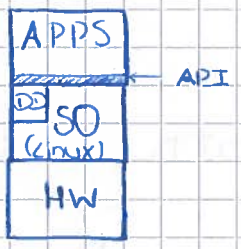
SO

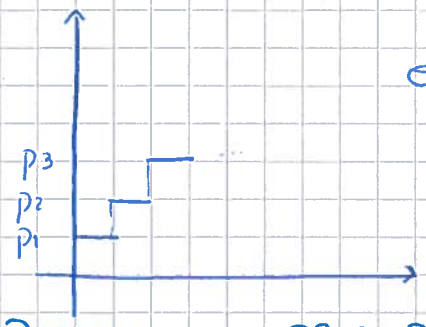
6-02-2017

Prof. Pedro Souto



8-02-2017

⇒ call (invocar uma função)

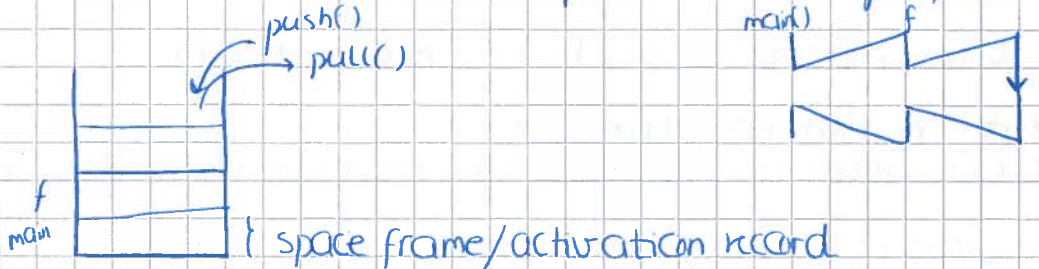


O processador "divide" os programas em execução ao longo do tempo.

Processos = programas em execução

Stack

→ normalmente é usada para invocar funções



↑ space frame/activation record

→ tipo LIFO

Dá a ilusão que tem toda a memória física disponível para ele

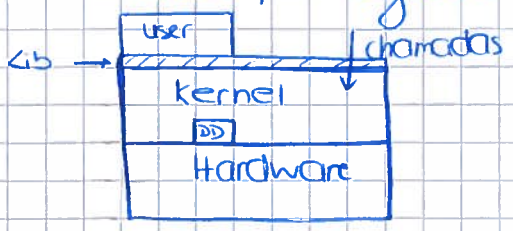
- swap file (windows)
- swap area (Linux && Mac)

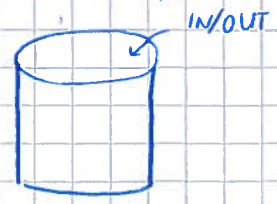
13-02-2017

Organização de um SO

→ Núcleo/Kernel

- modo privilegiado → pode executar todas as instruções





- reativo → responde às chamadas do sistema
  responde às interrupções

- parte do kernel realiza tarefas para o bom funcionamento do SO
  garantir memória suficiente

→ Chamadas do sistema

Implementação do Kernel dum SO

Implementação Monolítica

- não há processos ao nível do utilizador

Implementação baseada em Micro-Kernel

- tudo ao nível do utilizador
- cada processo funciona em espaço de endereçamento distinto
- isolamento entre componentes do SO
- + fácil → - código → - bugs
- MAS menos eficiente

Arranque de um SO

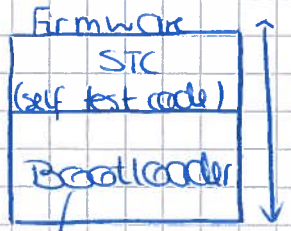
→ teste do hardware
realizado por código no firmware/memória não volátil na motherboard

→ boot(strapp)/loading
sistema está em disco tem que ser passado para a RAM
em sistemas embebidos + simples esta fase não existe (não há disco)

→ inicialização do SO
configurar dispositivos I/O

Boot(strapp) loading

Firmware → função para carregar o SO

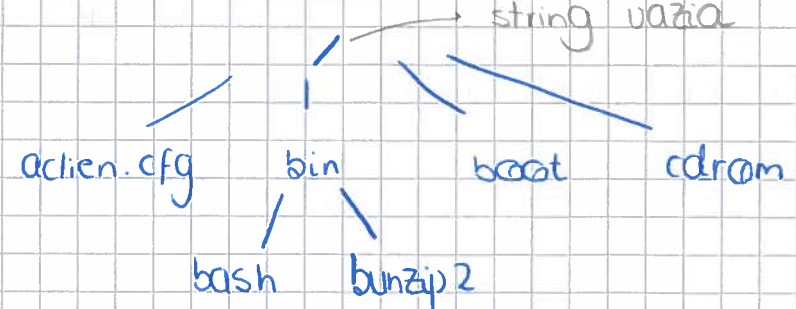


acede à memória não volátil (concretamente ao setor de arranque) e carrega para a RAM

bootloader carrega bootloader que carrega bootloader ?

Ficheiros

Processo — CPU
Espaço de Endereçamento — RAM
Ficheiro — Disco



/bin/bash → nome absoluto

File Descriptor

0 - STDIN — teclado
1 - STDOUT — ecrã
2 - STDERROR

} pré definidos

10 - 02 - 2017

3ª. Carolinas J. Luz

www.fe.up.pt/~jmcruz

Sistemas Operativos

- conceitos teóricos
- programação
  - API
  - núcleo
    - device driver → porta série

Avaliação

→ Exame escrito
- técnicas
- programação API
- device driver

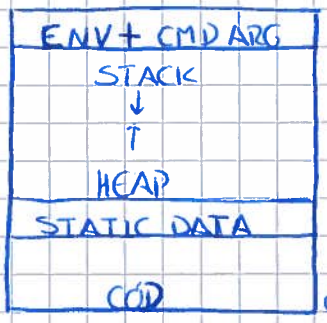
→ Mini-projecto

→ Prova Prática
- programação API

15-02-2017

Processos

- processador
- espaço de endereçamento



- ficheiros abertos (incluindo dispositivos de E/S)

STDIN
STDOUT
STDERR
} sempre abertos

Dados estáticos → exemplo "strings"

Criação de Processos

pid_t fork(void) /* clones the calling process */

→ O processo criado (filho):
- executa o mesmo programa que o programa pai
- inicia a sua execução na instrução que segue a fork

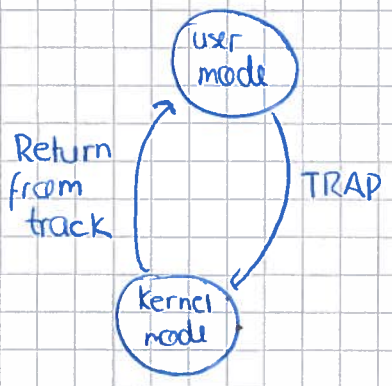
20-02-2017

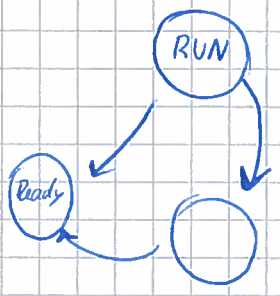
Limited Direct Execution

Privilege Levels

→ user mode

→ kernel mode





22-02-2017

LDE: (limited direct execution)
- proteção → chamadas ao sistema
- suspender processos em execução

Workload Assumptions

1- cada processo tem o mesmo tempo de execução
2- chegada ao mesmo tempo
3- processo executa até terminar
4- não há I/O
5- tempo de execução é conhecido

24-02-2017 (TP)

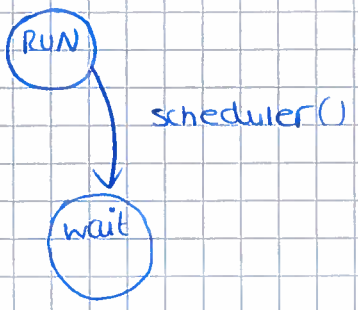
Acesso a ficheiros

fd → file descriptor

```
int fd = open(

nread = read(fd, &buffer[0], n)      // retorna nr caracteres que conseguiu ler
                                     // n bytes a ler

nwrite = write(fd, &str[0], n)

       = close(fd

char buffer[100];
char str[] = "olá!";
&buffer[0] ≡ buffer
```

1-03-2017

slide 18

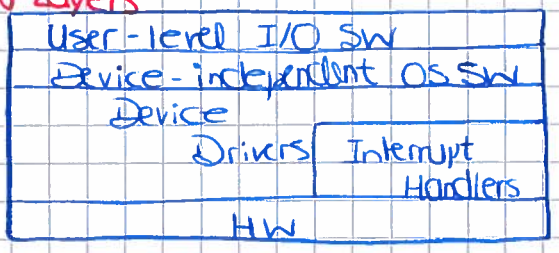


DMA → direct memory access

3-03-2017 (TP)

6-03-2017

I/O SW Layers



→ ex: printf

Character Device Programming Interface in Linux

```
struct file_operations {
        open()
        read()
        write()
        flush()
        release()
        ...
};
```

8-03-2017

Threads

→ abstract the execution of a sequence of instructions, i.e. a thread of execution

Simplifying, whereas a process abstracts the execution of a program, a thread abstracts the execution of a function.

Várias threads de um processo partilham tudo menos as respetivas stacks e o estado do processador.

13-03-2017

```
int pthread_create(pthread_t *id, const pthread_att_t attr,
                   void *(*start_fn)(void*), void *arg)
```

```
# include <pthread.h>
void *fun(void *arg){
    args_t *my_args = args;
    rt_t *rt = malloc (sizeof(rt_t));
    ...
    return rt;
}
```

gcc -pthread

assert (rc == 0) → aborta o programa se a condição for falsa

volatile → keyword de C
variável pode ser acessível em qualquer altura

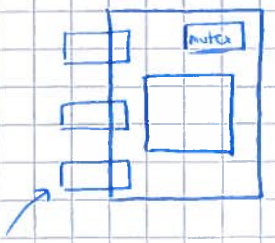
15-03-2017

```
lock (mutex);
Critical section
unlock (mutex);
```

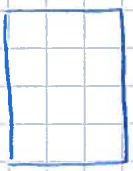
```
Pthread_mutex_lock (& lock);
for (i=0; i< max < i++)
        counter = counter + 1;
Pthread_mutex_unlock (& lock);
```

(não tem máxima concorrência)

Abstract data type ⟶ Monitor




```
f( ) {
    lock (& mutex);
```



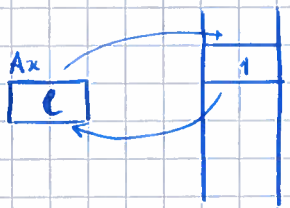
```
    unlock (& mutex);
}
```

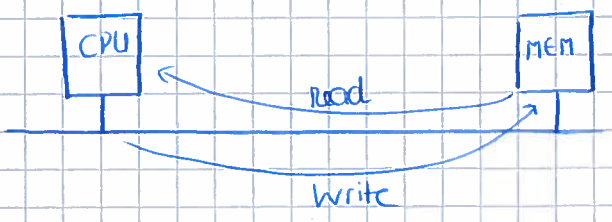
17-03-2017 (TP)

20-03-2017

XCHG → change (Intel 32 ISA)

swap de variável para memória e vice versa





20 queue_add() ← set park()
21 m→guard=0

unlock()
27
29
32

lost-wakeup

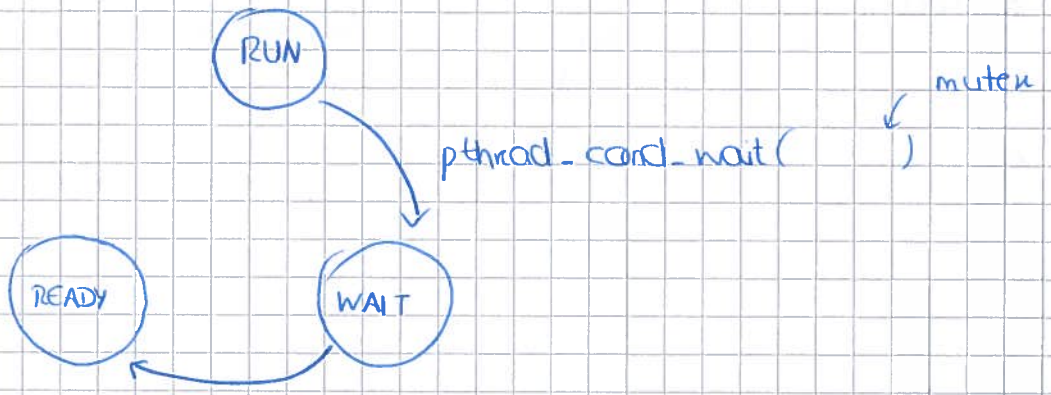
22 park()

22-03-2017

Condition Variables

volatile → variável pode ser acedida a qq momento

→ fila de threads que ficam à espera de uma condição



Slide 11 - lost wake-up

Thr 1

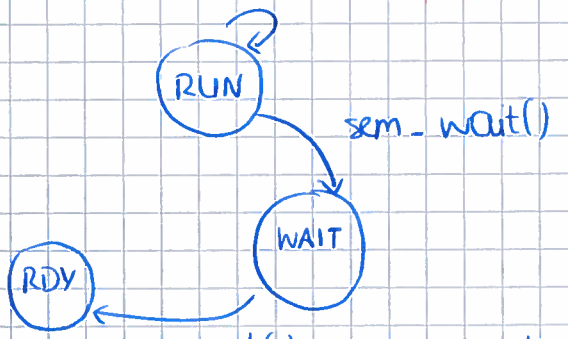
2. done = 1.
3. cond_signal()

Thr 2

7. if (done == 0)
8. cond_wait

Slide 16

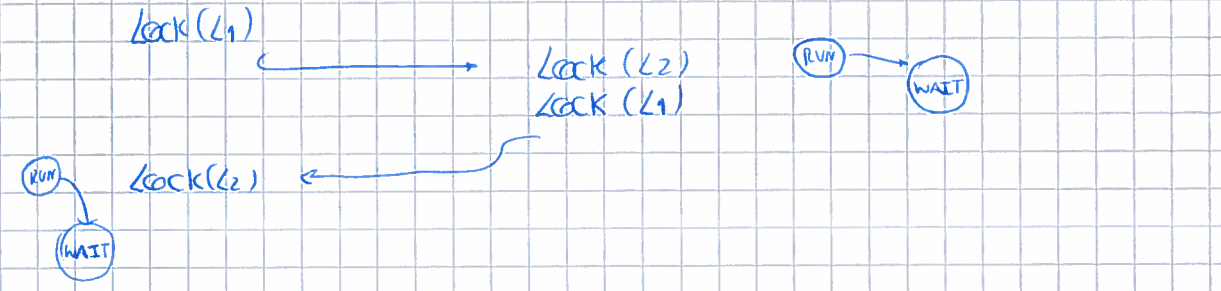
27-03-2017 Semaphors



→ quando outro thread executa isso

29-03-2017

Slide 13



//

V1.addAll(V2)

lock(V1);
lock(V2);

V2.addAll(V1)

lock(V2);
lock(V1);

NOTA: não tem race condition, mas pode ter deadlock

Prevention - estrutural

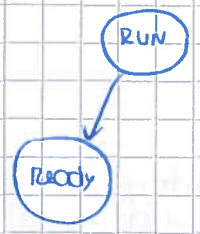
Avoidance - em run-time

3-04-2017

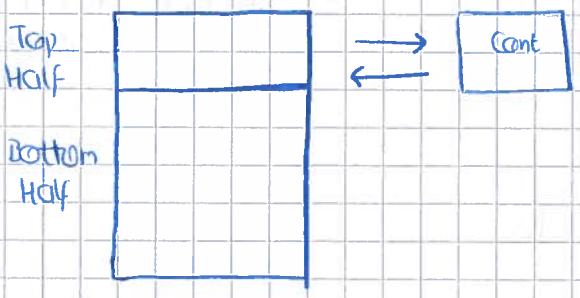
Kernel tem que implementar os mecanismos de sincronização

Kernel Synchronization

Preempção → processador retirado ao processo
↑ pelo kernel



yield() → processo passa de RUN para READY voluntariamente



Device-driver → no bottom half

request_irq semelhanças com pthread_create

leitura pode ser partilhada
escrita tem de ser em exclusão mútua } solução: read-write lock

void sema_init (struct semaphore * sem, int val)
↓
semáforo do kernel
e não do posix

não faz sentido que porta série seja partilhada pelos processos

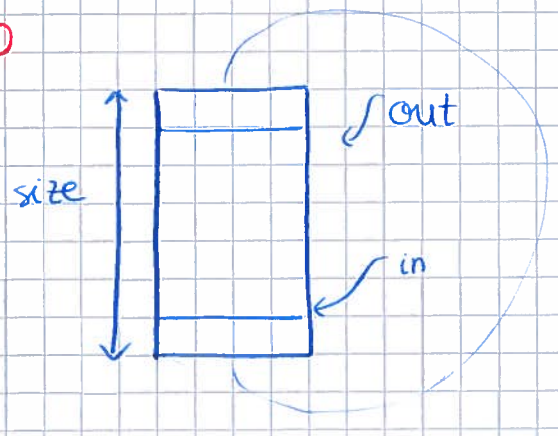
lock à chamada do sistema open, mas como open é atómico basta flag ocupado/livre

man 2 open

5-04-2017

DD down, filegas, up > semáforo inicializado a 1

KFIFO



kfifo → interface entre threads do utilizador e IH

→ usar spin-lock-irqsave

→ IH não precisa de spin_lock_irqsave

waitqueues ≃ condition variables

usar wait_event_interruptible em vez de wait_event

ex: thread em wait e queremos usar Ctrl+C, se usarmos wait_event o comando não terminará o programa

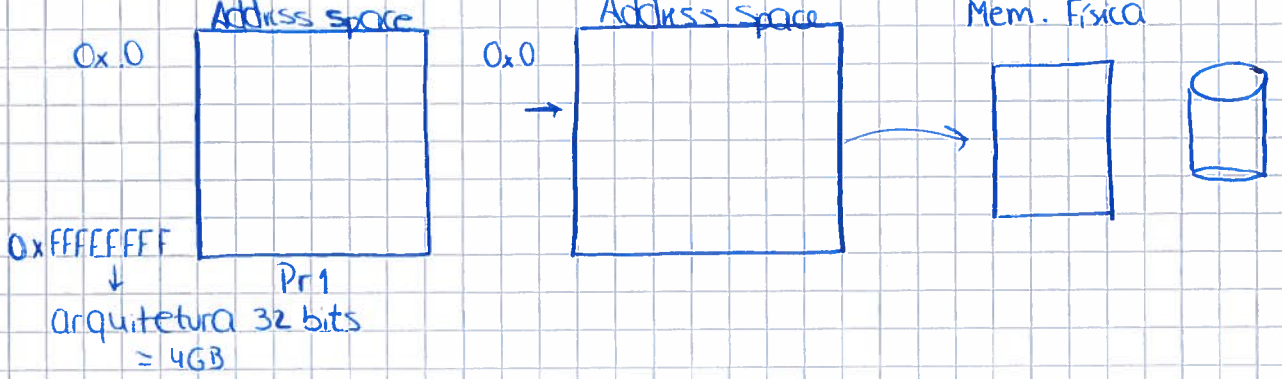
wake_up → útil : se kfifo vazia não há nada para ler, então thread vai dormir até fila ter algum elemento preenchido

usar API de alto nível

cdev → associado a qualquer character device

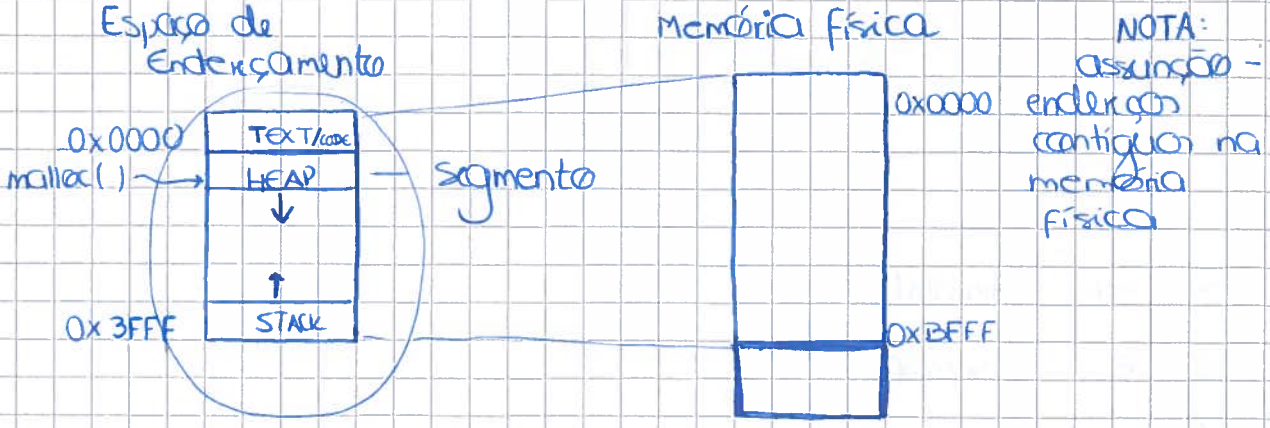
while (out ≠ in)

VM Introduction

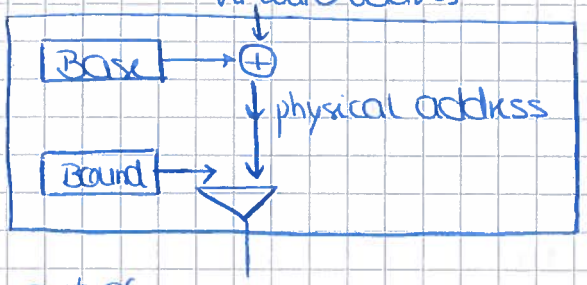


Stack → implementação de funções
Heap → variáveis alocáveis dinamicamente
Program code → código e variáveis estáticas

19-04-2017



MMU → memory management unit



Se estiver out of bounds → segmentation Fault (Exceção)

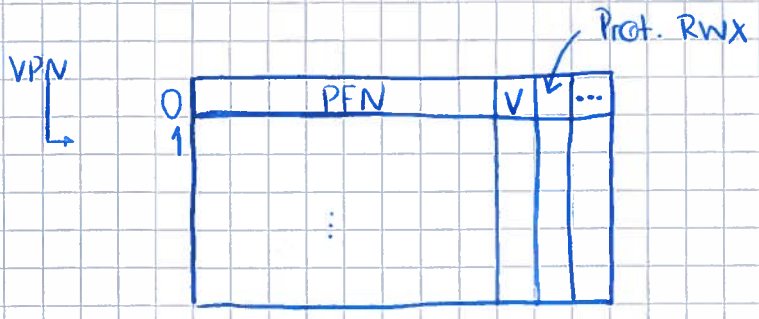
Se qualquer processo pudesse alterar base e bound, então era possível aceder-se a memória de outro processo. Necessário haver modos distintos: privilegiado e não privilegiado

24-04-2016

nr bits page offset = nr bits frame offset, pois o tamanho da page é igual ao tamanho da frame e mapeia-se 1 byte da page no 1 byte da frame

Virtual page number to page frame number mapping

array → the page table

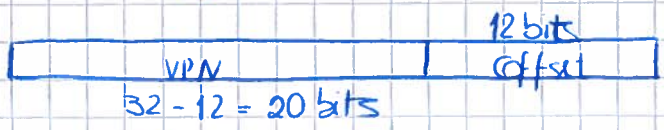


32-bit length(VA)

4 KiB sizeof(PAGE)

4 B sizeof(PTE) page table entry

$\log_2 4\,KiB = \log_2 2^{12} = 12$ → offset



nr páginas = $2^{20} \simeq 1\,Mi > 1M$

size of table = $2^{22} = 4\,MiB$ → 0.1% do AS

necessário uma por processo

Conclusão: guardar as page tables em memória

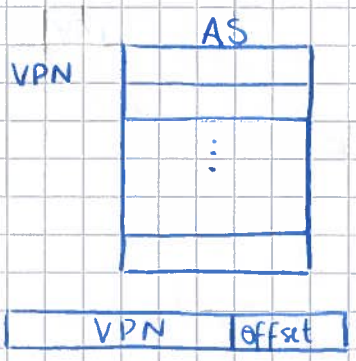
26-04-2016

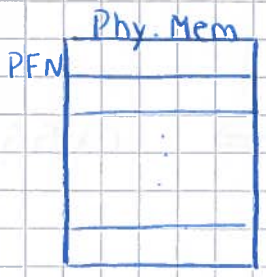
Slide - How to reduce PTs size?

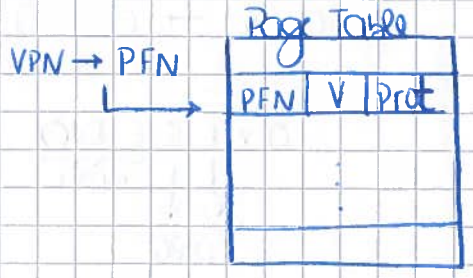
Usar páginas maiores → reduz tamanho da page table mas aumenta fragmentação interna

3-05-2017

VM - Paging







issues
- + 1 memory access
- TLB → cache
- page table too big
  - 4 MiB
    - 32 bit
    - 4 byte PTE
    - 4096 B/page

AS
16 páginas

PT
4 PTE/página

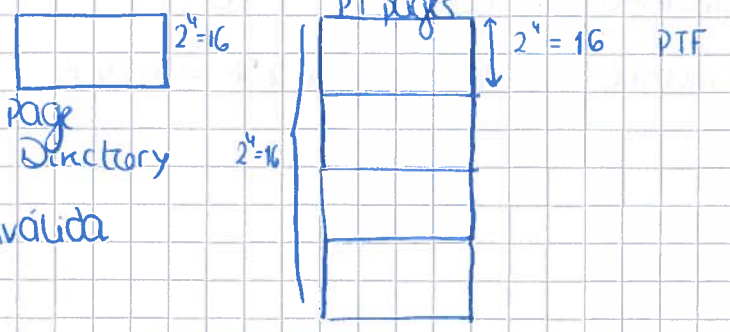
Exemplo

20 bits

12 bits — 4096 B/pag

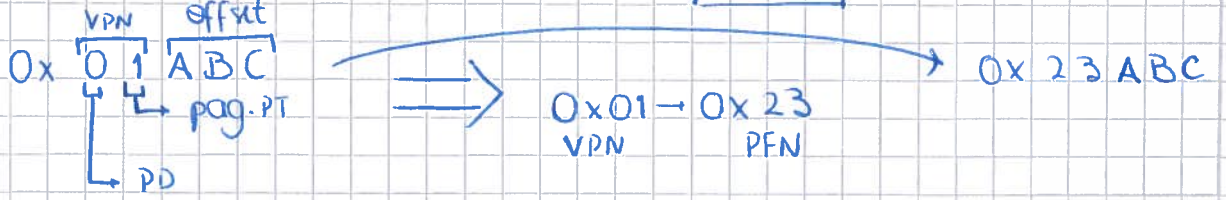
8 — VPN

$2^8 = 256$



slide 9 - falta 1 linha inválida

A.



B. 0x04000 ⇒ 0x80000

PD PG offset

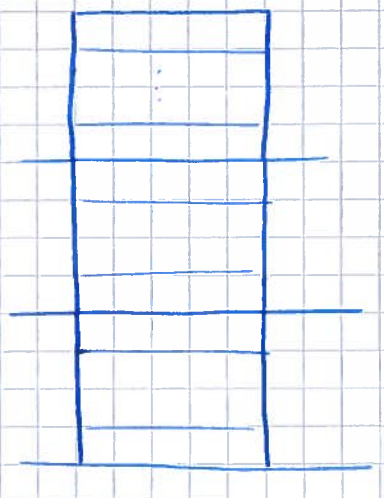
VPN → PFN

0x04 → 0x80

C. 0xFEED0 ⇒ 0x55ED0

PD PG offset

0xFE → 0x55



AS - 30 bits

Page size → 2^{12} = 4096 B

PTE size → 4 B

$\frac{4096\ B}{4\ B} = 1024$

$\log_2 1024 = 10$

8	10 (2nd Level)	12 Offset

30 bits

$2^8 = 256$ PTE

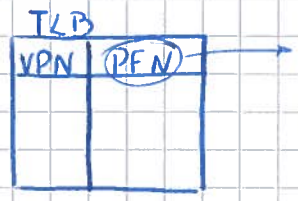
Tamanho de cada PTE = 4 B

Tamanho de PD = 4 B x 256 = 1024 B

15-05-2017

conteúdo da heap e da stack → guardadas na swap area
- file system
- partição do disco

reservar previamente o espaço, mas + rápida



17-05-2017

Input / Output Disks
↳ HDD

NOTA: não se dá RAID

22-05-2017

Metadata dos extents → (nr 1 bloco, tamanho do extent) x 10

slide 11 → each leaf node

24-05-2017