

Prüfungsprotokoll zum Kurs 1802: Betriebssysteme
Prüfer: apl. Prof. Dr. Christian Icking
Beisitzerin: Dr. Lihong Ma
Datum: 22. Sept. 2011

Vor Beginn der Prüfung stellte Prof. Icking ein paar Fragen (woher ich komme, was ich zuvor studiert habe usw.), die noch nichts mit der Prüfung zu tun hatten, aber halfen, das Eis zu brechen, und die mir ermöglichten, ohne Erfolgsdruck ins Reden zu kommen. Nach 2-3 Minuten Smalltalk überprüfte er meine Identität (Ausweise) und begann die Prüfung.

Was sind Prozesse?

Wichtig ist die Unterscheidung Programm – Prozess. Während ein Programm ein in einer Programmiersprache ausformulierter Algorithmus ist (vgl. Kuchenrezept), ist ein Prozess ein ablaufendes Programm (vgl. Vorgang des Kuchenbackens).

Nun muss das Betriebssystem die Prozesse ja irgendwie verwalten. Wie wird das gemacht?

Jedem Prozess wird neben einem Programm-, einem Daten- und einem Stacksegment ein Prozesskontrollblock zugewiesen. Dieser enthält eine Prozess-ID, den Prozessorstatus (Inhalt der Register und des Befehlszählers) und Prozesskontrollinformationen (Prozesszustand, Priorität, Informationen zur Speicherbelegung, Informationen zu den geöffneten Dateien, Buchhaltung).

Sie haben die Prozesszustände genannt – welche Prozesszustände gibt es? Können Sie mir die Zustände und die Übergängen zwischen den Zuständen aufzeichnen?

Prozesszustandsübergangsdiagramm aufgezeichnet und erläutert.

Was bedeutet präemptiv?

Das ist die Bezeichnung für eine Variante des Scheduling, bei der der Prozess nicht selbst entscheiden kann, wann er den Prozessor abgibt, sondern das Betriebssystem die Möglichkeit hat, den Prozessen den Prozessor zu entziehen.

Welche Qualitätsmerkmale gilt es denn beim Scheduling zu berücksichtigen?

Minimale Antwortzeit, minimale Durchlaufzeit, maximaler Durchsatz, maximale Effizienz (Prozessorauslastung), Fairness.

Wie kann denn nun das Betriebssystem einem Prozess den Prozessor entziehen? Wie wird das umgesetzt?

Das Betriebssystem teilt dem Prozess, der in den Zustand rechnend versetzt wird, eine Zeitscheibe zu. Nach Ablauf der Zeitscheibe wird eine Unterbrechung ausgelöst. In der Unterbrechungsbehandlung sichert der Dispatcher den Prozessorzustand, reiht den Prozess in die Warteschlange ein und beauftragt den Scheduler, einen neuen Prozess auszuwählen; für diesen Prozess wird der Prozessorzustand wiederhergestellt und ihm wird der Prozessor zum Rechnen übergeben.

Wie wird denn entschieden, wie groß die Zeitscheibe ist, die dem Prozess zugeteilt wird?

(Ich glaube im Nachhinein, Icking wollte schon hier darauf hinaus, dass das Quantum nicht zu klein und nicht zu groß gewählt werden soll (s.u.), aber ich habe an dieser Stelle stattdessen das rechenzeitabhängige Feedback-Scheduling erklärt:)

Eine gute Möglichkeit, wie das gemacht wird, ist das rechenzeitabhängige Feedback-Scheduling. Dabei wird einem Prozess vorerst ein kleines Quantum und eine hohe Priorität zugeteilt; wenn dieses Quantum aufgebraucht wird (der Prozess also nicht vorher blockiert), dann wird ihm als nächstes ein größeres Quantum und eine niedrigere Priorität zugeteilt. Der Prozess kommt dann zwar später dran, kann dafür aber länger am Stück rechnen. Dieses Verfahren wird verbunden mit „aging“, d.h. bei jedem Prozesswechsel wird die Priorität aller wartenden Prozesse erhöht, sodass auch Prozesse mit niedriger Priorität irgendwann zum Zug kommen und nicht verhungern.

Was ist denn das einfachste Verfahren, um präemptives Scheduling umzusetzen?

Round Robin. Dabei wird gemäß einer FIFO-Liste bzw. effizienter umgesetzt mit einem Ringpuffer jedem Prozess abwechselnd dasselbe Quantum zugeteilt.

Was passiert denn, wenn dieses Quantum zu klein oder zu groß gewählt wird?

Zu klein: Der Prozessor ist zu viel mit dem Prozesswechsel beschäftigt, für die Prozesse bleibt zu wenig Zeit zum Rechnen; zu groß: Prozesse müssen u. U. sehr lange darauf warten, rechnen zu können. Das Quantum bewegt sich daher üblicherweise im Millisekundenbereich.

Neben den Prozessen gibt es in manchen Betriebssystemen ja auch noch Threads. Können Sie mir erläutern, was das ist und worin die Unterschiede zu den Prozessen bestehen.

Threads werden auch als leichtgewichtige Prozesse bezeichnet und unterscheiden sich von Prozessen dadurch, dass sich die zusammen gehörigen Prozesse eines Threads das Programm- und das Datensegment teilen, also nur einen eigenen Registersatz (inkl. Befehlszähler) und einen eigenen Stack haben. Vorteile von Threads sind, dass sie weniger Verwaltungsaufwand verursachen (insbesondere Benutzer-Threads, bei deren Umschaltung nur der Registerinhalt ausgetauscht werden muss), dass sie weniger Speicher verbrauchen (wegen des gemeinsamen Programm- und Datensegments), dass sie (bei Kernel-Threads) auf Mehrprozessorsystemen auf unterschiedlichen Prozessoren ablaufen können.

Was passiert, wenn ein Thread blockiert ist?

Das kommt darauf an: Wenn es sich um Benutzer-Threads handelt, dann ist der gesamte Prozess blockiert (also auch die anderen Threads dieses Prozesses), weil das Betriebssystem ja von den Threads gar nichts weiß; wenn es sich hingegen um Kernel-Threads handelt, können die anderen Threads weiterarbeiten, weil sie wie Prozesse dem Scheduling unterliegen.

Was ist nun der eigentliche Vorteile von Threads gegenüber Prozessen?

(An dieser Stelle war ich unsicher und habe etwas herumgestammelt; ich habe einige vorher genannte Vorteile von Threads wiederholt, weiß aber nicht mehr, ob ich letztendlich selbst auch gesagt habe, dass die Threads über den gemeinsamen Datenbereich kommunizieren können, oder ob das Icking gesagt hat. Jedenfalls wollte er das hören: Der Hauptvorteil von Threads ist, dass sie auf gemeinsame Daten zugreifen können und darüber gut kommunizieren können.)

Nun können ja, wenn verschiedene Threads oder Prozesse auf gemeinsame Daten zugreifen, gravierenden Probleme entstehen. Welche sind dies?

Es kann zu einer Race Condition kommen, d. h. dass das Ergebnis des Ablaufs konkurrierender Prozesse davon abhängt, in welcher Reihenfolge die Prozesse die CPU zugeteilt bekommen.

Und wie kann man das verhindern?

Die einfachste, allerdings mit Problemen behaftete Variante, ist jene mit globalen Synchronisationsvariablen. Dabei wird in in der einfachsten Variante gespeichert, welcher Prozess als nächster in den kritischen Abschnitt eintreten darf. Allerdings können in dieser einfachen Variante die Prozesse nur abwechselnd in den kritischen Abschnitt eintreten. Vor allem aber leidet das Verfahren der globalen Synchronisationsvariablen unter dem Busy Waiting, sodass unnötig Prozessorrechenzeit vergeudet wird.

Nun gibt es ja auch noch die Möglichkeit von Semaphoren. Wie funktionieren diese?

Semaphore sind ganzzahlige Variablen, die mit einer positiven Zahl (oft: 1) initialisiert werden. Bevor ein Prozess in den kritischen Abschnitt eintritt, wird eine down-Operation ausgeführt, die die Variable um 1 herabsetzt. Sinkt der Wert unter Null, so reiht sich der Prozess in die Warteschlange ein und verbraucht keine Rechenzeit mehr, bis er aufgeweckt wird. Am Ende des kritischen Abschnitts wird die up-Operation ausgeführt. Dabei wird der Semaphor erhöht; ist er danach nicht positiv, dann wird ein Prozess aus der Warteschlange aufgeweckt, damit dieser in den kritischen Abschnitt eintreten kann.

Bei Semaphoren tritt zwar in den up- und down-Operationen auch Busy Waiting auf, dieses ist aber nur sehr kurz und kann vernachlässigt werden.

Dieser Vorgang wird durch einen Maschinenbefehl unterstützt. Können Sie mir den erklären?

Der Befehl ist atomar und heißt „Test and Set Lock“ (TSL). Dabei wird der Inhalt des Speicherwort LOCK in ein Register eingelesen und der Wert des Speicherworts auf 1 gesetzt. Wenn der Inhalt des Registers 0 anzeigt, kann der Prozess dann in den kritischen Abschnitt eintreten, ansonsten muss er warten.

So ungefähr. Worauf es v. a. ankommt, ist, dass dieser Befehl unteilbar ist und alles in einem Rutsch erledigt.

Sie interessieren sich ja auch für die Programmierung. Können sie da den make-Befehl brauchen? Wozu?

In einem Makefile werden targets definiert, die die Abhängigkeit von Teilen eines Programms von den Quellmoduln (u. U. hierarchisch aufgebaut) repräsentieren. Beim Aufruf des make-Kommandos wird nun der Zeitstempel der Quelldateien mit den Zieldateien verglichen. Wenn die Quelldatei neuer ist, dann wird dieser Teil des Programms neu übersetzt, ansonsten nicht.

Ja, genau, wobei dabei natürlich wichtig ist, dass sich make um die ganze Kette von Abhängigkeiten, auch über mehrere Stufen hinweg, kümmert.

An dieser Stelle bemerkte Icking, dass die Zeit um war und beendete die Prüfung. Ich wurde kurz hinausgeschickt, nach kurzer Wartezeit bekam ich die Beurteilung mitgeteilt. Mir war es ein Anliegen, an dieser Stelle Frau Lihong Ma noch einmal ausdrücklich für ihre hervorragende Betreuung des Kurses (Fragenkataloge, Beiträge in der Newsgroup) zu danken.

In meiner Sicht lief die Prüfung äußerst korrekt und in einer menschlich angenehmen Stimmung ab. Kleinere Ausrutscher wurden mir verziehen, wenn klar wurde, dass ich das grundlegende Konzept wirklich verstanden hatte.

Da es sich um ein Gedächtnisprotokoll handelt, kann ich natürlich nicht für die genaue Formulierung und Abfolge der Fragen und Antworten garantieren. Außerdem habe ich meine Antworten an manchen Stellen etwas gerafft, an anderen Stellen hab ich vielleicht in der Prüfung nicht ganz so strukturiert geantwortet wie hier wiedergegeben. Aber einen ungefähren Eindruck von Ablauf und Detailgrad der Prüfung sollte dieses Protokoll wohl wiedergeben können.

Betriebssysteme (1802)

Prof. Icking / Dr. Ma

11.11.2011

Note: 1,0

Prof. Icking und Dr. Ma wirken beide unglaublich sympathisch. Vor der Prüfung nimmt er sich Zeit für Small-Talk und strahlt absolute Ruhe und Wohlwollen aus. Prof. Icking stellt sehr allgemeine Fragen und läßt einem dann in Ruhe erzählen, was man sich dazu zurecht gelegt hat. Freies Reden ist sehr wichtig, wird aber auch belohnt. Beide nicken freundlich motiviert, wenn man etwas richtiges gesagt, so dass einem gleich von Anfang an die Angst genommen wird.

1. Festplatte

- Aufbau
- Programm-/Interrupt-gesteuerte Kommunikation, DMA-Controller
- Ablauf eines Lesezugriffs (das Schema aus dem Skript vor Augen gehabt und erläutert)

2. Dateisysteme

- FAT, i-node, NTFS erklärt

Prof. Icking: Was ist der Nachteil, wenn für i-nodes der Festplattenbereich festgelegt ist? A: Beschränkung der maximalen Anzahl von Dateien

3. Sicherheit bei UNIX

- Schutzbits erklärt und kleinen Exkurs zu den diskretionären Verfahren gebracht obwohl nicht gefragt

4. SETUID

- erklärt und Beispiel mit passwd erläutert

5. Bell-La Padula-Modell

- Damit hätte ich nie gerechnet! Hatte es gottseidank am Vortag nochmal gelesen

6. Synchronisation

- Synchronisationsvariablen, Semaphore, Monitore
- Erzeuger-/Verbraucher-Problem

Keine Fragen zu Prozessen und Hauptspeicherverwaltung!!!

Fazit: Toller Prüfer, super nett.=

Prüfung 1802 – Betriebssysteme

vom 14.10.11

Prüfer: **apl. Prof. Dr. Christian Icking**

Note: 1,3

Da ich der erste Prüfling an diesem Freitag morgen war, traf ich Professor Icking und Frau Dr. Ma noch im Mantel vor ihren Bürotüren an. Die Prüfung wurde dann in einem Seminarraum einen Stock höher durchgeführt. Obwohl der Weg zum Prüfungszimmer noch sehr stumm von statten ging, hat Prof. Icking es verstanden doch zu Beginn zunächst ein wenig themenunabhängig Smalltalk über die Anreise und ähnliches zu halten. Die eigentliche Prüfung begann mit der Einstiegsfrage:

Was ist ein Dateisystem?

Über abgeleitete verallgemeinerte Beschreibung von NTFS hangelte ich mich zu den i-nodes herüber und erwähnte kurz das grundlegend anders ja die FAT sei. Darauf schien Prof. Icking hinaus gewollt zu haben, denn hier hakte er nach:

Wie ist denn die FAT organisiert?

Ich habe kurz die Grundlegenden Eigenschaften aufgezählt, insbesondere die Tatsache, dass die FAT im RAM liegt und den Dateizugriff beschrieben. An passender Stelle unterbrach mich Prof. Icking und sagte:

Die FAT ist ja nun zentral organisiert, andere Dateisysteme arbeiten dezentral, Sie hatten ja schon die i-nodes erwähnt, worin liegt denn hier der eigentliche Unterschied?

Ich habe daraufhin diesmal ausführlich die i-Nodes erklärt, und nochmal Details der FAT dargelegt, um den Unterschied zentral/dezentral sauber herauszuarbeiten musste mich Prof. Icking mit sehr offenen aber doch hilfreichen Fragen ein wenig schubsen. Die Formel für die maximale Dateigröße beim Einsatz von i-Nodes hätte ich auswendig parat haben sollen, ich habe sie schnell hergeleitet (soweit man $10+x+x^2+x^3$ herleiten kann). Über den von mir erklärten Aufbau eines Verzeichnissystems kam er dann zur Rechteverwaltung:

Wie ist denn das mit diesen Zusatzinformationen, welche Rechteverwaltung ist bei UNIX im Dateisystem integriert?

Von den üblichen Lese-Schreib-Ausführen-Rechten der UNIX-Dateien bin ich dann gleich zu den SETUID-Rechteübertragungen gewandert. Daraufhin fragte er:

Welche Alternativen zu dieser rudimentären Rechteverwaltung gibt es denn?

Ich habe über ACL und ihren Einsatz in Windows-Netzen erzählt, bin auch auf die Art der Speicherung im NTFS-Dateisystem eingegangen. Daraufhin gab es einen Themensprung:

Wenn nun mehrere Programme ausgeführt werden sollen, so müssen die Hauptspeicher erhalten, wie hat man das früher realisiert?

Ich habe MFT und MVT erläutert bin auf externe und interne Fragmentierung eingegangen, die bereits angerissenen Speicherzuweisungsstrategien habe ich ausführlicher erläutert und habe im Anschluss auf das Paging übergeleitet, hier hakte er wieder nach:

Wenn nun eine Speicheradresse angesprochen werden soll, wie wird das im Detail erledigt?

Hier habe ich mir wohl den größten Patzer geleistet, da ich zwar die Funktion der MMU und das generelle ansprechen der Speicheradressen bei Paging erläuterte, da ich aber den Begriff

Seitenrahmen nicht selbst brachte und aufgrund dieser Unsicherheit ein klein wenig ins Rudern kam, ging diese Erklärung lange nicht so flüssig über die Bühne. Nach Beschreibung der Seitenverdrängungsstrategien inklusive Vor- und Nachteile dieser und den Umsetzungen dieser landeten wir beim Seitenflattern, daraus leitete ich dann über zu den Deadlocks, die ich noch schnell zusammenfasste (Bedingungen und Gegenmaßnahmen) aber dann war die Zeit auch schon deutlich abgelaufen.

Alles in allem kann ich den vorherigen Protokollschreibern insofern zustimmen, dass die Prüfungssituation im wesentlichen wie ein Fachgespräch ablief. Wobei man nicht denken sollte, dass man wirklich eine Möglichkeit hätte auf die abgefragten Themengebiete Einfluss zu nehmen, es scheint mehr so, dass Prof. Icking einen gewissen Katalog im Kopf hat, was er gerne erklärt haben möchte und dann lenkt er das Gespräch entsprechend in die Richtung. Man hat also sehr gute Chancen selbst viel frei zu erzählen. Wenn das Thema dann tief genug erläutert ist, springt er eben auch (siehe oben). Es bietet sich also absolut an zu den Verschiedenen Subthemen bereits probenhalber kleine Referate vorzubereiten. Ich selbst habe allerdings besonders von den Lernsitzungen per Skype profitiert, wo wir uns gegenseitig die Frage aus den Protokollen beantwortet haben.

Insgesamt kann man sagen, dass die Prüfung absolut angenehm, fair aber nicht leicht war. Es werden einem viele Chancen geboten zu zeigen, was man weiß und was man verstanden hat, aber es wird auch genau nachgehakt, wenn man unsicher ist. Dabei bleibt aber immer eine insgesamt offene sympathische Haltung bestehen. Ich denke aber auch, dass man natürlich für eine sehr gute Note auch entsprechend genau geprüft wird. Wobei Prof. Icking oft Redestränge (auch seine eigenen) mit "aber so genau wollen wir das gar nicht betrachten" abbrach, da waren wir dann aber schon immer sehr im Detail.

Dieses Protokoll ist ganz gewiss nicht vollständig, kurze Einwürfe wie die Frage nach der Größe einer Speicherseite, Sinn und Möglichkeiten von Kompaktifizierung und ähnlichem habe ich hier nicht erwähnt, da es einen übertrieben genauen Eindruck vermitteln würde, der bei einem Gedächtnisprotokoll ohnehin nicht gegeben ist. Wie gesagt, wir waren zum Teil sehr im Detail, und haben kaum etwas nicht betrachtet. Erstaunlicherweise wurde ich nicht nach **make** gefragt und auch die Eingangsfrage war nicht die nach den Aufgaben eines Betriebssystems.

Ein Tipp für alle die beabsichtigen ein Protokoll zu fertigen: Macht es gleich, selbst einen Tag später ist kaum noch ein aussagekräftiges Protokoll zu verfassen. Und: Macht es, es hat Euch geholfen, also helft anderen.

Welche Aufgaben hat ein Betriebssystem?

- Speicherverwaltung
- **Verwaltung der E/A-Geräte**
- **Schutz / Überwachung von Zugriffsrechten**
- **Entdeckung und Behandlung von Fehlern** (z.B. Div durch 0, Papierstau im Drucker, etc.)
- **Mehrprogrammbetrieb**
- **Prozesssynchronisation und Kommunikation**
- **Realisierung unterschiedlicher Betriebsarten**
- **Bereitstellung einer Kommandosprache**
- **Kostenabrechnung**
- **Administration**

Als eine der wichtigsten Aufgaben kann der **Mehrprogrammbetrieb** angesehen werden. Er ermöglicht, dass mehrere Prozesse **scheinbar parallel** laufen. "Echt" parallel können sie in einem Ein-Prozessor-System nicht laufen, da **ein Prozessor immer nur einen einzigen Prozess zu einem Zeitpunkt** bearbeiten kann. Sie laufen scheinbar parallel, als das sie sich **zeitlich überlappen**. Während ein Prozess z.B. auf eine Benutzereingabe wartet, kann ein anderer ausgeführt werden.

Wie verwaltet ein Betriebssystem den Speicher?

Es gibt logischen und **physischen Speicher**. Eine **physische Adresse** ist eine **reale Adresse** einer **Speicherzelle in einem Hauptspeicher**. Über die physische Adresse kann jede Speicherzelle einzeln adressiert werden.

Wenn **Programme im Benutzermodus** ausgeführt werden, werden oft **logische Adressen** verwendet, die mit Hilfe der **MMU (Memory Management Unit)** in physische Adressen **umgesetzt** werden. Der **logische Adressraum** ist dabei **meistens größer** (viel größer) als der physische Adressraum, da der **reale Hauptspeicher deutlich teurer** ist, **als der Plattenspeicher**, auf dem der logische Adressraum realisiert wird.

Physisch ist die MMU entweder **in der CPU integriert oder nahe** angeordnet, **logisch** gesehen liegt sie **zwischen CPU und Hauptspeicher**.

Bei der Speicherzuweisung muss man unterscheiden zwischen:

Einfach zusammenhängende Speicherzuweisung: Es befindet sich zu jedem Zeitpunkt **maximal ein Benutzerprozess im Hauptspeicher**. Hier ist die Speicherzuweisung einfach.

Und wenn sich mehrere Anwenderprogramme im Speicher befinden?

Das ist dann die mehrfach zusammenhängende Speicherzuweisung: **Mehrere Programme** und deren Daten werden **nebeneinander** aufgenommen. Hier gibt es verschiedene Zuweisungsstrategien für den Speicher. Man kann den Speicher fest in Segmente aufteilen (im Script wird das MFT genannt) oder den Speicher variabel zuweisen (Im Skript wird das MVT genannt).

MFT (multiprogramming with a fixed number of tasks) Hierbei werden die **Segmentgrößen**, in die Speicher eingelagert wird, **fix** gewählt. Leider ist es **fast unmöglich, günstige Segmentgrößen** zu finden, wodurch **interne Fragmentierung** entsteht. Diese kann jedoch durch **geschicktes Scheduling verringert** werden. Zunächst einmal wird einem Auftrag das **kleinste zur Verfügung stehende Segment** zugewiesen, also kein unnötig großes (**best-available-fit**). Der nächste Schritt besteht darin, einem Auftrag nur ein Segment zuzuweisen, das **nicht wesentlich größer als benötigt** ist (**best-fit-only**). Ein Auftrag muss dann ggf. warten. Eine falsche, bzw. ungünstige Segmentzuordnung ist jedoch nur dann ein **Problem, wenn kein Segmentwechsel möglich** ist (beim swapping (= Ein/Auslagerung von Prozessen)).

MVT (multiprogramming with a variable number of tasks) Beim MVT liegt die **Segmentgröße nicht fest** und wird erst dann bestimmt, wenn Speicher benötigt wird. Da die Segmentgröße dann genau an die benötigte Größe angepasst wird, kommt es nicht zu interner Fragmentierung. Dafür kann es jedoch zu erheblicher **externer Fragmentierung** kommen. Beim MVT gibt es verschiedene Speicherplatzvergabe-strategien, die entscheiden, wo eine Speicherplatzanforderung erfüllt wird:

First Fit: es wird **die erste verfügbare ausreichend große Lücke** genommen – bei kleinen Adressen entstehen besonders **viele kleine Lücken** – die **Suche nach großen Lücken dauert lange**. – i.d.R. am schnellsten

Next Fit: wie First Fit mit dem Unterschied, dass immer **hinter der vorherigen Lücke weitergesucht** wird und nicht von vorne.

Best Fit: es wird die **kleinste ausreichend große Lücke** genommen – Nachteil: Es entstehen **viele kleine Lücken**, die schwer nutzbar sind.

Worst Fit: die **größte Lücke** wird gewählt – Vorteil: Der **Rest kann am Besten wieder verwertet** werden. – Nachteil: **Wenig Platz für große Segmente**.

Buddy: Hauptziel ist die Vereinfachung der Verwaltung der Segmente. – Der zugewiesene **Speicherplatz wird immer auf die nächsthöhere 2er-Potenz aufgerundet** → **interne Fragmentierung** – Die Gesamtgröße des Speichers sei ebenfalls eine **2er-Potenz** und wird **halbiert** und eine obere und untere Hälfte, die man **buddies** (Kumpel) voneinander nennt. – **Jede Hälfte kann erneut in zwei zueinander gehörige buddies geteilt werden**. – Vorstellung am Besten als **Binärbaum** – Ein angefordertes Segment der Größe 2^x kann sofort gefunden werden. – Bei **Freigabe** eines Segmentes wird dieses wieder mit seinem **buddy zusammengelegt**, sofern dieser auch frei ist. – Vorteil: Anforderungs- und Freigabe-Operationen sind **effizient implementierbar**. – Nachteil: **Erhebliche interne und ggf. externe Fragmentierung**.

Bei **MFT** tritt vorwiegend **interne Fragmentierung** auf, bei **MVT** **externe Fragmentierung**.

Eine weitere Speicherzuweisungsstrategie ist Paging

Was ist interne und was externe Fragmentierung?

Interne Fragmentierung entsteht dadurch, dass einem Prozess ein **größerer Speicherbereich** zugewiesen wird, **als benötigt** wird.

Externe Fragmentierung bedeutet, dass **nicht zugewiesene Speicherbereiche zu klein** sind, **um die Speicherplatzanforderungen** von auf Ausführung wartenden Prozessen **zu erfüllen**.

Wie funktioniert Paging?

Der **logische Hauptspeicher** wird in **Einheiten einer bestimmten Größe** unterteilt, die **page (Seite)** genannt werden. Die **logische Adresse** wird wie folgt **in eine Seitennummer und einen Offset geteilt**:

Auch der **physische Speicher ist unterteilt** - hier nennt man die einzelnen Stücke **page frame (Seitenrahmen)**. **Seite und Seitenrahmen sind gleich groß**. Ein Seitenrahmen kann folglich genau eine Seite des logischen Hauptspeichers aufnehmen. Jetzt muss man sich nur noch merken, **in welchen Seitenrahmen welche Seite** steht. Diese Übersicht enthält die **Seitentabelle ST**. Die **Abbildung vom virtuellen Speicher auf eine physische Adresse** übernimmt die **MMU**.

Die **Seitennummer** wird als **Index** für die **Seitentabelle** benutzt. Da steht dann die entsprechende Seitenrahmennummer drin. Zu der Seitenrahmennummer kommt noch der **Offset** und man hat die **physische Adresse**.

Was passiert bei einem Seitenfehler?

Unter einem **Seitenfehler (page fault)** versteht man, dass sich eine **Seite beim versuchten Zugriff nicht im Hauptspeicher** befindet.

Die Abarbeitung des Befehls wird zunächst einmal abgebrochen und eine **Unterbrechung** wird ausgelöst. Nun wird die **Seite in einen freien Seitenrahmen eingelagert**.

Da dies u.U. dauert, wird ein **Prozesswechsel** durchgeführt, falls ein anderer Prozess ausführungsbereit ist.

Schließlich wird der **unterbrochene Prozess fortgesetzt** und der **abgebrochene Befehl erneut ausgeführt**, diesmal ohne Seitenfehler.

Von wo werden Seiten beim Paging eingelagert?

Vom **Swapbereich** auf einem **Sekundärspeicher**, also einer Festplatte.

Genau, das ist ja der Vorteil von Paging, so hat man viel mehr Platz verfügbar als der tatsächliche physische Hauptspeicher groß ist. Natürlich auf Kosten der Performance.

Ja, und Festplattenspeicher ist natürlich auch viel **billiger** als Hauptspeicher.

Wenn wir Seiten einlagern müssen wir Seiten auch wieder auslagern. Welche Strategien kennen Sie um Seiten auszulagern?

Es gibt dazu verschiedene Strategien. Eine ist **LRU (Least Recently Used)**. Man geht von der **Lokalität eines Prozesses** aus. Die Seiten die **als letztes häufig genutzt** wurden werden **auch in Zukunft wahrscheinlich wieder** genutzt werden, **die anderen kann man auslagern**.

LRU (Last Recently Used):

- eigentlich eine sehr gute Strategie

Wie würden Sie LRU implementieren?

LRU ist sehr **schwierig zu implementieren**. Da es schwierig ist immer zu kontrollieren welche Seiten häufig genutzt wurden. Man kann dazu das **R-Bit (Referenced-Bit)** benutzen.

Ja, aber das reicht nicht aus, Sie müssten einen Zeitstempel haben.

Okay. Es gibt noch weitere Strategien für die Auslagerung einer Seite:

Zugriffsbit:

- Idee ist ähnlich wie **LRU**, doch man merkt sich **nicht den exakten Zeitpunkt**, sondern eine **Beobachtungsperiode**.
- **Realisierung über Zugriffsbit (Referenced-Bit; kurz: R-Bit) - Speichern der R-Bits in Schieberegistern**

FIFO (First-In-First-Out):

- **leicht implementierbar**
- deutlich **höhere Seitenfehlerrate**

Second Chance:

- FIFO-Erweiterung mit **Ausgabepuffer**
- **Seiten werden nicht sofort ausgelagert**

Clock-Algorithmus:

- andere Implementierung von Second-Chance
- **Schlange wird durch zirkuläre Liste erreicht**
- **Seite eingelagert: R-Bit = 0 – Seite benutzt: R-Bit = 1** – Auslagerung geschieht wie folgt: Zeiger findet Seite mit **R-Bit=1** → **R-Bit=0**. Zeiger findet Seite mit **R-Bit=0** → **Auslagerung**

Was kann passieren wenn zwei Prozesse auf gemeinsam genutzte Bereiche zugreifen?

Wenn zwei (oder mehr) **überlappende Prozesse gemeinsam auf Speicher zugreifen** und mindestens **ein Prozess** davon **schreiben** und das **Ergebnis abhängig** ist von:

- der **Reihenfolge** der Ausführung der Anweisungen
- der **relativen Geschwindigkeit** der Prozesse

dann handelt es sich um eine **Race Condition**. Die **kritischen Abschnitte** der beiden Prozesse müssen dann **synchronisiert** werden.

Welche Arten der Synchronisation kennen Sie?

- **Synchronisationsvariablen**
- Wir definieren eine **globale Variable s**, über die ein Prozess den **kritischen Abschnitt** für den jeweils anderen freigibt.

- **Nachteil:** Die kritischen Abschnitte können **nur abwechselnd 1,2,1,2,... ablaufen** → kein Übereinander möglich und ein **Prozess kann nicht selbst zweimal hintereinander den gleichen kritischen Abschnitt aufrufen**.
- **Lösungsmöglichkeit:** **2 Synchronisationsvariablen** verwenden → Neuer **Nachteil:** Gefahr eines **Deadlock**
- **Peterson Algorithmus**
- **2 Variablen** zum **Anzeigen von Interesse** in den kritischen Bereich einzutreten.
- **1 Variable** hält fest **welcher Prozess dran ist**
- **Nachteil: Aktives Warten (busy waiting)**
- **Semaphore**
- Dijkstra hat daher das Konzept der **Semaphore** entwickelt. Eine Semaphore kann **als eine Variable betrachtet werden, die einen ganzzahligen Wert hat**. Mit Hilfe dieses Semaphores wird eine bestimmte kritische Ressource überwacht.
- **Initialisiert man s mit k**, so werden höchstens **k gleichzeitige Zugriffe auf diese Ressource zugelassen** → meistens ist $k=1$.
- Zur Benutzung der Semaphore stehen die **Operationen down und up** zur Verfügung.
- **down(s)** bedeutet, dass der aktuelle Wert der **Semaphore s um 1 vermindert** wird. Wird der **Wert negativ, blockiert der Prozess, der down ausführt**.
- **up(s)** **erhöht den Wert von s um 1**. Ist der neue **Wert nicht positiv**, wird ein durch eine down-Operation **blockierter Prozess freigegeben**, er geht folglich vom Zustand blockiert in **bereit** über.
- Sei **s der Semaphore** eines kritischen Abschnittes, so wird **down(s) vor und up(s) nach dem kritischen Abschnitt** ausgeführt.
- **Vorteil: hohe Geschwindigkeit, hardwareseitige Implementierung**
- **Nachteil:** up- und down-Operationen sind überall im Text verstreut, man verliert den **Überblick**
- **Nachrichtenaustausch**
- **Ringpuffer**
- **Sender-Empfänger-Prinzip**
- **Zugriff auf Puffer muss synchronisiert werden**
- Unterscheidung zwischen **Einweg-Kommunikation** (nur von A nach B) und **Zweiweg-Kommunikation** (Sender A erwartet von Empfänger B eine Bestätigung).
- **Monitore**
- **Vorteil:** alle **Funktionen nur im Monitor**, nicht überall verteilt
- **Unter einem Monitor versteht man eine Menge von Prozeduren, Variablen und Datenstrukturen, die als Betriebsmittel betrachtet und mehreren Prozessen zugänglich gemacht sind. Die gemeinsam genutzten Betriebsmittel können geschützt werden, indem sie im Monitor platziert werden.**
- Ein Monitor unterstützt die **Synchronisation durch Bedingungsvariablen**, auf die **zwei Funktionen** wirken:
 - **wait(c):** Prozess wird **blockiert**, wenn **c nicht erfüllt** ist; Prozess kommt in die **Warteschlange** der Bedingung c
 - **signal(c):** Wenn **in einer Monitorprozedur die Bedingung c wahr wird**, dann wird die **signal-Operation** aufgerufen. Ein **wartender Prozess wird ein aktiver**.
 - Monitor-Konzept ist vergleichbar mit einem **Raum**, zu dem es nur einen **einzigsten bewachten Eingang** gibt, so dass sich **immer nur ein Prozess im Raum** befinden kann. Andere Prozesse geraten in eine **Warteschlange** für blockierte Prozesse, die auf die Verfügbarkeit des Monitors (Änderung der Bedingung c) warten. Sobald ein **Prozess eingetreten** ist, kann er **sich selbst** auf Grundlage der Bedingung c **blockieren**, indem er **wait(c)** aufruft. Wenn ein **Prozess**, der **im Monitor** läuft, eine **Änderung von c** bemerkt, ruft er **signal(c)** aus.
 - **Monitor Hoare-Typ:** Einer der **wartenden Prozesse wird nun fortgesetzt**, der **signalisierende Prozess wird angehalten**. Dieser wird **fortgesetzt, wenn der wartende Prozess den Monitor verlässt**.
 - **Monitor Mesa-Typ:** Im Gegensatz zum Hoare-Typ **blockiert signal(c) den signalisierenden Prozess nicht**. Dieser wird stets fortgesetzt. **signal(c)** reiht stattdessen einen Prozess **von der Warteschlange der Bedingungsvariablen c in die Monitor-Warteschlange** um.

Was ist das Set UID Bit?

Setuid (Set User ID, manchmal auch **suid)** ist ein **Zugriffsbit** für Dateien oder Verzeichnisse des Unix-Betriebssystems. Ausführbare Programme bei denen dieses Bit gesetzt ist, werden **mit den Rechten des Besitzers einer Datei ausgeführt**, anstatt mit den Rechten desjenigen Benutzers, der die Datei ausführt.

Kennen Sie ein Beispiel für die Benutzung des Set UID Bits?

Die Datei, in der unter UNIX die Passwörter aller Benutzer gespeichert sind (`/etc/shadow`), kann **nur von root beschrieben** werden. Dennoch sollte **jeder Benutzer** die Möglichkeit haben, **sein eigenes Passwort zu ändern**. Dazu gibt es das Programm `/usr/bin/passwd`, bei welchem das **s-bit gesetzt** ist.

Der Prozess erhält beim Aufruf die **Rechte des Besitzers, also root, und kann die Passwortdatei ändern**. Das Programm wird aber **nur die Änderung des eigenen Passwortes** erlauben.

Was ist make und wie funktioniert es?

Ein Programm muss **compiliert** und **gebunden** werden.

Das Kommando **make** sorgt dafür, bei einer Programm-Aktualisierung **nur die von der Veränderung betroffenen Programmteile zu aktualisieren**, um das gesamte Programm auf den neuesten Stand zu bringen.

Dafür benötigt make eine **Beschreibung der Abhängigkeiten** zwischen den Dateien und die Definition der auszuführenden Aktion. Diese Angaben werden in einer Datei "**Makefile**" abgelegt. Das "Makefile" enthält verschiedene **Regeln** z.B. "all" oder "clean". Standardmässig wird "**all**" ausgeführt.

Die Regel "all" wird nicht defaultmässig ausgeführt - die erste Regel wird defaultmässig ausgeführt. Per Konvention ist "all" allerdings immer die erste Regel.

"make" ermittelt, **welche Dateien geändert** wurden, leitet aus den **im "Makefile" beschriebenen Abhängigkeiten** die notwendigen Aktionen ab und führt sie aus.

Das heißt, dass **das letzte Veränderungsdatum** eine wichtige Rolle bei make spielt.

Wie stellt make Veränderungen an Dateien fest?

make schaut in den **Dateiattributen**. Also liest die Modificationtime aus dem **I-Node**.

Genau, make vergleicht also nur Zeitstempel die sowieso schon da sind. Viele glauben nämlich das "make" ein eigenes Management der Zeitstempel führt.

Die Prüfung bei Dr. Icking war sehr angenehm.
Dr. Icking kann ich als Prüfer uneingeschränkt empfehlen!

Prüfungsprotokoll

Kurs: 1802 Betriebssysteme
Prüfling: Guenther Rasch
Prüfer: PD Dr. Icking / Dr. Lihong Ma
Datum: 21.07.2009 / 14:30 Uhr
Note: 1,7

Dateisysteme / Security / Kommandosprachen

- Was ist ein Dateisystem, welchen Zweck hat es?
 - *Definition + Transparenz schaffen! Verschiedene Arten von Dateisystemen aufgezählt*
- Könnte man auch ohne DS arbeiten? Wie funktioniert das dann?
 - *Arbeiten mit einem RAW-Device, Benutzer muss genau wissen, welche Blöcke wo stehen*
- Was ist FAT, wie ist es aufgebaut?
 - *Definition FAT, Vorteile / Nachteile*
- Ist denn das noch zeitgemäß?
 - *Nein, ist aktuell auch nicht mehr im Einsatz → Hinweis auf Größenverbrauch und dass FAT komplett im Hauptspeicher stehen muss.*
- I-node-basierte DS, wie sind die aufgebaut?
 - *Definition + Aufbau + Blockgrößen genannt. Beim Aufbau direkte, indirekte Adressierung erklärt, vorher noch die Attribute genannt.*
- Sie erwähnten Sicherheitsinfos, was ist das genau?
 - *Erklärung Sicherheitsbits rwx für User / Group / Other + Reihenfolge der Auswertung erklärt.*
- Wie kann man dann ein Programm mit anderen Rechten ausführen?
 - *SETUID-Bit erklärt*
- Make-Befehl, was macht er und wie?
 - *Definition make-Befehl. Hr. Dr. Icking wollte hier auch hören, dass er simpel zwei Daten bzgl. Änderungsdaten der voneinander abhängigen Dateien vergleicht.*

Prozesse:

- Was ist ein Prozess
 - *Definition (lange und kurze Form!) + Kuchenbackanalogie erklärt*
- Woraus besteht er und wie sieht der Hauptspeicher dazu aus?
 - *Aufbau des logischen Adressraums eines Prozesses erklärt, bei Erwähnung des PCB:*
- Was steht im PCB?
 - *Aufbau des PCB erklärt. Von den Prozesskontrollinformationen kamen wir dann auch schon zu den Zuständen:*
- Welche Prozesszustände gibt es
 - *Zustandsdiagramm aufgezeichnet und erklärt, wichtig hier die Übergangswechsel von bereit → rechnend und rechnend → bereit und erklärt warum hier beidseitig → präemptive / nicht-präemptive Systeme erklärt*
- Prozesswechsel, wie funktioniert das?
 - *Scheduler / Dispatcher erklärt, Aufgaben des Dispatchers, Schedulingstrategien, Aufgaben des Dispatchers aufgezählt*
- Qualitätsmerkmale für Scheduling
 - *5 Merkmale aufgezählt, mit Hinweis, dass man nicht alle 5 gleichzeitig optimieren könne*

- welche Auswirkung hat denn eine Festlegung des Quantums q ?
 - *Verschiedene Auswirkungen erklärt (q zu groß, q zu klein)*

Fazit:

Hr. PD Dr. Icking ist als Prüfer uneingeschränkt zu empfehlen! Ich hatte nun schon ein paar Prüfungen an der FU, bei dieser hatte ich die ganze Zeit das Gefühl, nicht in einer Prüfung, sondern in einem Fachgespräch zu sein. Sehr angenehm und für eine nervöse Natur wie mich absolut super! Weiter hat er sich, obwohl wir schon zu spät angefangen hatten, Zeit für ein kleines Gespräch vorab und auch nach der Notenbekanntgabe genommen!

Die Prüfung selbst lief eigentlich im Stile eines Gespräches ab; ich denke, es ist wichtig, die Themen von selbst wiedergeben zu können, z.B. bei den i-node-basierten Dateisystemen: einfach mal von der Definition zu den Vorteilen, Nachteilen, wie sieht die Adressierung aus, etc... Hr. Dr. Icking unterbricht hier auch nicht, vielleicht für eine Zwischenfrage bzw. weiterführende / überführende Fragen.

Ich hatte anfangs bei FAT Schwierigkeiten, ebenso beim make-Befehl, die Benotung ist gerecht.

Fachprüfung Kurs 01802 „Betriebssysteme“

Prüfer: Dr. Icking

Beisitzer: Dr. Lihong Ma

Datum: 20.07. 2009

Dauer: 30 min

Note: 1,0

Skizze des Rechnersystems

Aufgaben des Betriebssystems

Aufgaben des Scheduler

Inhalte Prozesskontrollblock

Aufgaben des Dispatcher

Wie können Prozesse parallel laufen?

Wie kann CPU entzogen werden?

Scheduling-Algorithmen

Qualitätsmerkmale Scheduling-Strategien

Paging, MMU, TLB, Umrechnung von Adressen

Virtueller Speicher (Ziel und Realisierung)

Seitenfehler, Seitenauslagerungsstrategien

Fragmentierung

Welche Synchronisationsverfahren gibt es?

Probleme globaler Synchronisationsvariablen

Semaphoren

Vorteil von Monitoren

Deadlock (Definition + Beispiel + Bedingungen)

Erkennung, Beseitigung, Vermeidung und Verhinderung von Deadlocks

Ich kann Herrn Dr. Icking uneingeschränkt weiter empfehlen. Zumindest bis jetzt war ich kein Fan von mündlichen Prüfungen, und da es meine erste Fachprüfung war bin ich umso aufgeregter gewesen, jedoch versteht es Herr Dr. Icking einem die Nervosität zu nehmen.

Die Fragen lassen sich wenig beeinflussen, jedoch kann man ruhig etwas weiter ausholen, z.B.

Qualitätsmerkmale, TLB, Vorteile von Monitoren, Vogel-Strauß-Algorithmus waren nicht gefragt,

aber waren trotzdem gut verbrauchte 5 Minuten :)

Zur Prüfungsvorbereitung habe ich die Frageliste von Frau Dr. Ma beantwortet und zu kleinen Vorträgen zusammengefasst. Damit ist man sehr gut vorbereitet, bis auf die erste Frage sind alle aus der Frageliste.

Anmerkung: Das Protokoll zu schreiben hat weniger als 10 Minuten gedauert, soviel Zeit sollte sich jeder nehmen!

Prüfungsprotokoll 1802 – Betriebssysteme

Version: Belegt im Jahr 2004, angeblich gibt es ab SS 2006 eine neue Version

Prüfling: Andreas Buschka (MSc. Informatik-Studiengang)

Prüfung am: 24.03.2006 bei Dr. Icking (LG Prof. Haake)

Beisitzerin: Frau Dr. Lihong Ma

Die Begrüßung:

Sehr freundlich, stellte mich kurz seiner Beisitzerin vor. Wir redeten erst einmal ca. fünf Minuten über meine bisherigen Prüfungen und den aktuellen Stand meines Studiums, als es dann schonend in die ersten Fragen ging.

Die Fragen:

- Welche Aufgaben hat ein Betriebssystem?
Schließen der semantischen Lücke Mikroprogrammebene <-> Anwendungsebene
Verwaltung von Geräten
Sicherheit (Zugriffsrechte)
Bereitstellung von Betriebsarten (erwähnt: Batch-, Echtzeit- und Dialog-Betrieb)
Prozessverwaltung
Kostenabrechnung (erwähnt, dass das veraltet ist)

Nicht von mir erwähnt, es wurde aber auch nicht danach gefragt:
Verwalten von Betriebsmitteln (Drucker, Speicher, CPU)
Bereitstellung von Kommandosprachen
Unterstützung der Administration
- Was ist ein Prozess?
Unterschied zwischen Programm und Prozess erklärt (Programm = Blaupause, Prozess = konkrete Instanz des laufenden Programms)
Wichtigste Bestandteile des PCBs erklärt (Register, Programm-Counter, Prozess-ID)
- Welche Zustände kann ein Prozess haben?
Die fünf Zustände (erzeugt, bereit, aktiv/rechnend, blockiert, beendet) aufgezeichnet und die möglichen Zustandsübergänge dargestellt und erklärt
- Welcher Teil des Kerns ist dafür zuständig, den Wechsel zu einem anderen Prozess durchzuführen?
Dispatcher. Habe auch gleich den Scheduler erwähnt und dargestellt, dass der Dispatcher die konkrete Umschaltung macht, während der Scheduler für das Auswählen des nächsten Prozesses zuständig ist. Grundlegende Anforderungen an den Scheduler erklärt (Fairness, Effizienz, Antwortzeit, Durchsatz, Verweilzeit)
- Wie wird erreicht, dass der Scheduler aufgerufen wird? Ein Prozess könnte ja endlos weiterrechnen, und die anderen Tasks würden nie zum Zuge kommen.
- Unterbrechung durch Timer-Interrupt.
- Eine Strategie des Schedulers ist Round-Robin. Wie funktioniert das?
Round-Robin erklärt und ein Beispiel (Entnehmen des Kopfes, Einfügen am Ende) demonstriert. Auch gesagt, dass Round-Robin schwächen bei I/O-lastigen Prozessen hat und kurz angerissen, wie Round-Robin mit Prioritäten funktioniert)
- Hauptspeicherverwaltung erklären
Habe die verschiedenen Verfahren mit ihren Vor- und Nachteilen dargestellt (siehe Kurstext):
- MFT (Problem der internen Fragmentierung erklärt)
- MVT (Problem der externen Fragmentierung erklärt), kurz auf die Möglichkeit der Kompaktierung und deren Nachteile eingegangen

- Nicht-zusammenhängende Speicherverwaltung
- Virtueller Speicher, Ablauf der Aus-/Einlagerung von Seiten.
- Ein Mittel der Synchronisation von Prozessen ist der Semaphor. Woraus besteht er? Erwähnt, dass es eine statische globale Variable mit Wertebereich der natürlichen Zahlen mit Null ist und zusätzlich eine Liste aller wartenden Prozesse verwaltet. P und V-Operation erklärt.
- Wenn der Prozess im kritischen Bereich vom Scheduler unterbrochen wird, gibt es da keine Probleme mit der Semaphoren? Hier wusste ich nicht, worauf Dr. Icking hinauswollte. Ich fragte, ob Deadlocks gemeint seien. Er meinte, es ging ihm darum, ob die Operationen P und V atomar sein müssen. Ich bejahte dies, aber ich sei mir nicht sicher, ob der Prozess im kritischen Bereich (also nach Ausführung von P aber vor Ausführung von V) durch den Scheduler unterbrochen werden kann. Er erkannte das Mißverständnis (gefragt war nur der der Atom-Eigenschaft von P und V) und verwarf die Frage.
- Erklären sie das Deadlock-Problem! Die vier Kriterien für die Entstehung eines Deadlocks erklärt (Exklusiver Ausschluss, Nicht-unterbrechbarkeit, Warte-Bedingung, Circular Wait). Gefragt, ob ich das Bild mit den gegenseitig blockierenden LKW zeichnen soll, das wurde aber verneint.
- Was kann man gegen Deadlocks tun?
 - Einen Prozess als „Deadlock-Opfer“ auswählen und töten (den Begriff kannte ich aus dem Bereich Datenbanken, steht so nicht im Kurstext). Dadurch wird der Circular Wait durchbrochen.
 - Alle Ressourcen freigebene und zusammen mit der zusätzlich gewünschten Ressource neu anfordern.
 - Nach Betriebsmitteltypen und Bänker-Modell wurde nicht gefragt.

Dann war die Prüfungszeit rum.

Fazit: Die Prüfung lief so ab, dass Dr. Icking eigentlich nur sehr wenige Fragen stellte, und mich dann sehr lange ausreden lies. Ich kann sehr empfehlen, zu den Themen, die immer wieder vorkommen (siehe auch die anderen Prüfungsprotokolle) einige kleine Refarate vorzubereiten. Ich hatte nach der Prüfung das Gefühl, dass ich mich an einigen Stellen zu salopp ausgedrückt habe (z.B: ist der Vergleich mit Blaupause und Instanz beim Prozess bei strenger Prüfung nicht ganz richtig), aber das scheint nicht zu einer Abwertung zu führen, selbiges, wenn man zwar die wichtigsten, aber nicht alle Elemente einer Aufzählung erwähnt (z.B. bei den Aufgaben des Betriebssystems oder dem PCB). Ein Rechtsanspruch darauf besteht natürlich nicht ☺

Wichtiger scheint ihm zu sein, frei und längere Zeit einen Teilbereich vorstellen und beschreiben zu können. Daher kann ich auch nur die Vorbereitung von Mini-Reparaten sehr empfehlen, denn damit kann man sich locker 5-7 Minuten von seiner (hoffentlich) besten Seite zeigen.

Ich bin zwar selbst in mündlichen Prüfungen selten auf dem Mund gefallen, aber dass Dr. Icking sich viel Zeit vor dem Beginn der eigentlichen Prüfung nimmt und während der Prüfungen immer ruhig bleibt und versucht, zu helfen, wenn etwas nicht verstanden wurde, macht ihn speziell für Studierende mit Ängsten vor mündlichen Prüfungen empfehlenswert.

Nach der Prüfung unterhielten wir uns noch, er fragte mich, ob ich schon eine Abschlussarbeit im Master-Studiengang habe. Außerdem geht die Benotung absolut in Ordnung.

Fach: Betriebssysteme 1802
Studiengang: Master of Science in Informatik
Prüfer: PD Dr. Icking
Beisitzer: Lioung Ma
Datum: 09.02.2006
Ort: Hagen
Note: 1,3

Wie sieht eine Festplatte aus?

Wer greift auf die Festplatte zu?

Greift der Controller auf Dateien zu?

Wo werden die Dateien in Sektoren übersetzt?

Was ist ein hierarchisches Betriebssystem?

Zugriffszeiten?

In welcher Größenordnung bewegen sich diese Zeiten?

Beschreiben sie FAT und Inode.

Wie groß kann das Array von FAT werden?

Wie groß ist ein Inode?

Wie funktioniert das mit den indirekten Adressen?

Worauf zeigt die indirekte Adresse?

Was steht noch in den Inods?

Schutzbits?

Setuid bit?

Was ist ein Prozess?

Erklären sie die Prozesszustände!

Die Prozesse arbeiten also vermeidlich Parallel. Wie funktioniert das?

Was passiert bei der Prozessumschaltung?

Wieso gibt der Prozess den Prozessor ab?

Wie groß sollte das Quantum sein?

Prozesse fordern Hauptspeicher an. Manche fordern auch noch nach. Wie geht das?

Was ist Fragmentierung?

Was ist Paging?

Fragmentierung bei Paging?

Wie funktionieren Semaphoren.

Ich erhebe keinen Anspruch auf Vollständigkeit. Es waren wohl noch mehr Fragen. Für einen 1,3 darf man wohl auch einmal auf dem Schlauch stehen und eine kleine Sache nicht wissen. Ansonsten immer schön ausführlich antworten dann muss nicht nachgefragt werden! Herr Icking ist als Prüfer nur zu empfehlen! Aber man sollte schon den ganzen Stoff parat haben.

Viel Glück euch!

Kurs: 1802
Prüfer: Icking
Beisitzer: Li Mahong
Datum: 8.11.2004
Note: 1,3

- Aufgaben Betriebssystem
- Dateisystem
 - * FAT + Liste für freie Blöcke
 - * Inode
- Speicherverwaltung
 - * einfach
 - * mehrfach (MFT/MVT)
- Paging
 - * Mapping
 - * Auslagerungsstrategien
- Zugriffsbits Unix
- setuid-Bit

Mündliche Diplomprüfung Kurs 01802 „Betriebssysteme“

Prüfer: Dr. Icking

Beisitzer: Dr. Lihong Ma

Datum: Sep. 2004

Dauer: 28 min

Note: 1,0

Von Dr. Icking gab es noch kein Protokoll. Auch bei einer telefonischen Anfrage konnte ich ihm keine Eingrenzung des Prüfungsstoffes entlocken (wie man sieht, kamen auch alle KEs dran.).

Daher hier kurz der Ablauf der Prüfung, soweit ich mich erinnere:

Icking: Aufgaben des Betriebssystems?

8 Punkte mit kurzer Erläuterung aufgezählt.

I: Da haben wir ja eine Menge Anknüpfungspunkte für die weiteren Fragen. Was ist ein Dateisystem?

I: Was ist FAT und i-node? Wo ist der Zugriff auf z. B. das 1millionste Bit schneller?

Erläuterung; FAT ist lineare Liste, da kann Durchlauf lange dauern– andererseits ist der Zugriff schnell, wenn sie im RAM liegt. Bei i-node muss man max. 3 Schritte tief suchen, bis man den gesuchten Sektor hat.

I: Was ist ein Prozess?

I: Bitte Prozesszustände aufzählen und Bild aufmalen mit möglichen Zustandsübergängen.

Wieso geht Prozess nicht von „blockiert“ direkt über nach rechnend?

I: Welche Schedulingstrategien gibt es?

Alle preemptiven und nicht preemptiven aufgezählt und erläutert mit Vor- und Nachteilen.

I: Was macht man mit dem RAM, wenn man mehrere Programme gleichzeitig laufen lassen will?

Speicherzuweisungsstrategien aufgezählt (MFT, MVT, Swapping, Fragmentierung, paging).

I: Seitenauslagerungsstrategien beim Demand Paging?

LRU, Zugriffsbits, Fifo, Second chance Fifo +Erläuterungen. Dann wurde die Zeit auch schon langsam knapp...

I: Welche Möglichkeiten der Synchronisation?

Alle 4 aufgezählen

I: Was ist ein Monitor?

Definition + kurze Erläuterung.

I: Darstellung der Rechte bei Unix?

rwx—x-rw- etc. erläutert.

I: Was ist ein setuid-Bit

I: Wozu dient der Make-Befehl?

Ouch! Das war tatsächlich eine Frage aus KE 7. Bei der Erläuterung nicht vergessen, dass im Makefile auch der Compileraufruf (z. B. „cc filename“) stehen muss!

Ich habe zu jeder Frage Erläuterungen wie Vor- und Nachteile der Verfahren gebracht, wobei Dr. Icking mich nicht abwürgte. Frau Dr. Ma erinnerte nach 20 min. daran, dass die Zeit bald herum sei... Daher musste bei den letzten Fragen alles etwas kürzer gehen.

Mein Vorbereitungsweg: Alles lernen. Danach Protokolle auch der anderen Profs. durcharbeiten, damit man ein Gefühl für die Fragestellungen und generellen Schwerpunkte bekommt.

Fazit: Faire Prüfung, netter Prüfer, gerne wieder!