* idee hebt van wat de instructies doen.
8. Breidt de IRQ handler in `kernel/handlers.c` uit met het volgende:
   - Controleer of `INTERRUPT_SYSTEM_TIMER_1` op het moment van aanroepen lopend is. Maak hierbij gebruik van `bool irq_is_pending(interrupt_id_t)` uit `kernel/hardware/interrupt.h`.
   - Wanneer dat het geval is roep je `void scheduler_task_switch(saved_cpu_state_t*)` aan.
   - **Belangrijk!** Vergeet vervolgens niet de lopende interrupt te wissen met `void timer_clear_irq_pending()`! Doe dit voordat je de interrupt daadwerkelijk afhandelt.

Nu kunnen we echt de task switcher implementeren.

9. Implementeer `void scheduler_task_switch(saved_cpu_state_t*)` in `kernel/schedular.c`.
   - Zoek de huidige actieve taak op en de volgende te actieveren taak. Gebruik hiervoor je gebouwde hulpfuncties.
   - Kopieër met `memcpy` één-op-één de CPU toestand van vóór de interrupt naar het TCB van de *oude* taak, zodat deze later weer door kan zoals die geëindigd was.
   - Kopieër met `memcp` één-op-één de CPU toestand uit het TCB van de *nieuwe* taak naar de locatie van de laatste CPU toestand vóór de interrupt.
   - Zet timer 1 weer op het `TIME_QUANTUM` zodat we na deze tijd weer een nieuwe interrupt krijgen.
   - **Let op:**
     - De signatuur van `mempcy` is `void memcpy(void *dest, void *source, size_t n)`. Het *eerste* argument is dus de destination, het *tweede* de source.
     - Het derde argument van `memcpy` is de grootte van het stuk geheugen dat je wilt kopieëren. In dit geval kun je aan de grootte komen met de expressie `sizeof(saved_cpu_state_t)`.
     - Vergeet absoluut niet timer nummer 1 weer op het tijdsquantum te zetten, anders wissel je nooit meer van taak!

Taken testen

Hopelijk is alles goed gegaan. We gaan je code nu testen door twee taken concurrent uit te laten voeren. In `user/tasks/rect1.c` en `user/tasks/rect2.c` vind je code voor het tekenen van twee rechthoeken op je scherm, eentje in de linkerbovenhoek en eentje in de rechterbovenhoek. Ze knipperen, de rechter twee keer zo snel als de linker.

10. Zorg er voor dat je `void kernel_init()` de scheduler initialiseert, dus dat die `void scheduler_init()` aanroept. Zorg er vervolgens voor dat je `void kernel_main()` er als volgt uit ziet:

    ```
    kernel_init();

    void task_rect1();
    void task_rect2();

    uint8_t task_rect1_stack[4096] __attribute__((aligned(8)));
    scheduler_task_add("rectangle top left", task_rect1, task_rect1_stack + 4096);

    uint8_t task_rect2_stack[4096] __attribute__((aligned(8)));
    scheduler_task_add("rectangle top left", task_rect2, task_rect2_stack + 4096);

    uart_log_begin("Yielding to task scheduler");
    scheduler_start();
    ```

    Draai je kernel! Hopelijk werkt het, zo niet, tijd om de opdracht nog eens van boven naar beneden door te nemen om te kijken waar het mis gaat... Maak goed gebruik van de log macros in `kernel/hardware/uart.h` zodat je weet hoe ver je kernel komt.

In bovenstaande code alloceren we twee keer een stuk geheugen dat als stack dient voor beide taken. Iedere taak heeft immers z'n eigen stuk geheugen nodig om daar in te draaien. In dit geval krijgen ze allebij 4KiB aan ruimte. Laatste vraag om je begrip te testen.

11. Stel dat `task_rect1` over deze 4KiB aan geheugen heen gaat. Wat gebeurt er?

Inleveren

Lever een zip *van je hele project* in via yOUlearn. Probeer dit te doen vóór de volgende bijeenkomst, dus voor dinsdag 20 december. Mocht dit niet lukken, neem dan contact op met de docent. Geef hieronder aan hoeveel uren je met deze opdracht bezig bent geweest.

Tijd gestoken in deze opdracht: ... uur.

Het moeilijkste aan deze opdracht vond ik:

- ...

Het leukste aan deze opdracht vond ik:

- ...