Eksamen 2012  
3. semester  
I3MPS - Microprocessor systemer  
Kasper Nissen

**Spørgsmål 2**

**Memory Protection**

**Spørgsmål**

* Hvad er memory protection?
* Hvordan fungerer en MMU?
* Hvordan laver man context switching vha en MMU?
* Hvordan benytter Linux sig af en MMU?

**Litteratur**

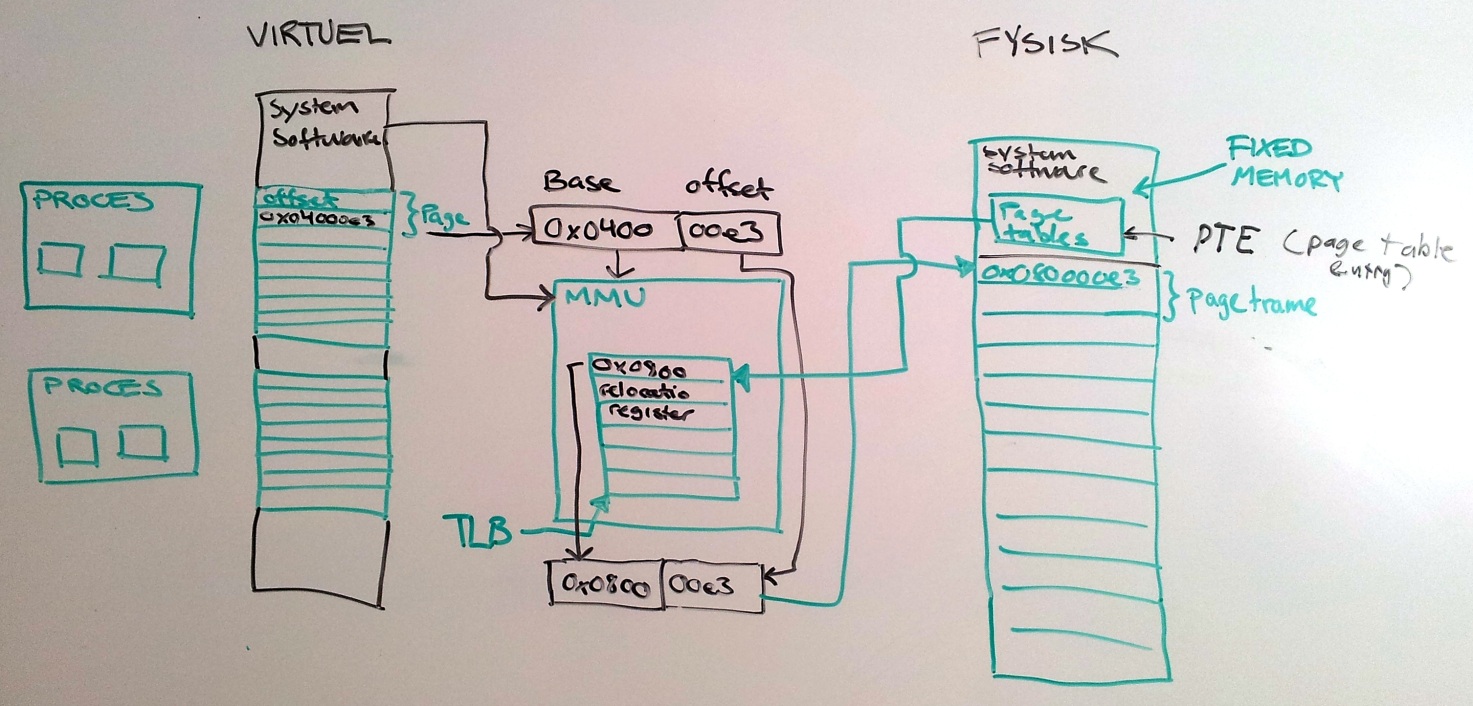
Text: ARM System Developer’s Guide, Sloss/Symes/Wright: 12-12.32 (Cache) + 14-14.5 (MMU) +<http://people.redhat.com/drepper/cpumemory.pdf> ch 6 -> side 50 linie 7 +<http://tldp.org/LDP/tlk/mm/memory.html> 3.1-3.6

**Exercise 10**

**Hvad er en MMU?**

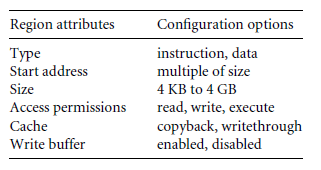
Memory Management Unit (MMU) er en hardware baseret memory management enhed. En af MMU’ens primære opgaver er at styre individuelle opgaver som uafhængige programmer kørende i hvert deres hukommelses område.

MMU’en gør programmeringen af applikationers opgaver simplere, da den sørger for at enable den virtuelle hukommelse – som er en hukommelse som er uafhængig af den fysiske hukommelse tilsluttet systemet. MMU’en fungere som en oversætter, idet den konvertere program- og dataadresser som er kombileret til at kører på virtuel hukommelse til at kører på fysisk hukommelse i main memory.



**Hvad er memory protection?**

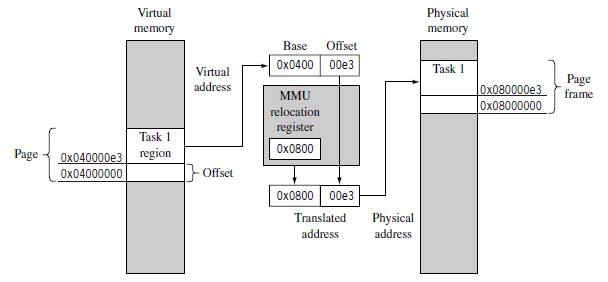
I nogle ARM processorer bruges i stedet for en MMU en MPU(memory protection unit), som benytter sig af det man kalder regions. En region er enten active eller dormant. Hvilket vil sige at en active region indeholder kode eller data som er i brug, hvorimod dormant regions er regioner som indeholder kode eller data der sandsynligvis snart skal i brug. En dormant region er beskyttet og er derfor utilgængelig få en kørende opgave(task). En MPU har dedikeret hardware som assigner attributter til de forskellige regioner. Disse attributter kan ses i figuren herunder:



Forskel på en MPU og MMU er tilføjelsen af hardware der understøtter virtuel hukommelse i en MMU. En anden forskel er at en MMU indeholder flere tilgængelige regioner, da registrer kan flyttes til main memory.

**Virtuel hukommelse**

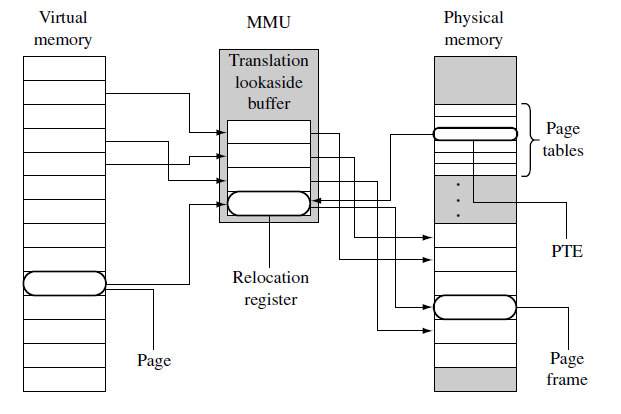
Hver proces har deres egen virtuelle hukommelse, som er den hukommelse som er placeret mellem CPU og MMU. Det vil sige at vi har mange forskellige virtuelle hukommelses områder, men kun et fysisk memory map. Som det ses på figuren nedenfor ses det at MMU’en relokere den virtuelle adresse til en fysisk adresse i main memory.



Som det ses ud fra figuren gør MMU’en den tager de øverste bits og ombytter dem med det er ligger i relokeringsregistreret, mens de nederest bits kaldes for offset og oversættes til en specifik adresse i den fysiske hukommelse. Området i den virtuelle hukommelse for en process kaldes også for en page, mens det tilsvarende fysiske område kaldes for en page frame.

**Translation Lookaside Buffer**

MMU’ens translation lookaside buffer er faktisk en fully associative cache, da den kan mappe en hvilken som helst virtuel adresse til en hvilken som helst fysisk adresse.



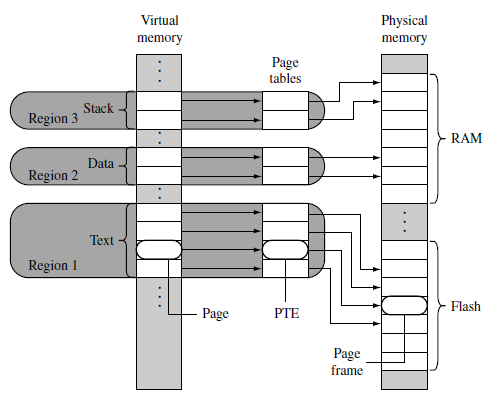
Udover relokations registrene bruger MMU’en også tabeller i den fysiske hukommelse, hvor data om det virtuelle hukommelsesområde gemmes. Disse kaldes for page tables. En række(PTE) i en pagetable repræsentere alt den information som er nødvendig som at oversætte en page i det virtuelle område til en page frame i det fysiske område.

PTE indeholder informationer om den fysiske baseadresse, rettigheder og cache/writebuffer konfiguration for pagen.

**Definiering af Regioner ved brug af Pages**

Modsat MPU’en hvor regioner i hukommelsen er hardware komponenter, bliver regioner i en MMU lavet af software ved gruppere blokke af virtuelle pages i hukommelsen.

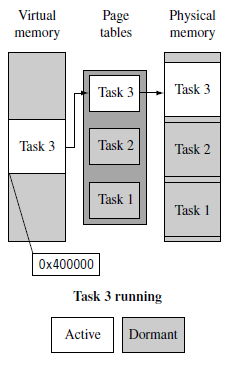
Regioner i MMU’en er defineret som grupper af page tables og er styret fuldstændig i software som sekventielle pages i virtuel hukommelse.



Figuren viser en task(opgave/process) som har 3 regioner; en til tekst, 1 til data og en til en opgavestak. Som det ses bliver hver region mapped til forskelligesteder i fysiske hukommelse. Tekst skal forståes som at det er her selve den eksekverbare kode ligger, som det ses bliver den mapped til flash hukommelsen.

**Multitasking og MMU’en (Context Switching)**

En måde at opbygge et multitasking system er ved at lave separate set af page tables, hvor hver mapper et unikt virtuel hukommelses område for opgaven. Når en opgave skal aktiveres, mapper MMU’en så det pågældende page table og det dertilhørende virtuelle hukommelsesområde in i MMU’en. De inaktive rækker i page tablen, repræsentere dormant opgaver(slumrende/sovende). Dette gør at alle opgaverne er klar i den fysiske hukommelse til (hvis) der sker et context switch.

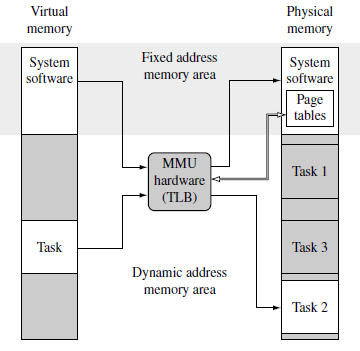


Skift mellem tasks kræver følgende trin / Krav før context switching:

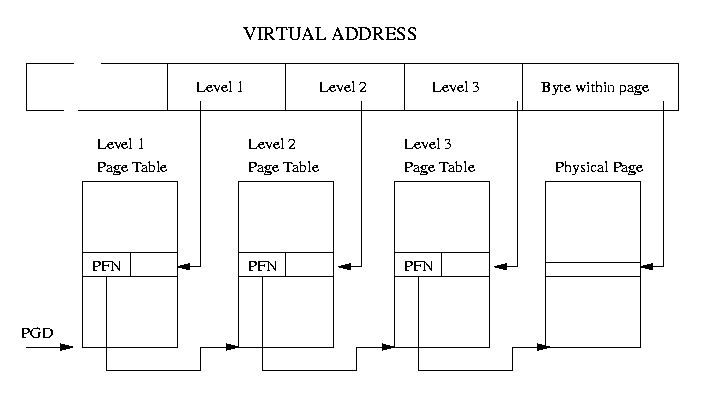
1. Gem den aktive opgave og placer opgaven i dormant tilstand
2. Ryd cachene.
3. Ryd TLB for at fjerne oversættelses informationerne
4. Konfigurer MMU’en til en ny page table, ved at oversætte virtuelle hukommelses execution område til den opvågne opgaves lokation i den fysiske hukommelse
5. Gendan den opvågnede opgave
6. Start eksekveringen af den gendannede opgave.

**Organisering af Memory i det virtuelle hukommelses system**

Typisk så ligger page tables i et område af main memory, hvor virtuel-til-fysisk adresse mapping er fixed. Det vil sige at data i en page table ikke ændre sig under normal operation. Det fixed område indeholder også system kernel og andre processer. Fordelen ved dette fixed mapping ses når der laves context switching. Ved at placere system software i fixed virtuelt hukommelses område eliminere nogle hukommelse organiserings opgaver, og vi får en pipeline effekt. Når et context switch opstår mellem 2 opgaver, foretager processoren i realitetn mange context switches(bl.a. mellem kernel mode og usermode). Ved at dele system softwaren i et fixed område af den virtuelle hukommelse som ses af alle user opgaverne, og vi behøver ikke koncentrer os om at skifte page table.

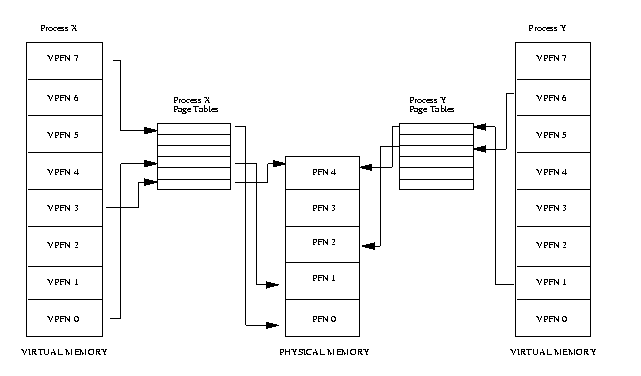


**Hvordan benytter Linux sig af en MMU?**

****

Linux bruger 3 levels af page tables. Hver table indeholder et page frame nummer på den næste page table. Linux nedbryder altså en virtuel adresse til nogle mindre dele. Hver del indeholder et offset.

For at oversætte en virtueladresse til en fysisk, tager processoren indholdet af hver del, konvertere dette til et offset ind i den fysiske page og læser page frame nummeret på det næste level, hvorefter dette gentages 3 gange indtil page frame nummeret på den fysiske adresse indeholdende den virtuelle adresse er fundet. Så bruges offsettet til at afgøre den endelige virtuelle adresse.

****

Linux bruger page tables til at omadressere en virtuel hukommelse til fysisk hukommelse. Processoren skal først have fundet det virtuelle page frame nummer og offset i den virtuelle page før der kan oversættes til fysiskhukommelse.

Demand paging

Linux bruger Demand paging, hviket betyder at man kun loader virtuelle sider ind i memory efterhånden som de gennemgåes. Når en kommando udføres, åbnes filen og dens indhold bliver mapped in i denne process virtuelle memory område. Dette gøres ved at modificere data strukturen som beskriver processens memory map – kaldes også for memory mapping.

Swapping

Linux bruger Least Recently Used(LRU) til at bestemme alderen af en page. Måden det foregår på er at alle pages har en alder, og hvergang denne sider har været tilgået, ændres alderen. Jo mere en side tilgåes jo yngre er den. De gamle sidder er gode kandidater til swapping. Swapping betyder at hvis der ikke er mere fysisk hukommelse til rådighed skal styresystemet gøre plads ved at slette nogle andre.

