

Klausur Betriebssysteme

Kurs TIT07G1 / Halbj. 4

Aufg. 1: (10 Pkte)

Geben Sie zu den folgenden Aussagen an, ob die Aussage wahr oder falsch ist.

- (a) UNIX Prozesse können über einen Systemaufruf der exec(l,v,...)-Systemaufruffamilie erzeugt werden
- (b) Bei der Erzeugung eines UNIX-Kindprozesses erbt der Kindprozess vom Vater die gleiche Prozess-ID
- (c) Bei der Erzeugung eines UNIX-Kindprozesses erbt der Kindprozess die "Liste" der offenen Dateien des Vaterprozesses
- (d) Windows verwaltet die "Ressourcen" eines Prozesses in einer Objekttabelle, wobei jeder Eintrag in dieser Tabelle auf ein Resource-spezifisches Objekt weist.
- (e) Ein File-Handle ist bzgl. des Windows-Betriebssystems vergleichbar zu einer Inode eines UNIX-Dateisystems
- (f) Das PPID-Feld in einem Prozesslisting mit dem UNIX Kommando ps gibt die "private Prozess ID" des Prozesses an
- (g) Alle Windows-Prozesse müssen über den "explorer" erzeugt werden
- (h) In einem Betriebssystem mit Threadkonzept kann einem Prozess nicht mehr als ein Thread zugeordnet werden, aber ein und derselbe Thread kann von mehreren Prozessen verwendet werden.
- (i) Ein Prozess unter Linux kann über die Zeit gesehen mehr als ein Programm während seiner Laufzeit ausführen
- (j) Ein Prozess kann mehr als einem Nutzer gehören

Aufg. 2: (10 Pkte)

Erläutern Sie die wesentlichen Vor- bzw. Nachteile der Schedulingalgorithmen FCFS (First Come First Serve) und SJF (Shortest Job First). Mit welchem anderen Schedulingkonzept kann man den wesentlichen Nachteil des FCFS-Algorithmus deutlich entschärfen?

Aufg. 3: (10 Pkte)

Das Systemprogramm "Process Explorer", das wir in der Vorlesung benutzt haben, kann für einen Prozess u.a. die Größen "Virtual Size", "Working Set" und "Peak Working Set" anzeigen. Erklären Sie, was die Größen bedeuten. Es gibt Prozesse, bei denen "Virtual Size" sehr groß ist. Ist das problematisch? Und wie sieht es bei Prozessen aus, bei denen die Working-Set Größe sehr hoch ist (z.B. annähernd so groß wie der Hauptspeicher)?

Aufg. 4: (10 Pkte)

Die Ausgabe der virtuellen Speicherbereiche (Regions), die ein Prozess, der das Programm mmap abarbeitet, verwendet, könnte unter einem UNIX-Betriebssystem folgendermaßen aussehen:

Start-Endadresse	prot	Offset	gerät	Inode (Major:Minor)	Name der Datei (sofern aus Datei)
08048000-08049000	r-xp	00000000	03:02	535555	/home/duepmeie/programme/mmap
08049000-0804a000	rwp	00000000	03:02	535555	/home/duepmeie/programme/mmap
40000000-40014000	r-xp	00000000	03:02	533362	/lib/ld-2.3.2.so
40014000-40015000	rwp	00014000	03:02	533362	/lib/ld-2.3.2.so
40015000-40016000	r--p	00000000	03:02	342350	/etc/passwd
40029000-40158000	r-xp	00000000	03:02	533368	/lib/libc.so.6
40158000-4015c000	rwp	0012f000	03:02	533368	/lib/libc.so.6
4015c000-40160000	rwp	00000000	00:00	0	
bffffe000-c0000000	rwxp	fffff000	00:00	0	

In welchen Adressraumbereichen sind der Programmcode, die globalen Variablen und der Stack untergebracht? Das Programm mmap enthält intern eine lokale, automatische Variable fd (Filedescriptor). In welchem Adressbereich ist diese untergebracht? Was enthält der Speicherbereich 40015000 - 40016000?

Aufg. 5: (10 Pkte)

Geben Sie für die folgenden Aussagen zu Prozessen an, ob sie wahr oder falsch sind.

- (a) Der Addressraum eines Prozesses kann größer als der Hauptspeicher sein.
- (b) Einzelne Addressraumseiten müssen hintereinander im Hauptspeicher liegen.
- (c) Seitentabellen können Einträge enthalten, die auf keine Kachel im Hauptspeicher weisen.
- (d) Wenn ein Prozess einen zu großen virtuellen Addressraum gegenüber dem realen Hauptspeicher eines Rechners hat, stellt das ein Problem für das Betriebssystem dar.
- (e) Am Anfang des Bootens eines Rechners muss zunächst einmal virtuelle Adresse = physikalische Adresse sein
- (f) Große Programme starten deshalb so langsam, weil beim Starten zunächst der gesamte Addressraum des Programmes in den Hauptspeicher geladen werden muss.
- (g) Freie Kacheln, die für Addressraumseiten des User Stacks eines Prozesses allokiert werden, müssen mit lauter Nullen initialisiert werden.
- (h) Die Anfangswerte von globalen Variablen werden aus der Programmdatei gelesen.
- (i) Automatische Variablen, die nicht initialisiert werden, sind mit 0 initialisiert
- (j) Datenstrukturen, die mit "new" in objektorientierten Programmen angelegt werden, werden im Stack untergebracht.

Aufg. 6: (10 Pkte)

Beantworten Sie die folgenden Fragen zu Dateisystemen

- (a) Wo sind in einem FAT-Dateisystem die Verwaltungsdaten einer Datei untergebracht? Was ist mit dem Dateinamen? Befindet sich dieser an der gleichen Stelle?
- (b) Welche Information (Informationen) befinden sich in einem FAT-Eintrag? Sind das die aus a)?

Aufg. 7: (10 Pkte)

Die folgenden Textzeilen zeigen Teile der Ausgabe eines Linux ls-Kommandos.

```
-rw----- 1 clemens.duepmeier iai-sg-umweltinfosys 133285 2003-06-02 19:09 05-05-05.pdf
-rwxrwr--- 1 clemens.duepmeier iai-sg-umweltinfosys 1869517 2004-02-18 14:43
4_BARRIERE.pdf
drwxrwxr-x  9 clemens.duepmeier iai-sg-umweltinfosys      4096 2006-11-27 11:14 ADMIN
-rw-rw-r--  1 clemens.duepmeier iai-sg-umweltinfosys       840 2005-03-16 19:15 a.out
drwxrwxr-x  2 clemens.duepmeier iai-sg-umweltinfosys      4096 2005-03-16 19:17 assembler
```

Erläutern Sie, welche Informationen sie in den einzelnen Spalten der Ausgabe sehen und in welchen Verwaltungsdatenstrukturen eines Linux-Dateisystems sich diese befinden.

Aufg. 8: (10 Pkte)

- (a) Erläutern Sie, was ein symbolischer Link in Linux ist.
- (b) Wie interpretiert der Kernel symbolische Links, wenn er bei der Auflösung eines Pfadnamens darüber stolpert und was macht er, wenn der symbolische Link "ungültig" ist?

Aufg. 9: (10 Pkte)

Erläutern Sie die Basisarbeitsweise eines Programmierers mit Dateien (z.B. an einem UNIX-C-Programm mit Pseudocode oder textuell), wenn er in seinem Programm Daten sequentiell lesen/schreiben oder auch ab einer bestimmten Position lesen/schreiben will.

Aufg. 10: (10 Pkte)

Wie merkt sich ein Unix-Dateisystem, in welchen Festplattenblöcken sich der Inhalt der Datei befindet?

Musterlösung zur Klausur Betriebssysteme

Kurs TIT07G1, 4. Semester

Aufg. 1:

- (a) falsch
- (b) falsch
- (c) wahr
- (d) wahr
- (e) falsch
- (f) falsch
- (g) falsch
- (h) falsch
- (i) wahr
- (j) falsch

Aufg. 2:

Beim FCFS-Algorithmus werden alle Prozesse in der Reihenfolge ausgeführt, in der sie im System ankommen (erzeugt) werden. Der Vorteil ist, dass hierbei kein Prozess "verhungert". Allerdings "bremst" langandauernde Prozesse das gesamte Prozesssystem aus. Beim Shortest Job First Algorithmus werden dagegen die Prozesse, die am kürzesten rechnen, bevorzugt behandelt. Dies hat den großen Vorteil, dass der Durchsatz des Systems (Menge an Prozesse, die im System bearbeitet werden) optimiert wird. Allerdings garantiert ein reiner SJF-Algorithmus nicht, dass Prozesse nicht verhungern. Der SJF-Algorithmus ist also nicht fair. Der wesentliche Nachteil des FCFS-Algorithmus, dass langandauernde Prozesse das gesamte System "ausbremsen" kann dadurch entschärft werden, dass man das "Zeitscheiben-Konzept" einführt. Mit diesem Konzept können lange Prozess nur kleine Teile ihrer Gesamtrechenzeit am Stück rechnen. Und das Betriebssystem nimmt zwischendurch immer mal wieder kleinere, schnellere Prozesse dran.

Aufg. 3:

Die "Virtual Size" gibt an, wieviel virtuelle Adressraumseiten (Adressraum) das Programm gerade tatsächlich im Adressraum abgebildet hat. Diese müssen aber längst nicht alle im Hauptspeicher sein. Die "Working Set" Größe gibt an, wieviel von diesem Adressraum tatsächlich im Hauptspeicher untergebracht ist und sollte in der Regel wesentlich kleiner sein. Die "Peak Working Set" Größe gibt an, welche maximale Einlagerungsgröße der Adressraum des Prozesses bislang im Hauptspeicher erreicht hat. Die "Working Set"-Größe sollte in der Regel klein gegenüber der "Virtual Size" sein, und es spricht in der Regel nichts gegen einen großen Adressraum, solange die "Working Set" Größe klein bleibt. Ist die Summe der "Working Set"-Größen aller im System aktiven Prozesse größer als der Hauptspeicher, muss das System swappen. Die Performance von Prozessen lässt dann drastisch nach.

Aufg. 4:

Der Programmcode ist in dem Adressraumbereich 08048000 - 08049000, das globale Datensegment im Bereich 08049000 - 0804a00 und das Stacksegment im Datenbereich bffffe000-c0000000 untergebracht. Die Variable fd ist eine lokale, automatische Variable und damit auf dem Stack realisiert.: also im Bereich von bffffe000 - c0000000. Der angegebene Adressbereich ist durch Aufruf eines mmap()-Systemaufrufs entstanden und "mappt" einen Teil der "/etc/passwd"-Datei in den Adressraum des Prozesses. In dem Bereich befinden sich also Zeichen aus der passwd Datei (also der gemappte Teil der passwd-Datei).

Aufg. 5:

- (a) wahr
- (b) falsch
- (c) wahr
- (d) falsch
- (e) wahr
- (f) falsch
- (g) falsch
- (h) wahr
- (i) falsch
- (j) falsch

Aufg. 6:

- (a) Bei einem FAT-Dateisystem sind die Verwaltungsdaten der Datei zusammen mit dem 8.3 Namen in dem Verzeichniseintrag für die Datei untergebracht. Der Dateiname befindet sich also an der gleichen Stelle, im Verzeichniseintrag.
- (b) Ein FAT-Eintrag steht für einen Block des FAT-Dateisystems. Der Eintrag sagt, ob dieser Block frei ist, der letzte Block einer Datei ist oder er weist auf den nächsten Block der Datei, zu der er gehört (Verlinkung der Blöcke einer Datei).

Aufg. 7:

Eine Zeile der ls-Ausgabe zeigt zunächst einmal eine Zeichenmaske, die den Typ der Datei (erstes Zeichen), sowie die Zugriffsrechte für den Dateieigentümer, die Eigentümergruppe und andere Nutzer angeben. Dabei gibt es jeweils drei Rechte (rwx), die jeder Gruppe zugeordnet werden. Die nächste Zahl ist die Anzahl der Hardlinks auf die Datei. Dann folgt der Loginname des Besitzers und der Gruppenname. Anschließend sieht man die Anzahl der Bytes in der Datei. Das Datum gibt das Datum der letzten Modifikation der Datei an. Am Schluß folgt der Dateiname. Außer dem Dateinamen befinden sich alle diese Informationen in der Inode der Datei, die durch die Zeile beschrieben wird. Der Dateiname selbst stammt aus dem zugehörigen Verzeichniseintrag.

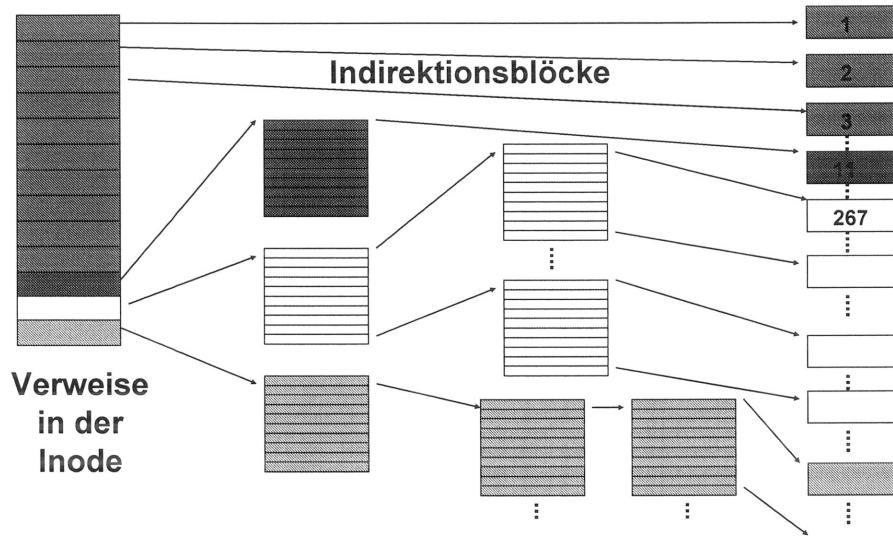
Aufg. 8:

- (a) Ein symbolischer Link ist eine Datei vom Typ "symbolischer Link", deren Inhalt ein absoluter oder relativer Pfadname ist.
- (b) Trifft der Kernel bei der Auflösung eines Pfadnamens auf einen symbolischen Link, so wird der Inhalt der Linkdatei ausgelesen und dann die Datei gesucht, auf die dieser Inhalt als Pfadname weist. Ist der Pfadname ungültig, wird eine entsprechende Fehlermeldung ("Keine Datei mit diesem Pfad") ausgegeben.

Aufg. 9:

Ein Programmierer muss eine Datei zunächst über einen open() Aufruf öffnen. Dann kann er auf die geöffnete Datei über die Systemaufrufe read() bzw. write() lesend bzw. schreibend zugreifen, wobei ein interner Lese-/Schreibzeiger der geöffneten Datei immer entsprechend der gelesenen bzw. geschriebenen Zahl von Bytes weitergesetzt wird. Sukzessive read() oder write() Aufrufe durchlaufen die Datei daher sequentiell. Möchte man ab einer bestimmten Position lesen oder schreiben, so kann man die Lese/Schreibposition mit dem lseek() Systemaufruf auf eine bestimmte Position setzen. Ein anschließendes Lesen bzw. Schreiben erfolgt dann ab dieser Position.

Aufg. 10:



Bei einer UNIX-Datei weisen 10 Referenzen in der Inode der Datei direkt auf die ersten 10 Inhaltsblöcke der Datei. Die elfte Referenz in der Inode weist dann auf einen Indirektionsblock, der selbst wiederum Referenzen auf die nächsten x Datenblöcke enthält. Die zwölfe Referenz in der Inode weist auf einen Indirektionsblock, der selbst x Referenzen auf Indirektionsblöcke enthält, die wiederum x Referenzen auf Datenblöcke enthalten (zweifache Indirektion). Schließlich gibt es noch eine 13-te Referenz, die dreifach indirekt ist.