Tutor: Friedrich
Tutorium: Dienstag 08-10 Uhr
Raum FH 314

Gruppenmitglieder:
Julius Basler
Fabian Philipp Berner
Jannik Leander Hyun-Ho Novak

Aufgabe 2.3

a)

Prozesse bieten dem Betriebssystem allgemein die Dienste des Prozessmanagements und der Prozessinteraktion. Dabei stellt ein Prozess sequentielle Aktivitäten in einem System dar, somit sind diese durch verschiedene Ressourcen und Verarbeitungsvorschiften definiert. Dabei können Prozesse als instanziierte Programme gesehen werden, da ein einziges mehrfach geöffnetes Programm seine Programmdaten (hier gemeint: Anweisungen) zwischen mehreren Prozessen teilen kann ohne diese mehrfach im Speicher zu benötigen. Des Weiteren wird somit aufgrund der durch das Betriebssystem verwalteten möglichen Prozesszustände die Effizienz des Systems gesteigert. (1)(2)

b)

(Erklärung Fork:
Ein Prozess kann eine Kopie von sich selber mittels fork Erzeugen. Dieser besitzt abseits der PID gleiche Eigenschaften, wobei der Erzeugende Prozess Parent, und der neue Prozess Child genannt werden. Dabei hat ein Child Prozess genau einen Parent und ein Parent Prozess beliebig viele Child Prozesse. (3))

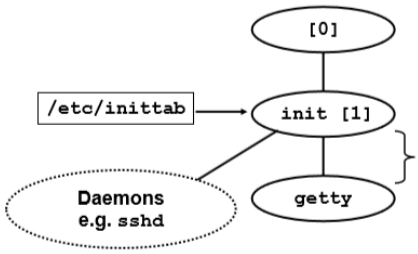
Entsteht durch die Erzeugung weiterer Prozesse durch die jeweiligen Child Prozesse.

-Bei Start wird PC mit Adresse des Bootstrapperprogrammes geladen.

-Bootblock, Kernelbinärprogramm werden geladen und Kontrolle an Kernel übergeben

-PCB für Prozess 0 wird manuell erzeugt

-Prozess 0 erzeugt mit fork() einen neuen Prozess welcher mittels execl() Befehl zu init[1] wird



-Init erzeugt nun mittels fork verschiedene Daemons, so auch getty(für die Konsole)

-Getty konfiguriert Konsole und wird zu login-Programm

-login überprüft Benutzername und Passwort und beendet sich bei ungültiger Eingabe (dann forkt init ein neues getty), bei gültiger Eingabe wird login zur Shell des Benutzers

Quelle: 4

Quellen:

1) Kao/Nordholz: Systemprogrammierung SS18; Vorlesungsfolien 2, S. 2.-5., 8.

2) Nordholz: Systemprogrammierung SS18 Vorlesung 12.05.2018

3) Kao/Nordholz: Systemprogrammierung SS18; Vorlesungsfolien 2; S. 41.

4) Kao/Nordholz: Systemprogrammierung SS18; Vorlesungsfolien 2; S. 50 ff.

c)

1. Festplattenzugriff (Block)
Wenn ein Prozess Daten von einer Festplatte lesen möchte wird dafür ein *Syscall* benötigt. Um die Berechtigungen etc. zu überprüfen wird er im Eintrag *Prozesszustand* vom Betriebssystem auf *Blockiert* gesetzt und erst bei abgeschlossenem Vorgang wieder auf *Bereit*. Dabei wird vom Betriebssystem der PCB des Prozesses u.A. darauf überprüft ob der Prozess auf die dementsprechenden Daten zugreifen darf. Des Weiteren werden die aktuellen Register Werte im PCB abgelegt, damit diese nicht verloren gehen. (1)

2. Prozess könnte wieder laufen (Ready)
Ein Prozess im *Blockiert* Zustand wird wieder ausgeführt da bspw. Die Behandlung eines Blockierenden *Syscalls* abgeschlossen wurde. Nun wird im PCB der *Prozesszustand* wieder auf *Bereit* gesetzt. (1)

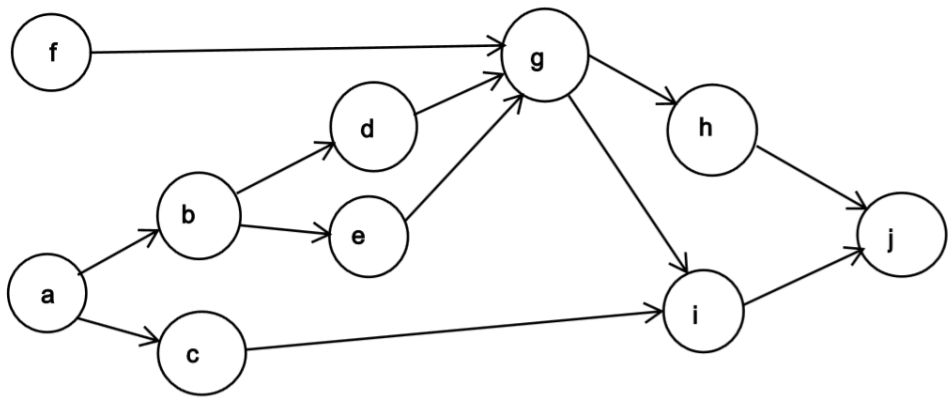
Der PCB stellt dabei eine Menge an Informationen dar, in welchen z.B. die eindeutige Prozessnummer (PID), eine Identifikation des Besitzers, aktuelle Registerzustände und vieles weiteres gesichert werden (somit also alles was den Prozess auszeichnet). Wobei das Betriebssystem diese Informationen bspw. für die Berechtigungsverwaltung und Weiterem nutzt.

d)

Ein *Prozess* kann mehrere *Threads* haben wobei diese dem Kernel auch als verschiedene (*Kernel Level) Threads* vorliegen, wobei mehrere *User Level Threads* immer mindestens einen *Kernel Level Thread* zugeordnet sind. Nun kann zwischen den *UL Threads* direkt aus einer Laufzeitumgebung heraus gewechselt werden, wobei das Betriebssystem nichts mitbekommt da es nur die *Kernel Level Threads* sieht. (2)

Aufgabe 2.4

a)



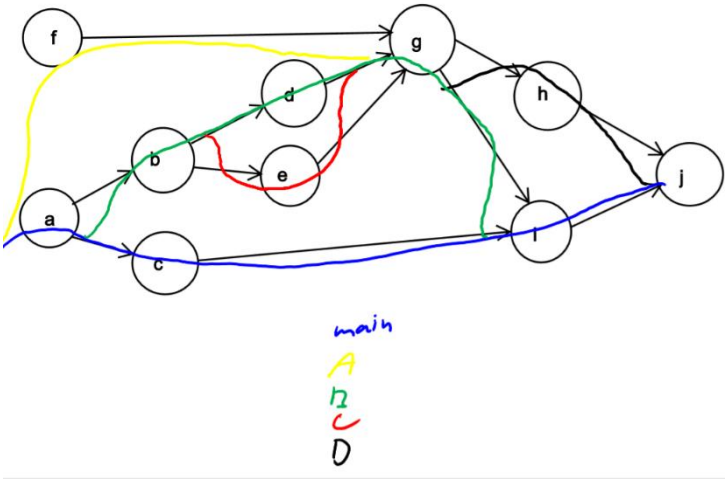
Aufgabe 2 Theorie: Basler, Berner, Novak

Quellen:

1) Kao/Nordholz: Systemprogrammierung SS18; Vorlesungsfolien 2, S. 9, 15.-18.

2) Kao/Nordholz: Systemprogrammierung SS18; Vorlesungsfolien 2; S. 32.-34.

b)




```
fork A
jobA();
fork B
jobC(a);
join B
jobI(g,c);
join D
jobJ(h,i);
A:
        jobF();
B:
        jobB(a);
        fork C
        jobD(b);
        join A
        join C
        jobG(d,e,f);
        fork D
C:
        jobE(b);
D:
        jobH(g);
```