Practicum 3

De scheduler – deel 1

Dit practicum demonstreert hoe onderbrekingen kunnen worden gebruikt om een eenvoudige scheduler te implementeren. De scheduler is het onderdeel van het besturingssysteem dat ervoor zorgt dat de beschikbare processortijd wordt verdeeld onder alle processen/taken die momenteel actief zijn. Dit gebeurt door taken te onderbreken (via een onderbrekingsroutine die vanwege een extern signaal wordt aangeroepen, zoals een kloksignaal), hun uitvoeringscontext op te slaan (hun registerwaarden e.d.), de uitvoeringscontext van een andere taak in te stellen en vervolgens weer terug te keren uit de onderbrekingsroutine.

Door zeer snel te wisselen tussen verschillende taken (typisch om de 10ms op moderne systemen), krijgt de gebruiker de indruk dat alle taken simultaan lopen.

3.1 Indienen

Practicum 4 zal verderbouwen op de functionaliteit die je moet implementeren in Opgave 5. Om die reden vragen we dat je het bestand dat je de code van die opgave indient voor (**2 april 2014 om 22:00:00**) via `https://indianio.ugent.be`. Let op! We verwachten per groep één enkel zipbestand met de volgende inhoud:

- `practicum3_vraag5.asm`

De verbetering zal gebeuren voor de aanvang van het 4e practicum zodat jullie de feedback kunnen gebruiken bij het oplossen daarvan. Merk op: je moet bij deze voorbereiding enkel code indienen, geen verslag.

3.2 Inleiding

Gegeven het programma `scheduler.asm`. Dit programma zal worden geassembleerd en gelinkt tot een programma dat op de bootsector van een floppy of cdrom kan geschreven worden. Bij het opstarten van de (gesimuleerde) computer gaat de controle over naar dit programma en hebben we complete toegang tot alle onderdelen van de processor, omdat we ons op dat ogenblik in kernelmode bevinden.

De bootsector bestaat uit de eerste sector van de floppy of cdrom. Deze sector zal de volgende sectoren inlezen en het bijhorende programma uitvoeren. Om als bootsector herkend te worden, moet de eerste sector eindigen met het bitpatroon `55AA`. De eerste sector van de floppy of cdrom wordt door de BIOS automatisch geladen op adres `7c00h`. We laten de stapel vanaf adres `7c00h` groeien naar adres `0` toe – we volgen hierin de normale conventie waarbij de stapel op de x86 groeit naar de lagere adressen.

Nadat alle sectoren ingeladen zijn, wordt omgeschakeld naar protected mode en wordt er begonnen met de uitvoering van 32-bit code.

Daarna wordt het hoofdprogramma uitgevoerd, dat bestaat uit het herhaald tekenen van een spiraal.

Om het practicum te kunnen uitvoeren beschik je over de volgende bestanden:

- `scheduler.asm` : het hierboven beschreven programma
- `k.sh` : script dat keyboard.asm assembleert en omzet in MyBoot.bin
- `b.sh` : script dat de bochs emulator opstart met het bestand MyBoot.bin

Voor dit practicum zullen we ons niet belasten met het herhaaldelijk herstarten van de computer vanaf een floppy of cdrom. We maken gebruiken van een een emulator (bochs) die een Intel Pentium emuleert. In het configuratiebestand van bochs (`bochs.conf`) staat dat de inhoud van een floppy opgeslagen ligt in `MyBoot.bin`. Op die manier zal de emulator booten van `MyBoot.bin`. Let op! Het gebeurt frequent dat men nog de oude `MyBoot.bin` gebruikt, omdat er iets faalt tijdens het compileren van `keyboard.asm`.

Je editeert `scheduler.asm` met jouw favoriete editor (b.v. met vim, emacs :), gedit, nano,...). Vervolgens bewaar je het bestand en voer je in de terminal in de directory van dit practicum het commando `./k.sh` uit om het programma te assembleren, en nadien `./b.sh` om het in bochs uit te voeren.[1]

Het bestand `MyBoot.bin` kun je indien gewenst disassembleren met het programma `ndisasm`. Je moet dit wel in twee keer doen: met de vlag `-b 16` voor de eerste sector en met de vlag `-b 32` voor de overige sectoren.

3.3 Overzicht

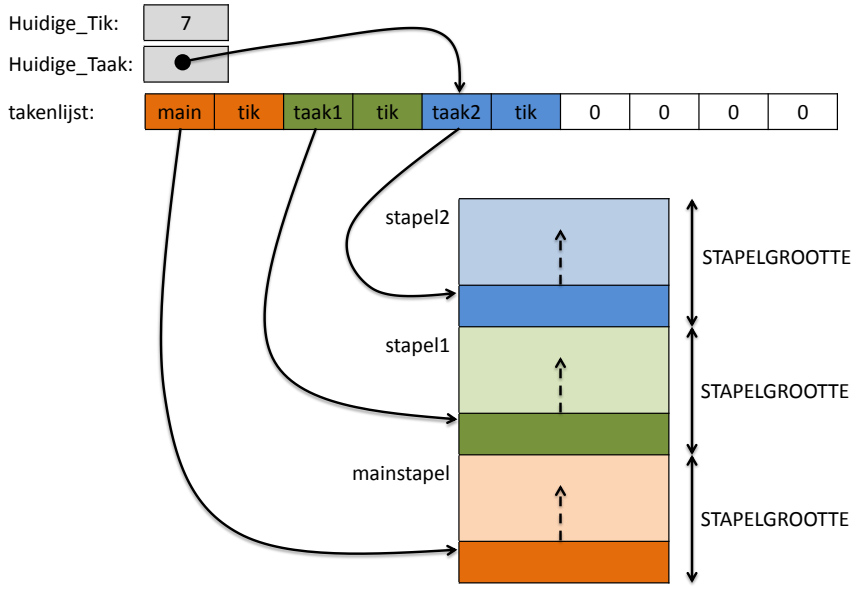
We gaan het programma in `scheduler.asm` te wijzigen zodat er in plaats van één spiraal, simultaan 2 spiralen getekend worden. We zullen dit niet doen door de routine spiraal aan te passen zodat ze de gevraagde twee spiralen tegelijk tekent, maar wel door de routine spiraal tweemaal op te starten (met verschillende parameters), en heel snel tussen die twee routines te wisselen. Om dit te kunnen doen, zullen we ook een onderbrekingsroutine moeten installeren.

De onderstaande stukjes code kunnen gebruikt worden om twee spiralen op te starten (een linkerspiraal en een rechterspiraal). Elk van deze twee taken zal over een eigen stapel moeten beschikken (zie verder).

```
Taak1: ; tekent een rechterspiraal
  mov eax,40
  mov ebx,79
  mov ecx,0
  mov edx,20
  jmp spiraal

Taak2: ; tekent een linkerspiraal
  mov eax,0
  mov ebx,39
  mov ecx,0
  mov edx,20
  jmp spiraal
```

[1] Nostalgische mensen kunnen hier ook echt een bootfloppy van te maken. Dat kun je doen door het commando `sudo dd if=MyBoot.bin of=/dev/fd0` uit te voeren. Je kunt de computer vervolgens van die floppy booten. Je kan ook het bestand MyBoot.bin op een cdrom branden als *Bootable cd* en de computer vervolgens van cdrom booten. Je kunt je dan vergewissen van het feit dat deze bootfloppy of bootcdrom ook in het echt werkt.



Figuur 3.1: Voorstelling van de takenlijst en de stapels die ermee geassocieerd worden.

Het afwisselen tussen de twee spiralen gebeurt als volgt. Op geregelde tijdstippen genereert de hardware van de computer een *timeronderbreking*, die door het besturingssysteem wordt onderschept. Telkens wanneer er een dergelijke onderbreking optreedt, zullen we alle registers van de taak van de (eerste) spiraal die op dat ogenblik getekend wordt, bewaren op de stapel van die taak en de finale versie van de stapelwijzer bewaren in de takenlijst. Dan veranderen we de stapelwijzer zodat hij wijst naar de stapel van de taak behorende bij de andere (tweede) spiraal. Op die stapel werden eerder de registers van die taak bewaard, zodat we nu alle registers kunnen herstellen, waarop de andere (tweede) taak uitvoert en de andere (tweede) spiraal verder getekend wordt. Als er dan opnieuw een timeronderbreking optreedt, vindt een analoge operatie plaats, waarbij nu terug wordt gegaan naar de eerste taak.

We lichten kort de datastructuren toe die door de scheduler worden gebruikt. In wezen beschikt de scheduler over een lijst van taken, waarin de structuur wordt gebruikt die ook kort in de slides van de les i.v.m. bordoefeningen assembler werd toegelicht, zie ook Figuur 3.1:

```
struct taak_entry {
  void* stapelwijzer;
  int tik;
}

struct taak_entry takenlijst[MAX_TAKEN];
```

Deze lijst zal steeds op een circulaire manier worden doorlopen. Bij alle taken, behalve de huidige taak, zal de toestand van de processor (de registers) op de stapel staan. Bij het aanmaken van een nieuwe taak moeten we er dus voor zorgen dat de initiële stapel de correcte beginwaarden van de registers bevat (de meeste hiervan zijn 0, behalve het vlaggenregister, de stapelwijzer en het codesegment). Daarnaast wordt voor elke taak bijgehouden op welke kloktik ze mag geactiveerd worden.

De onderbrekingsroutine die de taakwisseling uitvoert ziet eruit als volgt:

```
schedulerhandler:
```

```
        pushad
        inc     dword [Huidige_Tick]
        mov     al, 0x20
        out     0x20, al
        sti
        mov     ebx, [Huidige_Taak]
        mov     dword [ebx],esp
        mov     dword [ebx + 4], 0
        mov     ecx, [Huidige_Tick]
        call    animatiestap
        cli
        mov     esp, 0
.taakzoeklus:
        add     ebx, 8
        cmp     ebx, takenlijst + (MAX_TAKEN * 8)
        jl      .nog_niet_aan_het_einde
        lea     ebx, [takenlijst]
.nog_niet_aan_het_einde:
        cmp     dword [ebx],0
        je      .taakzoeklus
        cmp     dword [ebx+4], ecx
        jg      .taakzoeklus
        mov     [Huidige_Taak], ebx
        mov     esp, [ebx]
        popad
        iret
```

De eerste taak van deze routine is het bewaren van de registers voor algemeen gebruik op de stapel van de onderbroken taak (pushad). Vervolgens wordt de waarde op het adres Huidige_Tick verhoogd met 1 en laat de onderbrekingsregelaar weten dat de onderbreking ontvangen werd (**mov** + **out**). Daarna wordt de top van de stapel van de onderbroken taak in de takenlijst geschreven op de plaats aangewezen door Huidige_Taak. We zorgen ervoor dat de taakzoeklus niet onderbroken kan worden door gebruik van een cli instructie in de lus. Indien je zelf over de takenlijst itereert moet je zelf ook steeds zeker zijn dat de onderbrekingen uit staan.

Vervolgens zoekt de scheduler naar een volgende taak die in aanmerking komt om uitgevoerd te worden. Zodra het einde van de lijst wordt bereikt, wordt de zoektocht verdergezet aan het begin van de lijst. We gebruiken de datastructuur dus als een circulaire lijst. Taken met een kloktik groter dan Huidige_Tick komen nog niet in aanmerking om uitgevoerd te worden en ook in dit geval wordt er verder gezocht.

Eenmaal de scheduler een geschikte taak gevonden heeft, wordt de stapelwijzer van de taak hersteld, worden de registers hersteld (**popad**) en wordt de taak verdergezet op de plaats waar ze oorspronkelijk onderbroken werd (met **iret**). In dit schema veronderstellen we wel aan dat er minstens 1 taak zal gevonden worden. Zoniet zal de scheduler blijven zoeken.

Het enige verschil met een gewone onderbrekingsroutine is dus dat in dit geval niet de onderbroken taak, maar een andere taak verdergezet wordt.

Je zal merken dat er een kleine animatie wordt getekend door de schedulerhandler. Indien je wil, kan je zelf ook wat extra code toevoegen die de waarde van Huidige_Tick uitprint. Dit kan je bijvoorbeeld de volgende code toegevoen in de handler, na de sti instructie:

```
push 0
push 0
push dword [Huidige_Tick]
call printhex
```

```
add esp, 12
```

3.4 Opgaven

1. De instructies **pushad** en **popad** worden vaak gebruikt. Wat doen deze instructies? Teken de stapel na de uitvoering van een **pushad** instructie (je mag ervan uitgaan de de stapel initieel leeg is).

2. Verklaar waarom er in de schedulerhandler een **cli** instructie zonder corresponderende **sti** instructie voorkomt. Is dit een fout? **Indien wel**: waar dient de **sti** precies geplaatst te worden in de code? **Indien niet**: leg uit waarom niet.

3. Installeer de schedulerhandler op de timeronderbreking. Gebruik hiervoor de routine install_handler. Ga na op welke manier je het nummer van de vector en het adres van de handler moet meegeven. Op welk vectornummer moet de timerhandler geïnstalleerd worden? Merk op dat op dit punt de timeronderbreking nog niet ingeschakeld is, en je dus pas na de volgende opgave kan controleren of alles werkt.

4. Pas het onderbrekingsmasker in poort 21h aan zodat de timeronderbreking aangeschakeld wordt. Welke bit in het masker moet hiervoor aangepast worden? Het programma zal nu een aantal keer per seconde worden onderbroken en de taken in de takenlijst opeenvolgend laten uitvoeren. Zorg ervoor dat het programma blijft lopen zoals voorheen. In principe mag je de talrijke onderbrekingen niet opmerken. Je kan zien of je timeronderbreking wel is geïnstalleerd door het feit dat wanneer dit succesvol is gebeurd, er onderaan het scherm sowieso een extra klein animatietje zal worden getekend.

5. Schrijf de routine creëertaak die een nieuwe taak aan de takenlijst toevoegt en die drie argumenten heeft:

   (a) Het adres van de eerste instructie die moet uitgevoerd worden.

   (b) Het adres van de stapel die door deze taak gebruikt kan worden.

   (c) De tijd waarop gewacht moet worden alvorens de taak mag starten. Dit is dus de waarde die Huidige_Tick moet bereiken alvorens de taak mag gestart worden.

   Deze creëertaak wordt opgeroepen als creeertaak(adres, stapel, wachttijd), en moet:

   (a) De stapel van de nieuwe taak initialiseren zodat het lijkt alsof de taak onderbroken werd net voor de uitvoering van de eerste instructie in die taak. Op die manier is de taak klaar is om gestart te worden door de schedulerhandler.

   (b) Op een lege plaats in de takenlijst de resulterende stapelwijzer (dus na het goedzetten van de stapel!) en kloktik invullen. Die kloktik is de tick waarna de taak mag beginnen uitvoeren.

   (c) Ervoor zorgen dat de IF-bit op 1 staat in de data die de EFLAGS voorstellen.

   Hierbij willen we ook weten:

   (a) Waarom moet de IF-bit op 1 gezet worden?

   (b) Waarom moet je deze vlaggen op de nieuwe stapel bewaren?

Tip: Maak hierbij gebruik van de informatie die je in de voorbereidings-opgaven verzameld hebt.

Om het geheel uit te testen reduceer je de spiraal uit het hoofdprogramma tot de rechterhelft van het scherm (b.v. 40-79), en creëer je een linkerspiraal die de linkerhelft voor zijn rekening neemt (b.v. Taak1 uit de inleiding). Start deze taak op met de routine creeertaak en zorg ervoor dat deze taak pas later gestart wordt (b.v. na 2500 kloktikken; kies dit getal niet te klein, anders gebeurt dit opstarten zo snel na het booten dat je niet zal merken of de taak al dan niet correct wacht met starten). Gebruik steeds stapel1 voor taak1, stapel2 voor taak2, ...

Sla je bestand op als `practicum3_vraag5.asm`.

6. Zorg er nu voor dat het hoofdprogramma geen spiraal meer uitvoert, maar enkel nog de twee taken Taak1 en Taak2 uit de inleiding opstart. Laat het hoofdprogramma nadien wachten in een eindeloze lus (die we later zullen wegwerken). Start ook meteen de informatiegevende taak PrintInfoTaak, die informatie over de scheduling op het scherm zal printen.

   PrintInfoTaak is een taak die informatie afprint die handig kan zijn bij het debuggen van je code. Wat je hieruit kan afleiden is:

   (a) Of de ticks nog verhogen

   (b) Indien de ticks niet verhogen, kan het zijn dat er geen schedulingswitches meer gebeuren. Indien de spiralen niet meer draaien, maar het timestamp counter (TSC) veldje verhoogt, dan weet je dat deze infotaak wel actief is.

   (c) Bij elk taakslot zie je tot welke clocktick deze taak zal slapen, en of er wel een taak in zit (de 'A' staat hier voor *Actief*, de 'T' voor *geTermineerd*).

Succes!