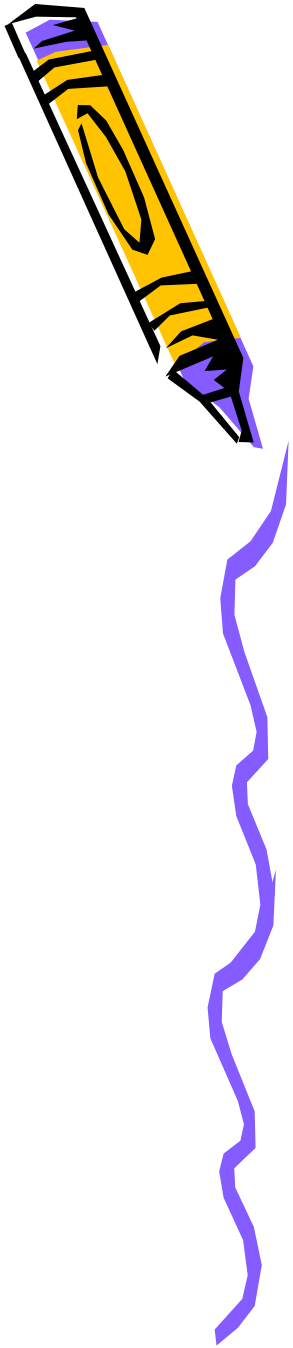
Alokasi Memori Kernel

Heri Kurniawan

OS-Gasal 2009/2010

Tujuan Pembelajaran



- Memahami buddy system
- Memahami Slab Allocator
- Memahami topik tambahan :
  - Prepaging
  - Ukuran page
  - TLBReach
  - Struktur Program

Alokasi Kernel Memori

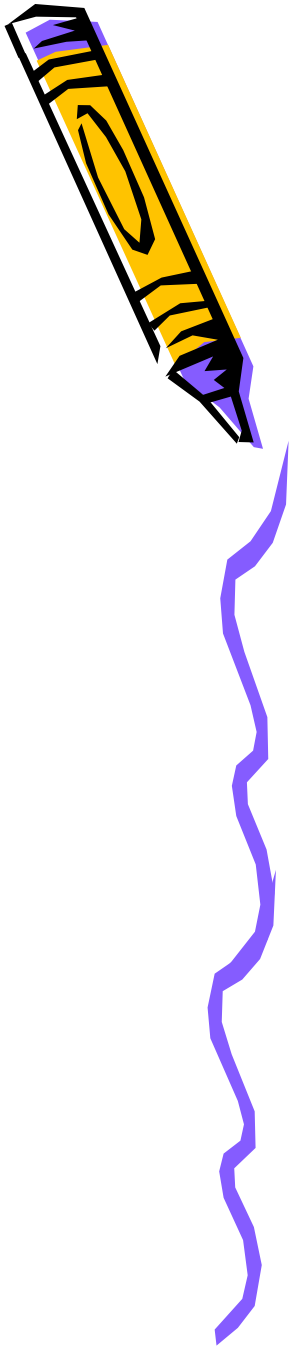
- Alokasi memory untuk proses kernel berbeda dengan proses user
  - Kebutuhan memory kernel bervariasi
  - Page proses user dapat ditempatkan secara acak dalam memori namun beberapa perangkat keras kadang harus mengakses memori fisik langsung (tidak menggunakan virtual memori).

  Manajemen memori untuk kernel
  - *Buddy system*
  - *Slab allocation*

Buddy System

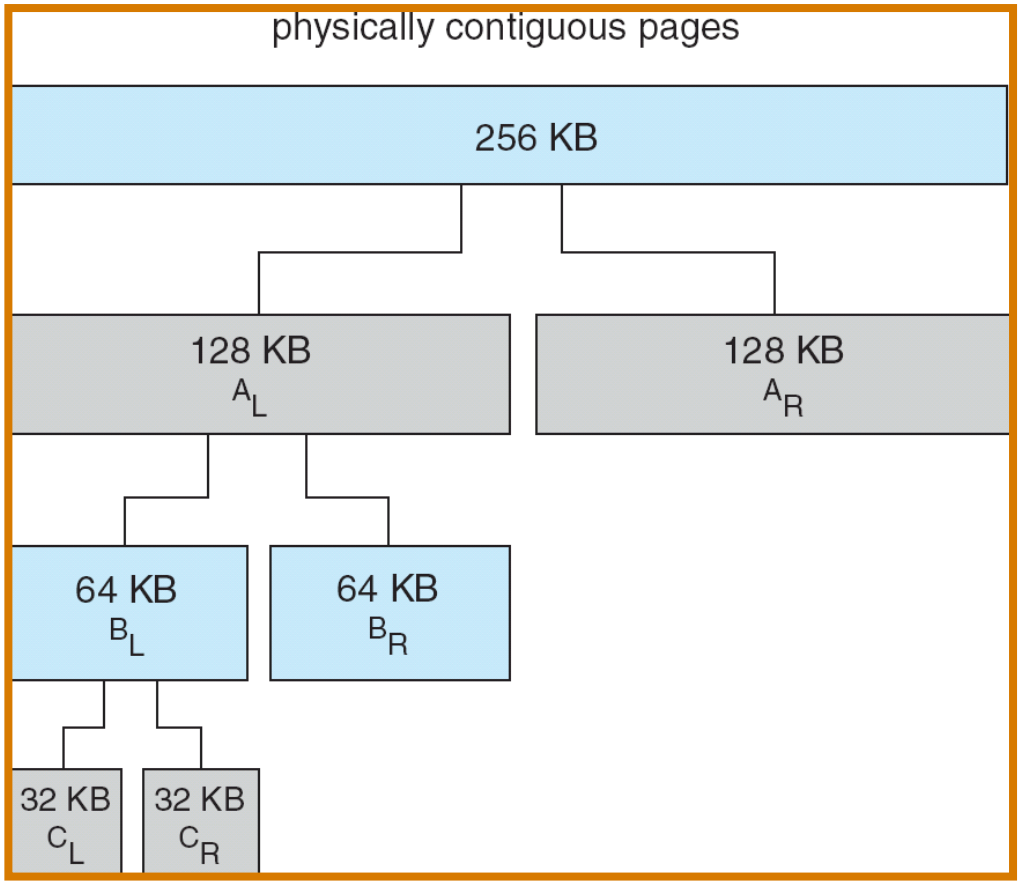
- Alokasi memori menggunakan segmen dengan ukuran segmen tetap. Segmen berisikan page yang berurutan (*contiguous*)
- Alokasi memori menggunakan berkelipatan dua
  - kebutuhan memory disesuaikan dengan besar segmen (kelipatan dua) yang memenuhi
  - Segmen dibagi menjadi dua bagian, jika segmen terlalu besar untuk memenuhi kebutuhan memory proses kernel. Pembagian terus dilakukan hingga besar potongan segmen memenuhi.

Buddy System



- Kelebihan : *buddy* yang berdekatan dapat digabung ->*coalescing*
- Kekurangan : fragmentasi internal

Buddy System Allocator



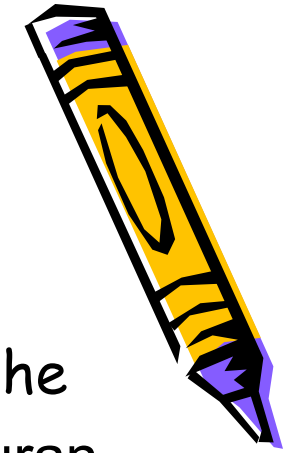
Buddy system allocator

Jika besar segmen = 256Kb, kernel membutuhkan 21kb, segmen dibagi menjadi dua *buddy,* salah satu *buddy* dibagi kembali hingga *buddy* memenuhi kebutuhkan kernel.

Slab Allocator

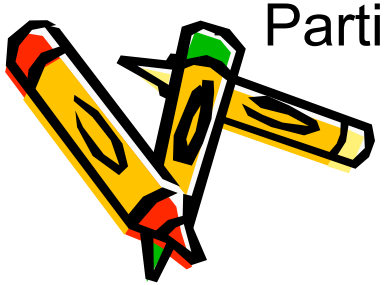
- **Slab** terdiri dari satu atau lebih page memory fisik yang berurutan
- **Cache** terdiri dari satu atau lebih *slab*
- Satu cache untuk setiap struktur data kernel (unik)
  - Setiap cache berisikan objek. Objek dapat merepresentasikan instance dari semaphore, process descriptor, dan sebagainya
  - Saat cache dibuat, objek tersebut kosong dan bebas digunakan

Slab Allocator



- Jumlah objek tergantung dari besar slab dalam cache
  - Misalnya 12Kb *slab* (terdiri dari 3 page dengan ukuran page masing 4kb) dapat menampung enam objek (satu objek besarnya 2kb)
- objek diberi tanda **used**, jika sedang digunakan
- Tiga status slab dalam linux
  - Full, semua objek dalam slab used
  - Empty, semua objek dalam slab *free*
  - Partial, slab terdiri dari objek yang used dan free
- Pencarian slab :

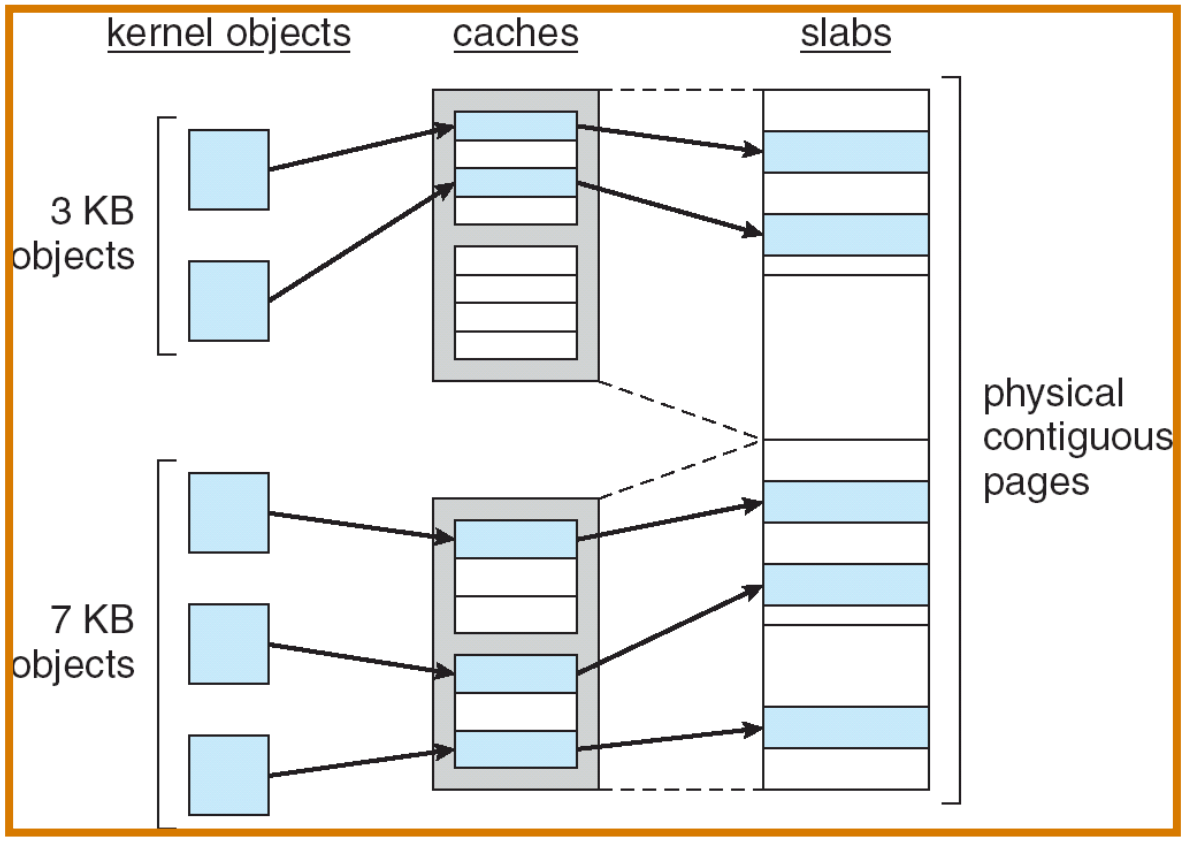
  Partial -> empty -> buat slab baru



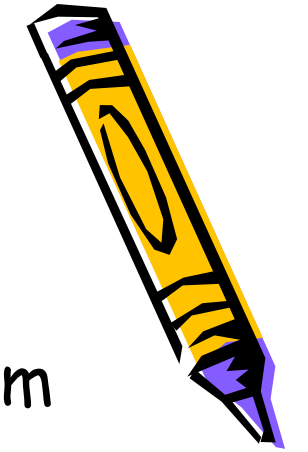
Slab Allocator

- Keuntungan
  - Tidak ada fragmentasi
    - Setiap struktur data kernel mempunyai *dedicated cache* masing-masing
  - Permintaan memori dapat dipenuhi dengan cepat
    - Objek dibuat diawal pembuatan cache
    - Proses selesai -> object diset free

Slab Allocation

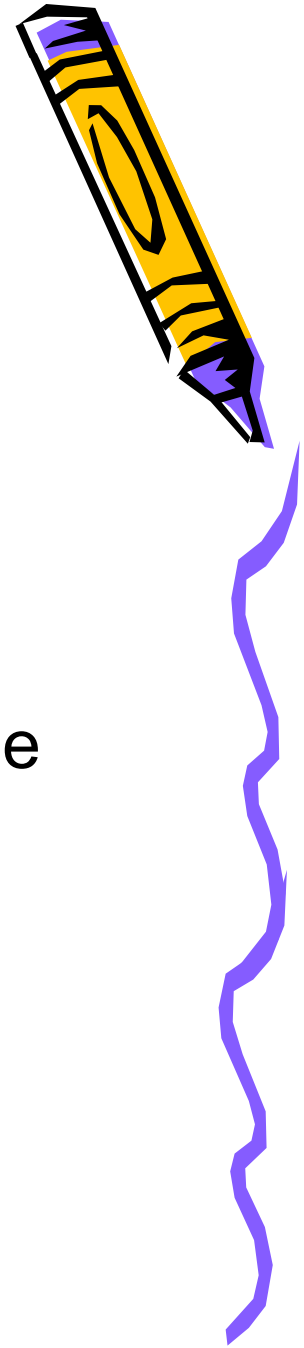


Topik tambahan



- Hal yang menjadi pertimbangan dalam sistem paging (telah dibahas sebelumnya)
  - Algoritma pergantian page
  - Alokasi frame
- Hal-hal lain
  - Prepaging
  - Ukuran page
  - TLBReach
  - Struktur Program

Prepaging



- Prepaging
  - Mengurangi terjadinya sejumlah *page fault* yang terjadi saat startup (pure demand paging)
  - Mengambil semua page yang dibutuhkan dalam sekali waktu sebelum referensi page terjadi
  - Jika terdapat banyak *page* yang tidak digunakan, I/O dan memory terbuang

Prepaging

- Jika jumlah prepaging sebanyak page *s* dan dari sejumlah page *s* yang digunakan sebanyak *α* page
  - $0 <= \alpha <= 1$
  - Jika $s * \alpha > s * (1 - \alpha) \Rightarrow$ cost-efektif dalam mengurangi page fault
  - Jika $s * \alpha < s * (1 - \alpha) \Rightarrow$ biaya prepaging lebih tinggi dibanding tanpa prepaging
  - *α* mendekati $0 \Rightarrow$ prepaging gagal
  - *α* mendekati $1 \Rightarrow$ prepaging efektif

Ukuran Page

- Faktor yang menjadi pertimbangan untuk menentukan ukuran page
  - Fragmentasi internal
    - Ukuran page kecil → mengurangi fragmentasi internal
  - Besar page table
    - ukuran page kecil → jumlah page bertambah → besar page table bertambah

Ukuran Page

- Waktu I/O
  - I/O overhead (read/write page)
    - komponen waktu I/O : seek, latency, transfer time
    - Jika transfer rate = 2mb/s=0.2 ms untuk mentransfer 512 bytes
    - seek time=20 ms, latency time= 8ms
    - total waktu I/O = 28,2ms ->512bytes
    - jika 2 page @512 byte butuh waktu I/O 56,4 ms
    - Jika 1 page @1024 byte butuh waktu I/O 28,4 ms, artinya ??

Ukuran Page

- Locality
  - Besar proses = 200 kb; yang dieksekusi=100 kb
  - Jika ukuran page > 200 Kb, eksekusi parsial (100kb) tidak dapat dilakukan
    - Terdapat page yang tidak digunakan
    - Butuh memori besar
  - Jika ukuran page < 200 Kb (misal = 1 byte), eksekusi parsial (100kb) dapat dilakukan
    - Keuntungan ukuran page besar: Butuh sedikit memori untuk menampung page table & waktu I/O sedikit
    - Kerugian ukuran page kecil: Page-fault tinggi & overhead

TLB Reach

- TLB Reach – Jumlah memory yang dapat diakses melalui TLB
- TLB Reach =(Besar TLB) X (Ukuran Page)
- Idealnya *working set* setiap proses disimpan dalam TLB
  - Jika tidak, frekuensi akses ke page table lebih tinggi dibanding ke TLB

TLB Reach

- Cara meningkatkan TLBReach :
  - Menambah ukuran page
    - Fragmentasi bertambah.
    - Beberapa aplikasi tidak membutuhkan ukuran page yang besar
  - Menggunakan ukuran page yang bervariasi (*multiple page size*)
    - Aplikasi yang membutuhkan ukuran page lebih besar dapat menggunakan tanpa menambah fragmentasi

Struktur Program

- Program structure
  - `int[128,128] data;`
  - Setiap baris disimpan dalam satu page
  - Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

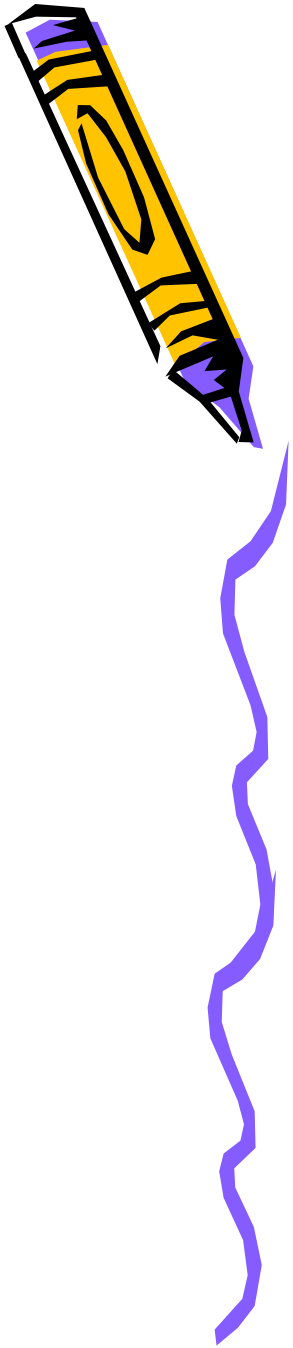
128 x 128 = 16,384 page faults

  - Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

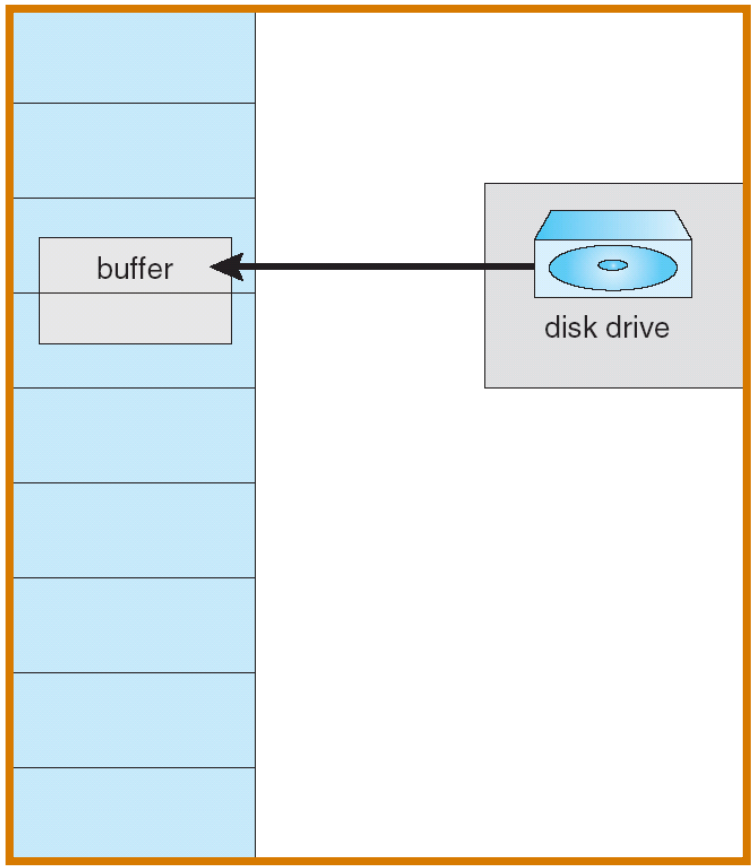
128 page faults

I/O Interlock



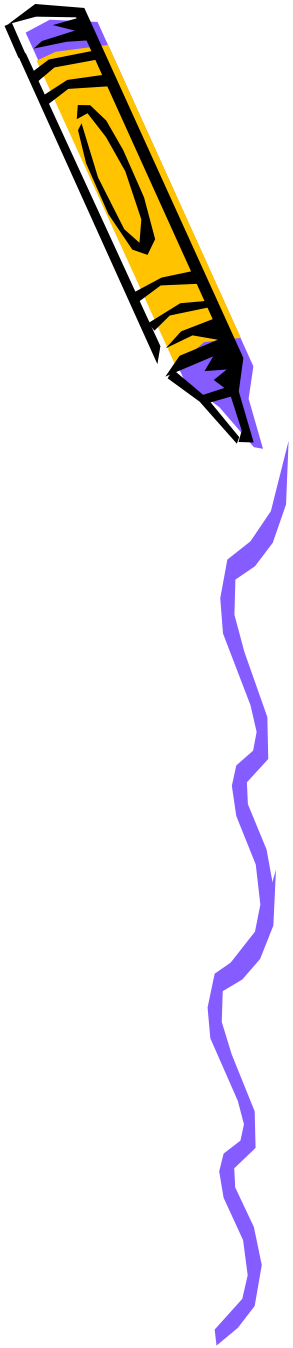
- **I/O Interlock** – Kadang page harus di *lock* didalam memory
- page yang digunakan untuk menduplikasi file dari sebuah *device* harus di *lock* agar tidak dikeluarkan oleh algoritma pergantian page.

I/O interlock



Frame I/O berada di memori

Sistem Operasi



- Windows XP
- Solaris

Windows XP

- Menggunakan demand paging dengan **clustering**. Clusterning memindahkan page yang terkena page fault namun juga beberapa page dengan faulting page.
- Setiap proses working **working set minimum** and **working set maximum**
- Working set minimum adalah jumlah page minimal yang akan diperoleh proses dalam memori
- Jika memori memenuhi, proses dapat memperoleh **working set maksimum**
- Ketika sejumlah memori kosong dibawah batas, **automatic working set trimming** untuk mengembalikan sejumlah memori kosong
- Working set trimming bekerja dengan memisahkan page suatu proses yang melebihi working set minimum