Systemprogrammierung
2. Hausaufgabe

Gruppe 1
Marc Radau
Ramin Khorsandi
Mustafa Karaca
Diego Nils Romero Koehler

2.3

Alle Folienangaben beziehen sich auf Vorlesungsblock 2.

a) Erklären Sie den Nutzen von Prozessen für das Betriebssystem. Gehen Sie dabei auf die Aussage „Prozesse sind instanziierte Programme“ ein.

(0,3 Punkte)

Ein Prozess ist ein Objekt, das sequentielle Aktivitäten in einem System repräsentiert. (Folie 2)

Dieser enthält Ressourcen, die Verarbeitungsvorschrift (das Programm selbst) und einen (ggf. mehrere) Aktivitätsträger (Threads), die das Programm zur Ausführung bringen. (Folie 2)

Ein Programm kann mehrmals gleichzeitig ausgeführt werden. Jede weitere Ausführung erzeugt einen weiteren Prozess. Die Programmvorschrift ist dabei die selbe, aber die Daten der verschiedenen Prozesse sind verschieden. (Folie 3 und 4)

Für die Verwaltung von Systemressourcen sind Prozesse wichtig. In der Regel sollen mehrere Prozesse logisch simultan ausgeführt werden. Um dies zu erreichen kann zwischen den Prozessen gewechselt werden. (Folie 12)

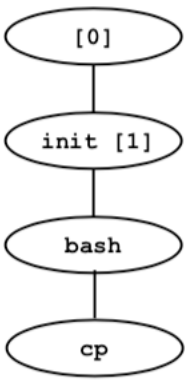
b) Beschreiben Sie die Unix-Prozesshierarchie? (0,2 Punkte)

Beim booten wird ‚manuell‘ ein Prozess 0 (inkl. PCB) erzeugt. Dieser forkt sich selbst und ruft *execl(/sbin/init)* auf und ist Prozess 1. (Folie 52)

Dieser Prozess 1 forkt sich selbst und ruft Daemon- (Hintergrund-)prozesse und *getty* auf, welches die Konsole konfiguriert. (Folie 53)

Dann wird das Login-Programm durch einen *exec* -Aufruf von *getty* gestartet, welches bei erlogreichem Login über *exec bash* oder eine ähnliche Shell aufruft. (Folie 54/55)

Alle weiteren Aufrufe des Benutzers werden durch die Shell (in diesem Fall *bash*) über *fork* und *exec* aufgerufen. Diese haben ggf. weitere Kindprozesse, die über diese beiden Methoden erzeugt werden. (Folie 56)



(Folie 56)

In dieser Grafik nicht abgebildet sind die von *init* geforkten Daemons. (vgl. Folie 53)

c) Nennen Sie zwei Ereignisse, die eine Prozessumschaltung zur Folge haben und erklären Sie, wie dieses Umschalten realisiert wird. Gehen Sie dabei auch auf den Begriff „Process Control Block“ (PCB) ein. (0,2 Punkte)

Die Prozessumschaltung kann durch mehrere Ereignisse ausgelöst werden.

Automatische Umschaltung passiert nach ablaufen der dem Prozess zugewiesenen Rechenzeit. Ein Intervallzeitgeber läuft mit einer gewissen Zeitscheibe oft zw. 1-10ms. Nach Ablauf wird der Prozess überprüft und der Prozess gewechselt. (Folie 17)

Implizite Prozessumschaltung passiert, wenn der Prozess einen blockierenden Systemaufruf tätigt. Das ist der Fall bei einem Aufruf auf Speicherzugriff, auf den der Prozess warten muss, bevor er weiterrechnen kann. (Folie 17)

Der Umschaltmechanismus speichert die Registerinhalte des laufenden Prozesses in seinen PCB und aktualisiert seinen Prozesszustand von *laufend* auf *blockiert*, *bereit*, ... (Folie 18)

Ein anderer Prozess wird dann geladen. Erst wird dessen Prozesszustand auf *laufend* gestzt. Dann wird der virtuelle Adressraum gemäß Konfiguration im PCB dieses Prozesses umgeschaltet und die Registerinhalte des Prozesses werden aus dessen PCB in die CPU-Register geladen. (Folie 18)

Der Prozesskontrollblock (PCB, Process control block) ist eine Datenstruktur, die den Prozess gegenüber dem Betriebssystem repräsentiert. Sie speichert die Prozessidentifikation (PID), den Benutzer, der der den Proess gestartet hat, die Registerwerte des Prozessed, wenn dieser nicht läuft, den Prozesszustand selbst, Informationen über zugeteilte Betriebsmittel, Verweise auf Eltern- und Kindprozesse und den zugeteilten Prozessor in Mehrprozessorsystemen. (Folie 15)

d) Was unterscheidet einen User Level Thread von einem Prozess? (0,3 Punkte)

Ein Prozess hat einen oder mehrere Aktivitätsträger, die das Programm im Prozess zur Ausführung bringen. (siehe a), Folie 2).

Ein Thread ist nur der Kontrollfluss (Ausführungsablauf..) des Prozesses, während der Prozess selbst die benötigten Ressourcen (Virt. Adressraum, Quelltext und Daten und andere Betriebsmittel) kontrolliert. (Folie 25/26)

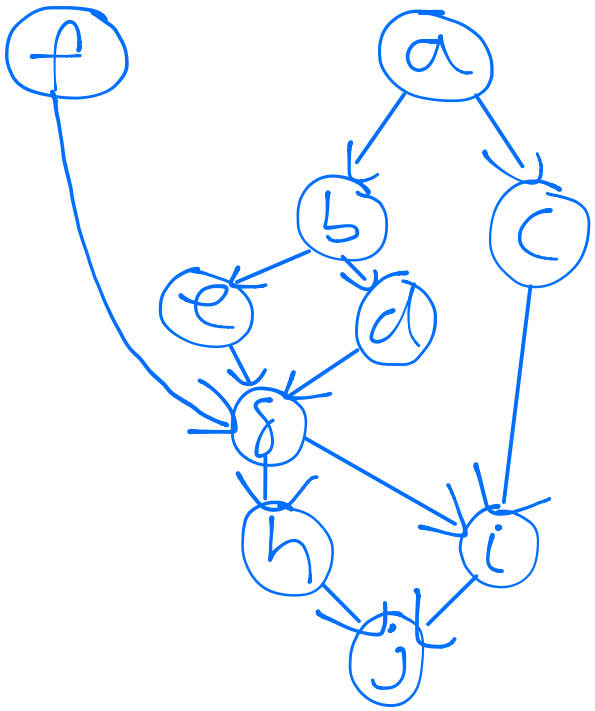
Threads können vom Betriebssystem (Kernel Level Threads) realisiert werden oder von der Anwendung (User Level Thread). (Folie 31)

Mehrere User Level Threads laufen unter einem dem Betriebssystem bekannten Kernel Level Thread der Anwendung. Die UL-Threads können dann ohne Systemaufruf von der Anwendung schnell selbst gewechselt und verwaltet werden. (Folie 33)

2.4

a) Wenn "a = jobA()" die Zeile 5 des C-Programms ist, sind die unabhängige Zeilen:

- Zeilen 5 und 10
- Zeilen 6 und 7 und 10
- Zeilen 7 und 8 und 9 und 10
- Zeilen 12 und 13



b)

```
fork f
a
fork c
b
fork e
d
join f
join e
g
fork h
join c
i
join h
j
```