Das a-Kernel Projekt

Übersicht

1. Vorstellung a-Kernel Projekt
   1. Ziele
   2. AspectC
2. Beispiel 1: Prefetching in FreeBSD v3.3
   1. Zugriffsart normal
   2. Zugriffsart sequential
3. Beispiel 2: NFS
   1. Erweiterung Konsistenz
   2. Erweiterung Performance

Das a-Kernel Projekt

- Ziele:
  - Anwendbarkeit von aspektorientierter Programmierung im Betriebssystem testen
  - Beispielhafte Implementierung
  - Vergleich mit „klassischem“ Ansatz
- Implementierungssprache: AspectC

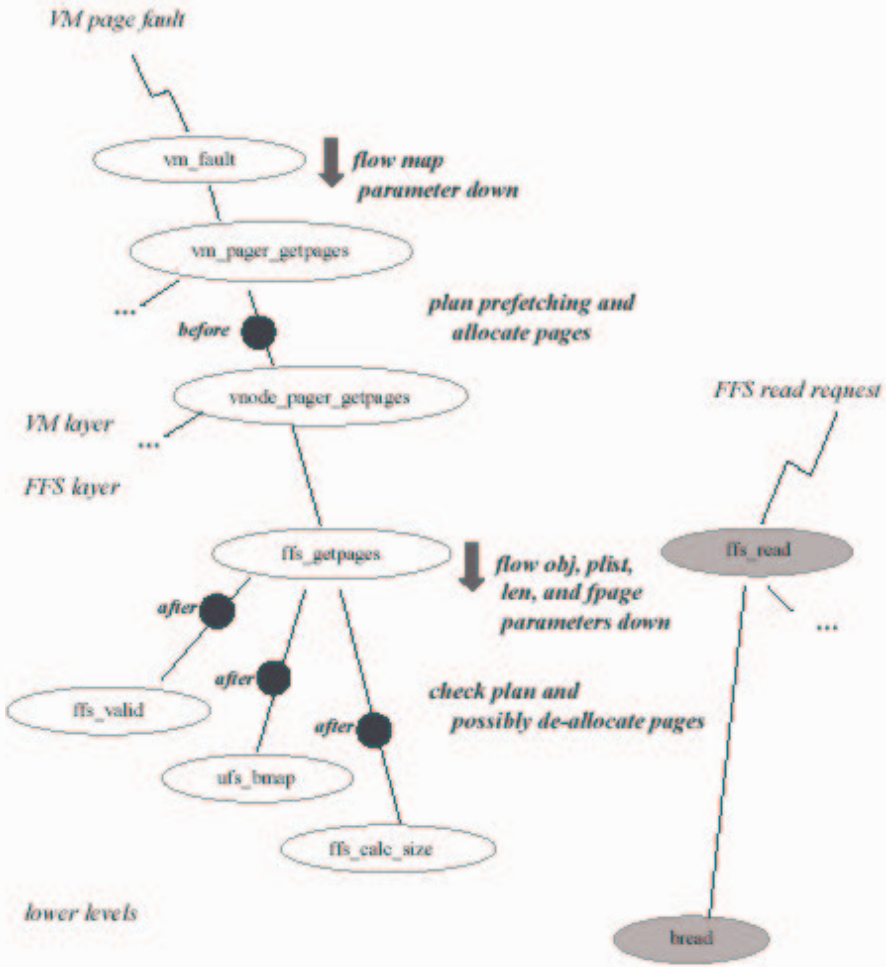
Das a-Kernel Projekt – AspectC

- AspectC bis heute noch nicht implementiert
- Kompilierung erfolgte deshalb per Hand
- AspectC ist angelehnt an AspectJ
- Sprachkonstrukte:
  - Deklarierung von Aspekten
  - Deklarierung von Pointcuts
  - before/after/around advice
  - Syntax sollte aus Beispielen klar werden

Beispiel 1: Prefetching

- Zugriff auf virtuellen Speicher löst Page Fault aus
- Mit Page Fault Handling verbunden: Prefetching
  - ➢ Effizienzsteigerung
- Verschiedene Zugriffsarten:
  - normal
  - sequential

Prefetching: normal



Prefetching: normal (1)

- Prefetching in Page Fault Handler planen
- Seite, die Page Fault ausgelöst hat, laden
- Danach überprüfen, ob geplante Seiten für Prefetching noch von Interesse. Prefetching nicht durchführen wenn:
  - Seite bereits im Systempuffer
  - „Page Fault“ Seite nicht auf Platte
  - Prefetching zu teuer (Seiten nicht zusammenhängend)

Prefetching: normal (2)

```
aspect normal prefetching {
    pointcut vm_fault_cflow(vm_map_t map):
            cflow(calls(int vm_fault(map,..)));
    pointcut ffs_getpages_cflow(vm_object_t obj, vm_page_t* plist, int len, int fpage):
            cflow(calls(int ffs_getpages(obj,plist,len.fpage)));

    before(vm_map_t map, vm_object_t obj, vm_page_t* plist, int len, int fpage):
            calls(int vnode_pager_getpages(obj,plist,len,fpage)) && vm_fault_cflow(map)
    {
            if(obj->declared_behaviour == NORMAL) {
                    vm_map_lock(map);
                    plan_and_alloc_normal(obj,plist,len,fpage);
                    vm_map_unlock(map);
            }
    }
```

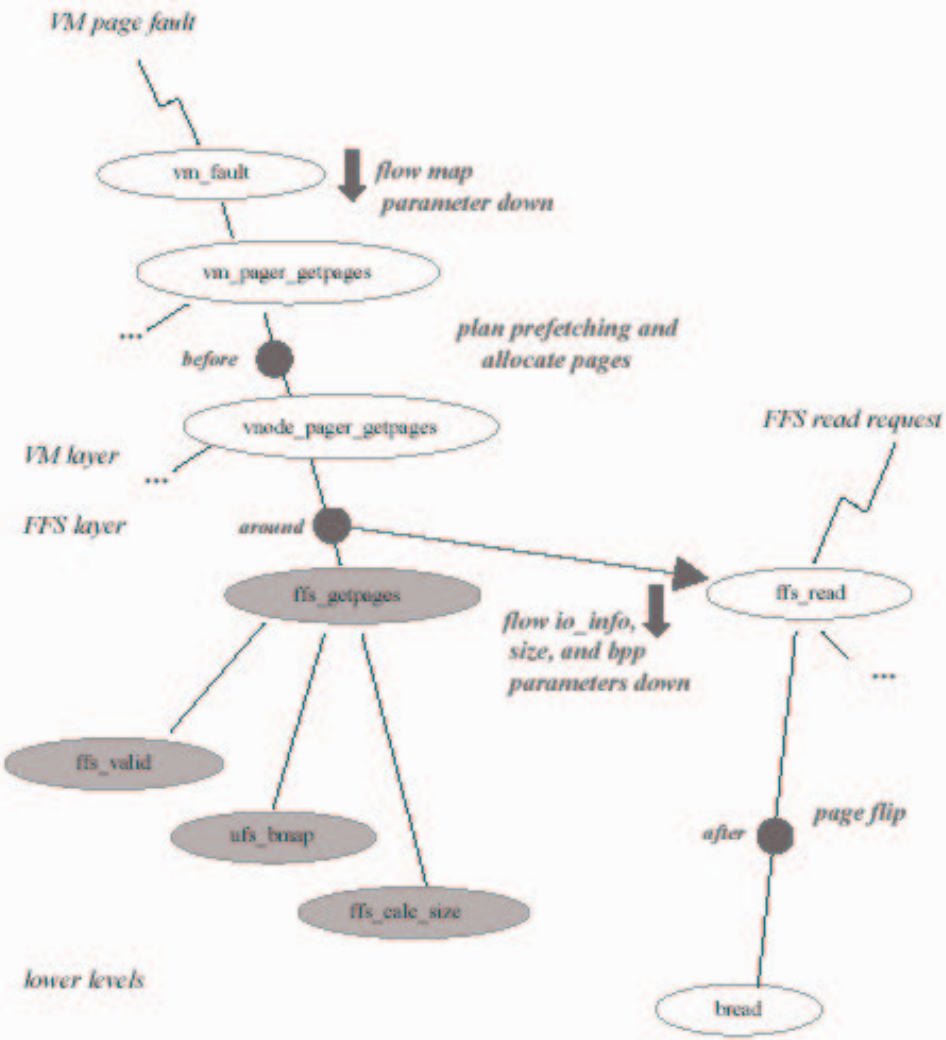
Prefetching: normal (3)

```
    after(vm_object_t obj, vm_page_t* plist, int len, int fpage, int valid):
            calls(valid ffs_valid(..)) && ffs_getpages_cflow(obj,plist,len,fpage) {
            if(valid) dealloc_all_prefetch_pages(obj,plist,len,fpage);
    }
    after(vm_object_t obj, vm_page_t* plist, int len, int fpage, int error, int reqblkno):
            calls(error ufs_bmap(struct vnode*, reqblkno, ..)) &&
            ffs_getpages_cflow(obj,plist,len,fpage) {
            if(error || (reqblkno == -1)) dealloc_all_prefetch_pages(obj,plist,len,fpage);
    }
    after(vm_object_t obj, vm_page_t* plist, int len, int fpage, struct t_args* trans_args):
            calls(int ffs_calc_size(trangs_args)) && ffs_getpages_cflow(obj,plist,len,fpage)
    {
            dealloc_noncontig_prefetch_pages(obj,plist,len,fpage,trans_args);
    }
}
```

Prefetching: sequential

- Komplettes Lesen einer bestimmten Anzahl von Blöcken
- Zugriffspfad wird „umgeleitet“
- Graphik und AspectC Code nächste Folie:

Prefetching: sequential (1)



Prefetching: sequential (2)

```
aspect sequential_prefetching {
    pointcut vm_fault_cflow(vm_map_t map):
        cflow(calls(int vm_fault(map,..)));
    pointcut ffs_read_cflow(struct vnode* vp, struct uio* io_inf, int size, struct buff** bpp):
        cflow(calls(int ffs_read(vp,io_inf,size,bpp)));
    before(vm_map_t map,vm_object_t obj,vm_page_t* plist,int len,int fpage):
        calls(int vnode_pager_getpages(obj,plist,len,fpage)) && vm_fault_cflow(map) {
            if(obj->declared_behaviour == SEQUENTIAL) {
                vm_map_lock(map);
                plan_and_alloc_sequential(obj,plist,len,fpage);
                vm_map_unlock(map);
            }
        }
```

Prefetching: sequential (3)

```
around(vm_object_t_obj, vm_page_t* plist, int len, int fpage):
  calls(int ffs_getpages(obj,plist,len,fpage)) {
    if(obj->behaviour == SEQUENTIAL) {
      struct vnode * vp = obj->handle;
      struct uio* io_inf = io_prep(plist[fpage]->pindex, MAXBSIZE,curproc);
      int error = ffs_read(vp,io_inf,MAXBSIZE,curproc->p_ucred);
      return cleanup_after_read(error,obj,plist,len,fpage);
    } else proceed;
  }
  after(struct uio* io_info,int size,struct buf* bpp):
    calls(int bread(..)) && vm_fault_cflow(..) &&
    ffs_read_cflow(struct vnode*,io_info,size,bpp) {
      flip_buffer_pages_to_allocated_vm_pages((char *)bpp->b_data,size,io_info);
  }
}
```

Prefetching: Diskussion

- „Klassische“ Implementierung über viele Module verteilt (Aspekt schneidet Schichten vertikal)
- AspectC Code sehr kompakt
  - Übersichtlichkeit
  - Erweiterbarkeit
  - Konfigurierbarkeit zur Übersetzungzeit!

Beispiel 2: NFS

- Verteiltes Dateisystem auf Client-Server Architektur
- Zwei konkurrierende Probleme:
  - Konsistenz
  - Performance
- Wünschenswert: Konfigurierbarkeit des Verhältnisses von Konsistenz zu Performance je nach Anwendungszweck
- a-Kernel Projekt erweitert Originalimplementierung in beiden Gebieten

NFS: Erweiterung Konsistenz

- Auf Clientseite:
- Client vergleicht Zeitstempel seiner Kopie mit Original auf Server
  - Inkonsistenzen in bestimmtem Zeitintervall möglich
- Zeitintervall soll minimiert werden

NFS: Erweiterung Konsistenz (1)

- Auf Serverseite:
- Jede Datei hat Zustand
- Änderung des Zustands bei jeder Lese-/Schreiboperation

NFS: Erweiterung Konsistenz (2)

```
aspect validation_check {
    pointcut validation_check_points(file_id fid):
            calls(int nfs_client_read(fid,..)) || calls(int nfs_client_write(fid,..)) ||
            calls(int nfs_client_cache_check(fid,..)) ||
            calls(void daemon_write(fid,..)) ||
            calls(void daemon_prefetch(fid,..));

    before(file_id fid): validation_check_points(fid) {
            get_fresh_timestamp(fid);
    }
}
```

NFS: Erweiterung Konsistenz (3)

```
aspect active_invalidation {
    after(file_id fid): calls(int nfs_server_read(fid,..)) {
            update_state_info(fid->state_info);
    }
    after(file_id fid): calls(int nfs_server_write(fid,..)) {
            check_state_and_invalidate(fid->state_info);
    }
}
```

NFS: Erweiterung Performance

- Ähnlich lokalem Fall: Aggressives Prefetching bei sequentiellem Zugriff

NFS: Erw. Performance (1)

```
aspect sequential_prefetch {
    /* client-side advice */
    before(file_id fid, block_num bnum): calls(int nfs_client_read(fid,bnum,..)) {
            update_pattern_of_access(fid,bnum);
    }
    after(file_id fid, block_num bnum): calls(int nfs_client_read(fid,bnum,..)) {
            if(fid->access_pattern == SEQUENTIAL)
                    nfs_daemon_prefetch(fid,bnum+1);
    }
    /* server-side advice */
    after(file_id fid, block_num bnum): cflow(calls(void nfs_daemon_prefetch(..))) &&
      calls(int nfs_server_read(fid,bnum,..)) {
            aggressive_prefetch_into_server_cache(fid,bnum+1,bnum+MAX_PREFETCH);
    }
}
```

Fazit

- a-Kernel Projekt weist Anwendbarkeit aspektorientierter Programmierung auch in komplexen Softwarearchitekturen nach
- Vorteile:
  - Übersichtlichkeit, Wartbarkeit
  - Erweiterbarkeit
  - Konfigurierbarkeit

Literatur

- http://www.cs.ubc.ca/labs/spl/projects/a-kernel.html
- Titel der einzelnen Papers siehe Ausarbeitung