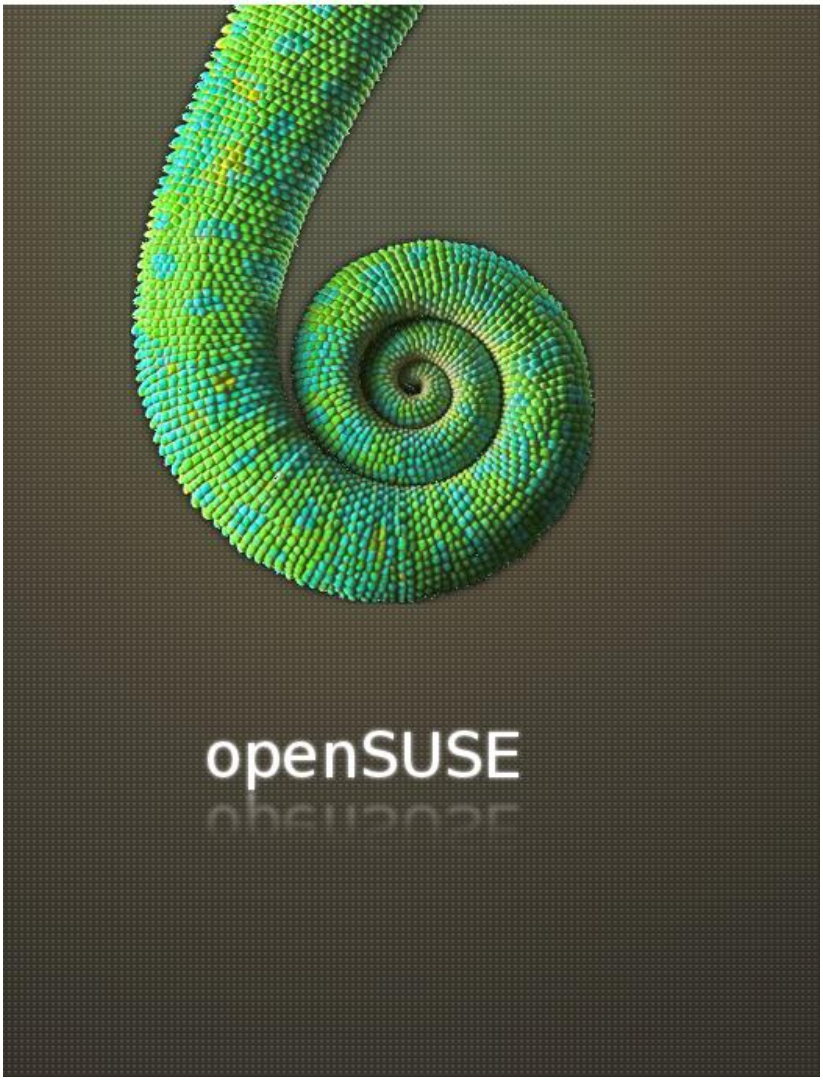

10.DEZEMBER 2013

Betriebssysteme Praktikum 3 Gruppe 2
Virtuelle Speicherverwaltung

LABOR PROTOKOLL

Dies ist das Protokoll zum dritten Laborversuch und umfasst eine kurze Beschreibung der Aufgabe, sowie eine Darstellung des Entwurfs und Diskussion des Ergebnisses. Anbei legen wir vier Logs

Steffen Giersch & Maria Lüdemann

HAW Hamburg

INHALTSVERZEICHNIS

DAS PROBLEM DER SPEISENDEN PHILOSOPHEN

AUFGABENSTELLUNG

> Schreiben sie eine Anwendung die virtuellen Speicher initialisiert und verwaltet

Die Speicherverwaltung besteht aus mehreren Komponenten. Die vmappl Komponente die aus vmappl.c und vmaccses.c besteht kümmert sich um das initialisieren des Speichers und schreibt und liest die Pages aus den Frames. Sind Pages nicht präsent reicht vmaccess.c die Anfrage zur mmanage Komponente durch die dann die gewünschten Seiten aus dem Speicher holt und sie bereit stellt bzw. Auslagert wenn sie nach den Kriterien des jeweiligen Algorithmusses ausgelagert werden sollte. Es wurden drei Algorithmen implementiert die zur mmanage Komponente gehören das sind FIFO, CLOCK und CLOCK1 die sich in ihrer Effizienz unterscheiden.

DER ENTWURF

Steffen Giersch, Maria Lüdemann
Betriebssysteme Praktikum Nummer 3 - Pseudocode

```
##### vmaccess #####

function vm_init
  binde an geteilten Speicher an
end function


function vm_read (address)
  wenn vmem noch nicht initialisiert ist
    rufe vm_init auf

  page_num = address / VMEM_PAGESIZE

  Wenn das Present-Flag der Page nicht gesetzt ist
    schreibe die benötigte Page in vmem
    Schicke ein SIGUSR1 an mmanage
    Warte auf sema auf mmanage

  physAddr = frameDerPage * VMEM_PAGESIZE + address % VMEM_PAGESIZE
```

```
  Setze entsprechende Used_Flags

  return vmem.data[physAddr]
end function


function vm_write(address, data)
  wenn vmem noch nicht initialisiert ist
    rufe vm_init auf

  page_num = address / VMEM_PAGESIZE

  Wenn das Present-Flag der Page nicht gesetzt ist
    schreibe die benötigte Page in vmem
    Schicke ein SIGUSR1 an mmanage
    Warte auf sema auf mmanage

  Setze das Dirty-Flag und die Used-Flags

  physAddr = frameDerPage * VMEM_PAGESIZE + address % VMEM_PAGESIZE

  vmem.data[physAddr] = data
end function


##### mmanage #####

function main
  rufe init_pagefile auf

  lege das logfile an

  rufe vmem_init auf

  Initialisiere den Signalhandler mit der Funktion sighandler

  endlosschleife
    pause

    Wenn das Signal SIGUSR1 kam
      signal_number = 0
    Wenn das Signal SIGUSR2 kam
      signal_number = 0
    Wenn das Signal SIGINT kam
      rufe cleanup auf
```

```
    breake aus der Endlosschleife
  Bei allen anderen Signalen
    Gebe eine Nachricht darüber aus

end function


function sighandler(signo)
  signal_number = signo

  Wenn das Signal SIGUSR1 kam
    rufe page_fault auf
    gib sema für vmaccess wieder frei
  Wenn das Signal SIGUSR2 kam
    rufe make_dump auf
end function


function page_fault
  Inkrementiere den pf_count in vmem

  Rufe find_free_frame auf und speichere das Ergebnis in free_frame

  Wenn kein freies Frame gefunden wurde
    Rufe einen der find_remove-Algorithmen auf und speichere das Ergebnis in to_delete
    Rufe store_page mit to_delete auf
    Setze free_frame = to_delete

  Rufe fetch_page mti free_frame auf

  Schreibe Logdaten
end function


function vmem_init
  Erstelle den Geteilten Speicher und verbinde mit ihm
  Schreibe Initialwerte in den geteilten Speicher
end function



function store_page(pt_index)
  old_page = pagenum im frame an Position pt_index
  Setze das Present-Flag von old_page auf 0
```

```
  Wenn das Dirty-Flag bei old_page gesetzt ist
    Setze es auf 0
    Schreibe die Daten in die Pagefile
end function


function fetch_page(pt_index)
  old_page = pagenum im frame an der Position pt_index
  Setze das Present-Flag von old_page auf 0

  Lade vmem.req_pageno an die Stelle von pt_index

  Setze das Present und Used-Flag auf 1 und das Dirty-Flag von req_pageno auf 0
  Aktualisiere in der Framepage an der Stelle pt_index die geladene Seite auf req_pageno
end function


function find_remove_fifo
  Gib den aktuellen Wert von vmem.adm.next_alloc_idx zurück und inkrementiere ihn danach
end function


function find_remove_clock
  endlosschleife
    Wenn bei der Page an der Stelle next_alloc_idx das Used-Flag nicht gesetzt ist
      Gib den Frame-Index zurück und inkrementiere ihn danach
      Abbruch der Endlosschleife
    Ansonsten
      Setze das Used-Flag der Page auf 0
      Inkrementiere den Frame-Index
end function


function find_remove_clock2
  endlosschleife
    Wenn bei der Page an der Stelle next_alloc_idx das Used-Flag nicht gesetzt ist
      Gib den Frame-Index zurück und inkrementiere next_alloc_idx danach
      Abbruch der Endlosschleife
    Ansonsten wenn das Used1 Flag gesetzt ist
      Setze es auf 0
      Inkrementiere next_alloc_idx
    Ansonsten
      Setze das Used-Flag auf 0
      Schalte next_alloc_idx einen weiter
end function
```

function cleanup
  Trenne vom geteilten Speicher

  schließe das logfile und das pagefile
end function

function init pagefile(String)
  Öffne das Pagefile mit read- und write-Berechtigung
  Schreibe VMEM_PAGES * VMEM_PAGESIZE Default-Werte in das pagefile
end function

PROBLEME/ BESONDERHEITEN DER ABGABE

Im Laufe der Bearbeitung vielen uns einige Punkte auf die sehr schwer lösbar, bzw. nur mit unschönen Workarrounds lösbar waren. So machten auch die Signalhandler einige Probleme und die Wahl des Rechners auf dem das Programm läuft hatte ebenfalls faszinierende Auswirkungen.

1. Auf einem Heimrechner wurden die ersten 16 Stellen auf 1 gesetzt und erst ab der 17 stelle wurde mit 2,3… weiter gezählt. Dieses Problem trat auf den Uni Rechnern nicht mehr auf. Woher das kam ist uns unerklärlich.
2. Beim write mussten wir ein usleep von 10 Millisekunden einfügen da sonst der erste Eintrag eine Zahl war die nicht dabei sein dürfte (267) Das sieht man im Log logfile_clock_ohne_usleep .Außerdem wird der nullte Index falsch befüllt. Wir vermuten dass dies an fwrite und fread liegt da das vmaccess blockiert und die Funktionen dann vermutlich Signale verschicken die vmappl aufwecken. Jedoch sicher sind wir uns nicht.

DIE ALGORITHMEN

Die Aufgabe verlangte die Implementation dreier Algorithmen FIFO, CLOCK und CLOCK1 in unserem Beispiel unterscheiden sie sich in ihrer Effizienz nur marginal jedoch ist ein Unterschied sichtbar der sich bei anderen Beispielen noch deutlicher hervortun kann.

So erzeugt FIFO 459 Seitenfehler, CLOCK 445 und CLOCK1 443 Seitenfehler.