Universität der Bundeswehr München

Sicherheit moderner Betriebssysteme

HT 2021

Übungen zur Rechenzeitvergabe/Scheduling und Speicherverwaltung

H. Görl

Universität der Bundeswehr
ETTI – Institut 2

28. Oktober 2021

Aufgabe 1: Speicherverwaltung und Kernel

1. Was versteht man unter Seitenflattern (Trashing) und wann tritt dies auf?
2. Beschreiben Sie kurz die Seitenersetzungsstrategie „Working-Set-Verfahren“ und geben Sie kurz an, was jeweils zur dessen Realisierung benötigt wird (Flags, Rechenzeit, Speicher, etc.). Was ist der Working-Set eines Betriebssystems?
3. Was versteht man unter der Beladys Anomalie und wann tritt dieser Effekt auf?
4. Warum benötigt man die Systemdienste `copy_from_user` und `copy_to_user` und was würde passieren, wenn diese Dienste nicht verwendet würden?
5. Beschreiben Sie eine Möglichkeit, wie ein Linux Kernel zur Laufzeit verändert bzw. erweitert werden kann. Skizzieren Sie den Vorgang des Erweiterns (`insmod`) und erläutern Sie, was im Kernelmodul auf jeden Fall vorhanden sein muss, damit das Modul geladen und entladen werden kann.
6. Beschreiben Sie bitte stichpunktartig, wie Kernelmodule aus einem C-Quelltext übersetzt werden. Was ist dazu nötig, welche Dateiendungen werden benutzt und was für eine Datei entsteht dann am Ende? Wie werden Module in ein laufendes System eingebunden, wie kann man deren Vorhandensein feststellen und wie werden diese wieder entfernt?

Aufgabe 2: Virtuelle Speicherverwaltung

1. Welche Aufgabe hat eine MMU? Wie ist eine MMU organisiert, wie sehen die Seitenkacheltabellen aus, welche Einträge sind mindestens vorhanden?
2. Nennen Sie vier Vorteile, die man durch den Einsatz von virtueller Speicherverwaltung gewinnt. Welchen (einen) Vorteil hätte man dagegen beim direkten Einsatz von realem Speicher ohne die Zwischenübersetzung der virtuellen Adressen?
3. Was ist eine inverse Seitentabelle und wieso wird diese benötigt?
4. Wieviel Speicher benötigt ein 2-stufiges Speicherverwaltungssystem maximal zur Verwaltung, wenn die virtuellen und physikalischen Adressen 32 Bit groß sind, die Pagetable (Seitentabelle) der ersten Ebene 1024 Einträge hält und eine Seite 4096 Bytes groß ist? Nur in den Tabellen der zweiten Stufe werden insgesamt genau 40 Bits Informationen abgelegt, keine weiteren (alles, auch z.B. das Valid-Bit, ist in den 40 Bits bereits enthalten).

Aufgabe 2: Rechenzeitvergabe – Teil 1

Als Scheduling-Modell sei ein System mit 5 Prozessen $P_1, \ldots, P_5$ und den Rechenzeiten bis der jeweilige Prozess beendet ist mit $b = (4, 2, 1, 3, 4)$ gegeben.

1. Die Ankunftszeiten sind $a = (0, 1, 2, 3, 4)$. Vor dem Zeitpunkt $t = 0$ besitzt das System keine Prozesse. Berechnen Sie die *mittlere Verweilzeit* $\overline{V}$ und die *mittlere Wartezeit* $\overline{W}$ für ein Round Robin-Verfahren mit einem Quantum von 2 Zeiteinheiten.

   Hinweis zur Berechnung:

   Gibt es n Prozesse, so gilt für die **mittlere Verweilzeit**:

   $$\overline{V}(S, a, b) = \frac{1}{n} \sum_{i=1}^{n} v_i$$

   wobei v_i die Verweilzeit des Prozesses i darstellt.

   Für die **mittlere Wartezeit** gilt:

   $$\overline{W}(S, a, b) = \frac{1}{n} \sum_{i=1}^{n} w_i$$

   analog stellt w_i die Wartezeit des Prozesses i dar.

   Bezeichnet $\overline{B} = \frac{1}{n} \sum_{i=1}^{n} b_i$ die mittlere Bedien- bzw. Rechenzeit, so gilt folgender Zusammenhang:

   $$\overline{V} = \overline{W} + \overline{B}$$

   Nutzen Sie zur Lösung das ausgeteilte Formblatt mit Skalierung.

Aufgabe 2: Rechenzeitvergabe – Teil 2

1. Wie kann man grundsätzlich abschätzen, ob ein Prozesssystem mit harten Echtzeitanforderungen planbar sein kann? Beschreiben und benennen Sie bitte die entsprechende Formel.
2. Welche Aussagen kann man grundsätzlich über das EDF-Verfahren machen? Wann ist ein Echtzeitsystem garantiert mittels RMS planbar?
3. Prüfen Sie bitte die Planbarkeit eines Echtzeitsystem mittels EDF und RMS, für das gilt, dass die Prozesswechselzeiten und sonstiger Overhead keine Zeit beansprucht. Die Startzeitpunkte für die Prozesse P1 und P2 sind bei $t = 0$. Die Ausführungszeiten sind $e1 = 15$ und $e2 = 25$. die Deadline-Zeiten sind $d1 = 20$, $d2 = 40$ und die Periode beträgt $p1 = 25$ und $p2 = 40$.

Aufgabe 3: Scheduling in BSD-Unix – Teil 1

Eine wesentliche Aufgabe der Prozessorverwaltung besteht darin, zu entscheiden, welcher der um den bzw. die Prozessor(en) konkurrierenden Prozesse zu einem Zeitpunkt an den bzw. die Prozessor(en) gebunden wird bzw. werden. Es werden Prozessorverwaltungsstrategien benötigt, die darin bestehen, dass an die Prozesse **Prioritäten** vergeben werden. Der Prozessor wird dann jeweils dem Prozess mit der höchsten Priorität zugeteilt. Die Prioritätenvergabe kann dabei statisch oder dynamisch sein. Im ersten Fall hat jeder Prozess für die Dauer seiner Existenz eine feste Priorität; im zweiten Fall können sich die Prioritäten der Prozesse dynamisch verändern, d.h. sie werden in gewissen Zeitabständen neu berechnet.
Bei einer **Zeitscheibenstrategie** werden die Prozesse an den Prozessor jeweils für ein festgelegtes Zeitquantum (in 4.3 BSD-Unix beträgt z.B. die Zeitscheibe 100 ms) gebunden und spätestens nach dem Ablauf dieser Zeitspanne wird den Prozessen der Prozessor wieder entzogen.
Zeitscheibenstrategien und Prioritätenvergabe können zu effizienten Verwaltungsstrategien kombiniert werden.

In dieser Aufgabe wird das Scheduling in dem Betriebssystem **4.3 BSD-Unix** genauer betrachtet. Bei dem Unix-Scheduling handelt sich um eine Zeitscheibenstrategie mit dynamischer Prioritätenvergabe. Unix vergibt für seine Prozesse Prioritäten von 0 - 127 (0 ist die höchste Priorität), die in 32 Warteschlangen verwaltet werden. Ein Prozess mit Priorität *PRIO* wird in die Schlange $PRIO/4$ eingeordnet. Alle Prozesse einer Prioritätsklasse befinden sich in einer Warteschlange, die nach einer Round-Robin Strategie abgearbeitet wird. Zunächst wird allen Prozessen der höchsten Priorität die CPU zugeteilt bis die Warteschlange leer ist. Dann kommen die Prozesse mit der nächstniedrigeren Priorität zum Zuge.

Aufgabe 3: Scheduling in BSD-Unix – Teil 2

Die Prioritäten werden fortlaufend neu berechnet (multilevel-feedback-queue). Die Prioritäten der Prozesse, die in einem gewissen Zeitabschnitt viel Rechenzeit verbraucht haben, werden erniedrigt; Prozesse, die lange gewartet haben, erhalten eine höhere Priorität („short-term-scheduling“). Die Prioritäten werden folgendermaßen berechnet:

$$u\_prio = USER\_PRIO + p\_cpu/4 + 2 * p\_nice \tag{1}$$

wobei *p_cpu* die Prozessornutzung des rechnenden Prozesses ist und alle 10 ms um 1 inkrementiert wird. *p_nice* ist ein vom Benutzer bestimmter Gewichtungsfaktor (-20 $\leq$ *p_nice* $\leq$ 20) und USER_PRIO ist die Priorität, die dem Prozess beim Start zugeteilt worden ist. *p_cpu* wird jede Sekunde angepasst durch:

$$p\_cpu = \big((2 * load)/(2 * load + 1)\big) * p\_cpu + p\_nice \tag{2}$$

wobei *load* eine Abschätzung der CPU-Auslastung ist. Die Anpassung (2) sorgt dafür, dass bisher verbrauchte Rechenzeit nach einer gewissen Zeit nicht mehr ins Gewicht fällt.

Aufgabe 3: Scheduling in BSD-Unix – Teil 3

1. In dieser Teilaufgabe sollen Sie sich das Unix-Scheduling an einem kleinen Beispiel verdeutlichen. Gegeben seien dazu folgende Prozesse mit ihrer Bedienzeit und Anfangspriorität:

Prozess	Bedienzeit	Priorität
1	10	3
2	1	1
3	2	3
4	1	4
5	5	2

   Zeichnen Sie die Abarbeitungsfolge der Prozesse, die sich bei Anwendung der oben erläuterten Strategie des „short-term-schedulings“ ergibt, geeignet auf.
   Zur Vereinfachung nehmen Sie an, dass ein Zeitquantum den Wert 1 hat, *p_cpu* (des rechnenden Prozesses!) in jedem Zeitquantum um 8 erhöht wird, *p_nice* den Wert 0 hat und *load* gleich 1 ist. Die Priorität aller Prozesse soll nach jedem vierten Quantum neu berechnet werden.

Mit den angegebenen Werten ergeben sich für die Berechnung von *u_prio* und *p_cpu* folgende Formeln:

$$u\_prio = USER\_PRIO + p\_cpu/4 \tag{3}$$

$$p\_cpu = \frac{2}{3} * p\_cpu \tag{4}$$

Aufgabe 3: Scheduling in BSD-Unix – Lösungsskizze Teil 1

Die Abarbeitungsfolge der Prozesse nach der Strategie des „short-term-schedulings“ kann an den folgenden Diagrammen abgelesen werden:

	1			2			3			4			5		
t	B-Zeit	prio	p_{cpu}	B-Zeit	prio	p_{cpu}	B-Zeit	prio	p_{cpu}	B-Zeit	prio	p_{cpu}	B-Zeit	prio	p_{cpu}
0	10	3	0	1	1	0	2	3	0	1	4	0	5	2	0
1				0	1	8									
2													4	2	8
3													3	2	16
4													2	2	24
	10	3	0	-	-	-	2	3	0	1	4	0	2	6	16
5	9	3	8												
6							1	3	8						
7	8	3	16												
8							0	3	16						
	8	6	11	-	-	-	-	-	-	1	4	0	2	5	11
9										0	4	8			
10													1	5	19
11													0	5	27
12	7	6	19												
	7	6	13	-	-	-	-	-	-	-	-	-	-	-	-
13	6	6	21												
14	5	6	29												
15	4	6	37												
16	3	6	45												
	3	10	30	-	-	-	-	-	-	-	-	-	-	-	-
17	2	10	38												
18	1	10	46												
19	0	10	54												