Disclaimer

Die Übungen die hier gezeigt werden stammen aus der Vorlesung *Betriebssysteme*! Für die Richtigkeit der Lösungen wird keine Gewähr gegeben.

Übung 1

Aufgabe 1

Wie würden Sie heute, zu Beginn des Kurses, den Begriff "Betriebssystem"beschreiben? Sehen Sie Analogien zwischen den Aufgaben und Funktionen von Betriebssystemen und denen gesellschaftlicher oder wirtschaftlicher Einrichtungen?

Begriff "Betriebssysteme"beschreiben: Verknüpfung zwischen Hard- und Software.

- Ermöglicht Kommunikation
- Software: Browser, Office, Bibiliotheken,...
- Betriebssystem: Betriebssystemdienste, Resourcenmanagement, Schnittstellen, Bibiliotheken,...
- Hardware: CPU, GPU, E/A, Speicher, Netzwerk,...

eine Abstraktion von Hardware-Ressourcen, Sammlung von programmen, Vorteil: Komplexität verborgen vor dem Nutzer

Aufgabe 2

Wie sind diejenigen Betriebssysteme in Erscheinung getreten, mit denen Sie bisher schon gearbeitet haben? Welche Aufgaben haben sie gelöst? Welche Probleme haben sie gezeigt? verschiedene

Betriebssysteme:

- Universalsysteme (Windows, Linux, Mac, Android, iOS): Benutzerfreundlich, keine Spezialisierung (GUI gehört nicht dazu)
- eingebettete Systeme (Microkontroller): resourcensparend, belastbarkeit
- Echtzeitsysteme (motorsteuerung): deadlines!
- chipkarten/sicherheitssysteme: nicht wiederbeschreibbar, meist genau 1 Aufgabe

Aufgabe 3

In der Vorlesung haben Sie ein breites Spektrum an Einsatzgebieten für Betriebssysteme kennen gelernt, das verdeutlichen sollte, welche sehr unterschiedlichen Anforderungen heute an Betriebssysteme gestellt werden. Kennen Sie über diese Szenarien hinaus Beispiele für Einsatzgebiete, in denen folgende eine zentrale Rolle spielen? Was wären die jeweiligen Konsequenzen, wenn ein Betriebssystem dabei diese Eigenschaften nicht besitzen würde?

Echtzeitfähigkeit einhalten von Fristen ist Hauptziel, harte oder weiche EZS
Robustheit betrieb in widrigen Umgebungen (Umwelteinflüsse, Anwenderfehler)
Sicherheit Security: schutz gegen Angriffe von außen (Verschlüsselung etc); Safety: Schutz gegen Ängriffe von innen"(Speichersicherheit etc)
Korrektheit (gegenüber einer Spezifikation) Testen (keine vollständige Sicherheit), Verifikation (mathematischer Beweis)
Performanz Geschwindigkeit von Anwenderprogrammen, Effiziente Algorithmen für BS Dienste
Sparsamkeit BS Größe, energie- und ressourcenschonend
Skalierbarkeit Lastskalierbarkeit (Bsp Online shops), Ressourcenskalierbarkeit

Aufgabe 4

Wenn Sie auf einem der heute üblichen Computer Bürosoftware, Internetprogramme (Webbrowser, E-Mail-Clients etc.) und Audio/Video-Applikationen nutzen möchten und die Wahl eines Betriebssystems hätten: Welche der oben genannten und welche weiteren Eigenschaften würden Sie von diesem Betriebssystem erwarten? Anforderungen an Office-PC-BS (universalsystem): Interaktivität, Performance, Robustheit, korrektheit, sicherheit, sparsamkeit, bequemlichkeit, hohe abstraktionsebene

Aufgabe 5

Warum könnte es problematisch sein, ein und dasselbe Betriebssystem auf Großrechnern (Mainframes) und gleichzeitig auf eingebetteten Systemen einzusetzen? gleiches Betriebssystem für Großrechner und eingebetteten Systemen? Portierbarkeitsprobleme

- Aufgaben sehr unterschiedlich; Batch-Verarbeitung, Interaktiv
- Ressourcen Verwaltung/Größen
- Platformspezifische Einheiten (Befehlssätze, RISC vs CISC)
- Funktionsumfang

Aufgabe 6

Welche Vorteile hat es aus Anwendersicht, Betriebssysteme als Virtualisierung von Maschinen zu verstehen? Vorteile für Anwender die Betriebssysteme als Virtualisierung zu sehen

- Bequemlichkeit
- Beherschbarkeit + Isolation

Übung 2

Repetitorium

- **Welcher Zusammenhang besteht zwischen den Konzepten Nebenläufigkeit und Parallelität? Wann können Aktivitäten auf einem System nur pseudoparallel ausgeführt werden?** Nebenläufig: kausal unabhängig; Parallel: zeitlich überlappend; Aufgaben können nur parallel ausgeführt werden wenn diese Nebenläufig sind
- **Wozu dienen Prozessmodelle? Aus welchen Bausteinen setzt sich ein Prozessmodell zusammen? Welche finden sich typischerweise in Prozessdeskriptoren wieder?**
- **Aus welchem Grund wurde das Thread-Konzept entwickelt? Welche zwei eigentlich unabhängigen Konzepte sind im Modell des ursprünglichen (Schwergewichts-)Prozesses vereint?** Prozesse von einem Ressoircenmanagement und Parallelität; Threads laufen im Ressourcenkontext über Prozesse
- **Wozu dienen Prozess- bzw. Threadzustände? Welche elementaren Zustände finden sich in jedem Prozessmodell?** Elementare Zustände: bereit, aktiv, blockiert, (frisch, beendet); ermöglichen Scheduling
- **Warum benötigt jeder Thread einen eigenen Stack?**
- **Worin besteht der Unterschied zwischen Kernel- und User-Level-Threads? Welche Vor- und Nachteile besitzt die jeweilige Form? Wo befinden sich die PCB- und TCB-Datenstrukturen?**

Aufgabe 1: Prozesserzeugung in Linux-Systemen

In Betriebssystemen der Unix/Linux-Familie werden neue Prozesse durch den fork-Systemaufruf erzeugt. Dabei entsteht sukzessive eine Abstammungshierarchie, in der ein Prozess, der ein fork() (erfolgreich) ausführt, zum Elternprozess ("parent") des von ihm erzeugten Kind-Prozesses ("child") wird. Die bei Unix/Linux-Systemen benutzte Technik funktioniert wie folgt: Durch fork wird eine nahezu exakte Kopie des Elternprozesses zum Zeitpunkt des fork()-Aufrufs erzeugt, bei der der neue Kindprozess eine Vielzahl der Eigenschaften des Elternprozesses erbt. Falls der Kindprozess ein anderes als das vom Elternprozess vererbte Programm ausführen soll, kann das Kind unmittelbar nach fork einen Systemaufruf der exec[ute]-Familie verwenden, der das durch den aufrufenden Prozess ausgeführte Programm austauscht. a) Informieren Sie sich über $fork$

und exec∗ und finden Sie Antworten auf die folgenden Fragen. Wie unterscheiden sich Eltern- und Kindprozess unmittelbar nach dem fork()-Aufruf? Woran können sich Eltern- und Kindprozess unmittelbar nach einem fork()-Aufruf selbst erkennen ("Wer bin ich?")? Finden Sie mindestens 3 Möglichkeiten. Welche unterschiedlichen Werte kann der Funktionsaufruf fork() zurückgeben, und was bedeuten sie?

- Unterscheidung: getrennte Speicherbereiche, unterschiedliche PIDs, Programmierung gleich
- Selbsterkkennung: *getpid*(), *getppid*(), *system*()-calls, Rückgabewert von $fork()$
- Rückgabewert von $fork()$:
  - PID des Kindes von Parent
  - 0 im Kindprozess

– -1 Fehler (errno gesetzt)

b) Demonstrieren Sie mit dem einfachen C-Programm p1 (in der Anlage), dass nach der Ausführung von fork() tatsächlich zwei Prozesse existieren.

c) Führen Sie Programm p2 aus. Dieses enthält einen execl-Systemaufruf, mit dem ein Programm p4 ausgeführt werden soll. Das Programm p4 muss dabei ein (mit dem C-Compiler) übersetztes, ausführbares Programm im gleichen Verzeichnis sein. Sie können dazu das vorgegebene Programm p4 verwenden, das lediglich einen Ausdruck erzeugt.

d) Wie viele Prozesse werden durch das Programm p3 erzeugt? Warum? Was passiert, wenn execl() nicht erfolgreich ausgeführt werden kann, weil z. B. das Programm p4 nicht gefunden wird? Führen Sie zur Kontrolle p3 aus, während das Programm p4 einmal existiert und ein weiteres Mal, während dieses nicht existiert.

Aufgabe 2: Prozessdeskriptoren und Prozesszustände

Beschäftigen Sie sich mit dem Shell-Kommando ps. Es dient dazu, bestimmte Informationen aus den Prozessdeskriptoren ausgewählter Prozesse auszugeben. Die ausgewählte Prozessmenge und der Umfang der wiedergegebenen Informationen kann dabei durch entsprechende Parameter beeinflusst werden. So bedeutet z. B. $ps-el$, dass eine Liste mit vielen Parametern (l, "long") für alle Prozesse (e) erzeugt wird. Aus Gründen der Übersichtlichkeit kann auch $ps-al$ verwendet werden. a) Welche Prozesszustände sind in Linux-Betriebssystemen definiert, und wie erkennt

man diese an den Ausgaben von ps?

b) Starten Sie das Programm p5, in dem der durch $fork()$ erzeugte Kindprozess in einer Endlosschleife läuft (Zweck?). Welche Prozesszustände haben Eltern- und Kindprozess? Beobachten Sie, welche Komponenten der Prozessdeskriptoren sich in Abhängigkeit von der Zeit ändern und interpretieren Sie dies.

c) Wenn ein Prozess auf ein sogenanntes Ereignis warten muss, beispielsweise eine Eingabe, dann ändert sich dessen Zustand. Starten Sie Programm p6, welches mittels $getchar()$ auf eine Eingabe vom Standardeingabegerät (ohne weitere Maßnahmen ist dies die Tastatur) wartet, und untersuchen Sie die Zustandsinformationen des wartenden Prozesses.

d) Mit dem Systemaufruf sleep() kann sich ein Prozess selbst ßchlafen legen". Starten Sie p7 und untersuchen Sie, welchen Zustand der hierdurch erzeugte Prozess nach dem Aufruf von sleep() einnimmt.

Aufgabe 3: Dateiformate ausführbarer Programme

Kompilierte Programme liegen immer in einem genau definierten Format vor. Ziel dieser Aufgabe ist es, Erkenntnisse darüber zu gewinnen, wie ein Betriebssystem aus einem kompilierten Programm einen Prozess erzeugt. Ein in Linux-Betriebssystemen verbreitetes Binärformat ist ELF (Executable and Link Format), welches Gegenstand dieser Aufgabe ist. a) Im Mittelpunkt

Ihrer Recherchen über ELF sollte stehen, welche Informationen das Betriebssystem zur Erzeugung eines Prozesses benötigt und wo und wie diese in ELF zu finden sind. Berücksichtigen Sie ebenfalls die Metainformationen, die sich im ELF-Header befinden. Finden Sie Antworten auf die folgenden Fragen.

- *Wie findet man (bzw. das Betriebssystem) die erste auszuführende Instruktion innerhalb des Text-Segments?*
- *Auf welche Weise bekommen bereits im Quellprogramm (z. B. C-Programm) initialisierte Variablen ihre Anfangswerte vor dem Start der Ausführung eines Programmes?*
- *Woran erkennt man, um welchen Typ einer in ELF dargestellten Datei es sich handelt? Für welche Dateitypen ist ELF prinzipiell vorgesehen?*
- *Unterscheiden sich ELF-Dateien für 32-Bit- und 64-Bit-Prozessorarchitekturen?*
- *Woran ist das gegebenenfalls erkennbar?*
- *Welchen Zweck haben die so genannten Sektionen (sections) bzw. die program headers?*
- *Welche Bedeutung hat eine Symboltabelle als Teil einer in ELF dargestellten Datei?*

b) Untersuchen Sie experimentell Binärdateien hinsichtlich ihrer ELF-Metainformationen. Hierzu können Sie die Werkzeuge readelf 9 und objdump 10 verwenden, um zu ermitteln, welche konkreten Informationen eine Datei im ELF-Format enthalten kann. Als Beispiele sollen mindestens die folgenden ELF-Binärdateien dienen:

- *das ls-Utility, zu finden im Verzeichnis /bin,*
- *die Executable zum Programm p1.c (siehe Anlage zu Aufgabe 1),*
- *eine dynamisch ladbare Bibliothek aus dem Verzeichnis /lib oder /usr/lib.*

Übung 3

Repetitorium

Durch welche Ereignisse wechselt ein Thread vom Zustand aktiv in den Zustand blockiert? Durch we

Welche Auswirkung hat im Round-Robin-Schedulingalgorithmus die Veränderung der Größe der Zei

Round-Robin-Scheduler verwalten normalerweise eine oder mehrere Listen von Prozessen, wobei jed

Welche Form des Schedulings – preemptiv oder nicht preemptiv – führt aus grundsätzlichen Überleg

Aufgabe 1: EDF

Die Scheduling-Strategie Earliest Deadline First (EDF) kommt dann zum Einsatz, wenn die Abarbeitung eines Prozesses bis zu einem definierten Zeitpunkt (Frist, Deadline) erfolgen muss. Wir wollen uns in dieser Aufgabe auf statische Deadlines beschränken, auch wenn diese in bestimmten, realen Anwendungen während der Abarbeitung von Prozessen (dynamisch) neu bestimmt werden können. Die Strategie ist nun wie folgt: Ein Prozess wird einem anderen vorgezogen, falls er eine frühere Deadline hat. Ein neu ankommender – und aufgrund einer zeitigeren Deadline höher priorisierter – Prozess verdrängt einen bereits rechnenden, aber niedriger priorisierten Prozess. Das Verfahren ist also präemptiv. a) Implementieren Sie EDF. Sie finden dazu eine

Musterklasse im Projekt unter Simulation → Source Packages → frm.pssav.sim → PreemptiveEDFScheduler.java.

b) Vergleichen Sie nun den einfachen Round-Robin-Algorithmus (eine Warteschlange) mit EDF. Überlegen Sie sich zwei Szenarien: Im ersten sollen beide Algorithmen die gesetzten Fristen einhalten. Im zweiten sollte ersichtlich werden, in welchen Fällen Round Robin gegenüber EDF versagt. Benutzen Sie ausreichend viele Prozesse, so dass die Simulation lange genug läuft, damit ihre Kommilitonen genügend Zeit haben, sich der dargestellten Probleme bewusst zu werden.

Aufgabe 2: Round Robin mit Prioritäten

Die Scheduling-Strategie Round Robin soll ein möglichst faires Scheduling mehrerer Prozesse ermöglichen. In der Vorlesung haben Sie die zusätzliche Möglichkeit kennengelernt, Round Robin mit einem Prioritätenschema zu kombinieren. Wir wollen uns in dieser Aufgabe auf statische Prioritäten beschränken, auch wenn diese in bestimmten, realen Anwendungen während der Abarbeitung von Prozessen (dynamisch) neu bestimmt werden können. a) Implementieren

Sie Round Robin mit Prioritäten. Sie finden dazu bereits eine Implementierung der Strategie

ohne Berücksichtigung von Prioritäten unter Simulation → Source Packages → frm.pssav.sim → RoundRobinScheduler.java, die Sie entsprechend anpassen müssen.

b) Von welchen Faktoren ist die Länge einer Zeitscheibe in der Praxis abhängig? Von welchen die Priorität? Welcher Unterschied ergibt sich daraus für das Setzen dieser beiden Parameter?

Demonstrieren Sie die Auswirkungen unterschiedlich langer Zeitscheiben sowie unterschiedlicher Prioritäten anhand zweier Szenarien: Eines mit (genügend vielen) sehr kurzen Prozessen, ein anderes mit sehr viel länger rechnenden. Diskutieren Sie, welche realen Einflussfaktoren – die in PSSAV nicht berücksichtigt werden können – hier in der Praxis eine Rolle spielen müssen.

Aufgabe 3: Linux-Scheduling

Moderne Universal-Betriebssysteme müssen heute mit Prozessen und Threads sehr unterschiedlichen Charakters umgehen können. Einerseits könnten z. B. Echtzeitprozesse zur Audio/Video-Verarbeitung ablaufen, andererseits gibt es auch zeitunabhängige aber rechenzeitintensive Prozesse wie beispielsweise das in der Vorlesung gezeigte Ray-Tracing-Programm. a) Recherchieren Sie

für das Betriebssystem Linux die für das Scheduling von Prozessen verantwortlichen funktionalen Komponenten (Scheduling-Subsystem) und ermitteln Sie, welche Schedulingstrategien Linux unterstützt.

Damit das Betriebssystem sinnvoll mit diesen unterschiedlichen Prozessen umgehen kann, müssen Prozesse selbst ihre Lastmerkmale dem Betriebssystem mitteilen (neben der Beobachtung und Klassifizierung der Prozesse durch das Betriebssystem selbst). Eine sehr einfache Möglichkeit, das Scheduling von Prozessen zu beeinflussen, besteht in der Vergabe unterschiedlicher (Basis-)Prioritäten, auf deren Grundlage die dynamische Änderung durch das Betriebssystem vorgenommen wird. Für jeden regulären Prozess steht hierzu der Systemaufruf nice zur Verfügung; ein Nutzer kann die Priorität seiner Prozesse mithilfe der Ausführung der Kommandos nice oder renice verschlechtern (der Name deshalb, weil er damit "nettßu anderen Prozessen ist). b)

Starten Sie zwei gleichartige rechenzeitintensive Prozesse (ggf. hierfür ein einfaches Programm schreiben), die lange genug rechnen. Demonstrieren und erläutern Sie deren Verhalten, wenn einerseits beide die gleiche Priorität haben und andererseits ein Prozess seine Priorität verringert hat. Überlegen Sie sich geeignete Demonstrationsmöglichkeiten.

Hinweise: Falls Sie nicht selbst geeignete Ideen haben: Beispielsweise lassen sich lange laufende Prozesse durch Hochzählen einer Variablen vom Typ long int (lange ganzzahlige Variable) erzeugen, oder durch Starten mehrerer Instanzen einer ausreichend anspruchsvollen Anwendung Ihrer Wahl (anspruchsvoll für den Hauptprozessor, nicht nur für die Grafikhardware).
Falls Sie an einem System mit einem Mehrkernprozessor arbeiten, könnten die zu zeigenden

Effekte unsichtbar bleiben, wenn z. B. jeder Prozess auf einem eigenen Prozessorkern läuft. Beschäftigen Sie sich hierzu z. B. mit dem Systemaufruf $sched_setaffinity()$.

Übung 4

Repetitorium

Welche prinzipiellen Probleme entstehen, wenn parallel ablaufende Threads auf einen gemeinsamen S

In der Vorlesung haben wir binäre Semaphore kennen gelernt, die innerhalb eines kritischen Abschni

Wie viele Semaphore braucht man mindestens zur Lösung des allgemeinen Erzeuger/Verbraucher-Pr

Wir haben Bedingungsvariable im Kontext Hoare'scher Monitore kennen gelernt. Ist ein derartiges S

Aufgabe 1: Das Problem des schlafenden Barbiers

In einem Friseurladen gibt es einen Friseur, einen Frisierstuhl (an dem der Friseur arbeitet) und n Stühle für wartende Kunden. Entwickeln Sie einen Algorithmus, der unter den nachfolgenden Annahmen Friseur und Kunden so synchronisiert, dass jeder wartende Kunde (irgendwann) bedient wird.

- Der Friseur und alle Kunden agieren parallel.
- Falls keine Kunden da sind, geht der Friseur schlafen.
- Wenn ein Kunde kommt, während der Friseur schläft, weckt der Kunde den Friseur und setzt sich in den Frisierstuhl (und wird bedient).
- Wenn ein Kunde eintrifft, während der Friseur arbeitet und ein freier Kundenstuhl vorhanden ist, setzt sich der Kunde und wartet.
- Trifft ein Kunde ein, während der Friseur arbeitet und alle Kundenstühle belegt sind, verlässt der Kunde den Friseurladen sofort wieder.
- Wenn der Friseur mit dem Bedienen eines Kunden fertig ist, verlässt dieser Kunde den Friseurladen und einer der wartenden Kunden (falls vorhanden) belegt den Frisierstuhl und wird bedient (sonst gilt Bedingung 2).

Aufgabe 2: Das Achterbahnproblem

Das Achterbahnproblem nach J.S. Herman wird durch das folgende Szenario beschrieben. Eine Anzahl n von "Vergnügungssüchtigen"(im Folgenden Passagiere genannt) versucht, möglichst oft eine Fahrt mit einem der m zur Verfügung stehenden Achterbahnwagen zu unternehmen. Dabei gelten allerdings die folgenden Bedingungen.

- *Ein Wagen darf nur losfahren, wenn er voll besetzt ist. (Dabei gilt: Jeder Wagen fasst c Passagiere, wobei die Gesamtzahl der beteiligten Passagiere mehr als nur einen Wagen füllt, d. h. $c < n$.)*
- *Wenn ein Wagen vollständig besetzt ist, fährt er los.*
- *Wenn der Wagen nach Abschluss der Fahrt wieder anhält, steigen alle Passagiere aus und bemühen sich erneut, in einem "neuen"Wagen eine weitere Fahrt zu unternehmen. (Unter entsprechenden Umständen kann es natürlich auch wieder der gleiche Wagen sein.)*
- *Wagen dürfen sich nicht überholen (was ja beim gegebenen Achterbahnproblem auch technisch nicht möglich ist), d. h. die Reihenfolge der Wagen bleibt immer gleich.*

Passagiere und Wagen sollen durch Aktivitäten simuliert werden, die synchronisiert werden müssen. Für eine der möglichen Lösungen könnten die folgenden Hinweise hilfreich sein: Bei der Ankunft eines Wagens sollte dieser eine Prozedur Einsteigen() aufrufen, danach sollten c Passagiere ihrerseits eine Prozedur $In_Wagen_Einsteigen()$ aufrufen. Nach beendeter Fahrt sollte ein anhaltender Wagen eine Prozedur Aussteigen() aufrufen, und die sich im Wagen befindenden c Passagiere sollten daraufhin eine Prozedur $Wagen_Verlassen()$ aufrufen.

Aufgabe 3: Der Kaffeeautomat

Ein Kaffeeautomat, seine Kunden und ein Lieferant, der den Automaten regelmäßig mit Kaffee und Kaffeebechern auffüllt, sollen sich mittels Semaphoren synchronisieren. Synchronisieren Sie das Verhalten dieser Aktivitäten so, dass folgendes Verhalten realisiert wird.

- *Der Automat kann entweder einen Kunden bedienen oder durch den Lieferanten nachgefüllt werden. Beide Vorgänge sind nicht gleichzeitig möglich!*
- *Ein Kunde muss nach Aufforderung durch den Automaten eine 1-Euro-Münze als Bezahlung einwerfen, erst danach bekommt er seinen Kaffee. (Um eine ungeeignete Betriebsweise auszuschließen, soll angenommen werden, dass sich nur Kunden anmelden, die eine 1-Euro-Münze parat haben – und nach Aufforderung natürlich auch einwerfen!)*
- *Der Lieferant bekommt durch den Automaten mitgeteilt, dass dieser für den Auffüllvorgang bereit ist.*
- *Ein einmal gestarteter Vorgang (Bedienen bzw. Auffüllen) kann nicht mehr unterbrochen werden. Eine neue Anmeldung (durch den nächsten Kunden oder den Lieferanten) wird erst nach Abschluss dieses Vorgangs akzeptiert.*
- *Der nächste Kunde und der Lieferant können sich jeweils unabhängig voneinander beim Automaten anmelden. Die Reihenfolge der Bedienung hängt dann davon ab, in welcher Reihenfolge die Anmeldungen erfolgen.*
- *Lieferant und Kunde bekommen den Abschluss des jeweiligen Vorgangs durch den Automaten mitgeteilt.*
- *Falls der Kaffeevorrat verbraucht ist oder keine Becher mehr vorhanden sind, versetzt sich der Automat selbst in einen Wartezustand und wartet bis er durch den Lieferanten wieder befüllt ist.*

Übung 5

Diskussionsfragen

Frage 1: Speicherbasierte vs. nachrichtenbasierte Interprozesskommunikation *Stellen Sie sich vor, Ihre Übungsgruppe müsste ein Videoschnittsystem entwickeln, welches aus Robustheitsgründen die Berechnung komplexer Effekte, die En- und Dekodierung verschiedener Formate sowie die Steuerung der gesamten Applikation in jeweils unterschiedlichen Prozessen implementieren soll. Diese Prozesse müssen natürlich miteinander kommunizieren und dabei neben Kontrollinformationen auch die zu verarbeitenden Video- und Audiodaten austauschen. Prinzipiell stehen dafür nachrichtenbasierte und speicherbasierte Kommunikationsmechanismen zur Verfügung. Für welche der existierenden Mechanismen würden sie sich entscheiden, um einerseits Kontrollinformationen und andererseits Mediendatenströme auszutauschen? Begründen Sie Ihre Antwort.*
Hinweis: Klären Sie zuerst, was die prinzipiellen Vor- und Nachteile dieser beiden Kommunikationsvarianten sind. Betrachten Sie anschließend die Kommunikationsmuster und Anforderungen der beiden Klassen (Kontroll- und Multimediadaten), bevor Sie eine Empfehlung geben.

Frage 2: Synchronisation durch Semaphore *Bei asynchroner nachrichtenbasierter Kommunikation kommen stets Warteschlangen zum Einsatz, um unterschiedliche Geschwindigkeiten der Sender- und Empfängerprozesse auszugleichen. Der Zugriff auf diese Warteschlangen muss aus verschiedenen Gründen durch Synchronisationsmechanismen (z. B. Semaphore) geregelt werden. Was sind diese Gründe und weshalb sind insgesamt drei Semaphore pro Warteschlange notwendig?*

Frage 3: Synchronisationsvarianten bei nachrichtenbasierter Kommunikation *Welche Nachteile asynchroner Kommunikation treten beim Einsatz synchroner Varianten der Sende- und Empfangsoperationen nicht auf? Warum ist es trotzdem manchmal sinnvoll oder unumgänglich, die asynchronen Varianten einzusetzen? Nennen Sie mindestens drei Beispiele realer Applikationen, in denen asynchron kommuniziert wird.*

Frage 4: Management asynchroner Ereignisse *Welche Alternativen haben die Entwickler von Betriebssystemen, um mit asynchron auftretenden Ereignissen (Mausbewegungen, Einstecken von USB-Geräten etc.) umzugehen? Welche Technik erlaubt es auch einem Benutzerprozess, auf asynchrone Ereignisse zu reagieren, ohne direkten Hardwarezugriff zu haben?*

Aufgabe 1: Nachrichtenwarteschlangen (Message Queues)

a) Recherchieren Sie die Funktionsweise, Charakteristiken und Eigenschaften von Message Queues. Wie wird der Kontrollfluss der Prozesse dabei durch das Betriebssystem gesteuert (z. B. Synchronisation durch Blockierungen, durch die Ankunft von Daten usw.)?

b) In der Anlage zu dieser Übungsaufgabe (u4-a1-anlage) befinden sich ein Server- und ein Client-Programm, die beide Lücken enthalten. Vervollständigen und übersetzen Sie die Programme. Starten Sie anschließend zuerst den Server und dann den Client. Falls Sie die Lücken richtig ausgefüllt haben, muss das Client-Programm ein "Passwortän den Server senden und anschließend ein Geheimnis ausgeben, das es vom Server als Antwort erhalten hat.

Aufgabe 2: Gemeinsamer Speicher (Shared Memory)

a) Recherchieren Sie die Funktionsweise, Charakteristiken und Eigenschaften von Shared Memory. Wie wird der Kontrollfluss der Prozesse dabei durch das Betriebssystem gesteuert? (Wann und wodurch erfolgt eine Synchronisation?)

b) In der Anlage zu dieser Übungsaufgabe (u4-a2-anlage) befinden sich ein Server- und ein Client-Programm, die beide Lücken enthalten. Vervollständigen und übersetzen Sie die Programme. Starten Sie anschließend zuerst den Server und dann den Client. Falls Sie die Lücken richtig ausgefüllt haben, muss das Client-Programm ein "Passwortän den Server senden und anschließend ein Geheimnis ausgeben, das es vom Server als Antwort erhalten hat.
Hinweis: An den verwendeten Semaphoroperationen sind keine Änderungen notwendig.

Aufgabe 3: Benannte Pipes (Named Pipes, FIFOs)

a) Recherchieren Sie die Funktionsweise, Charakteristiken und Eigenschaften von Pipes und Named Pipes. Wie wird der Kontrollfluss der Prozesse dabei durch das Betriebssystem gesteuert? (Wann und wodurch erfolgt eine Synchronisation?)

b) In der Anlage zu dieser Übungsaufgabe (u4-a3-anlage) befinden sich ein Server- und ein Client-Programm, die beide Lücken enthalten. Vervollständigen und übersetzen Sie die Programme. Starten Sie anschließend zuerst den Server und dann den Client. Falls Sie die Lücken richtig ausgefüllt haben, muss das Client-Programm ein "Passwortän den Server senden und anschließend ein Geheimnis ausgeben, das es vom Server als Antwort erhalten hat.

Übung 6

Repetitorium

Wodurch sind die Mehrkosten eines Systemaufrufs im Vergleich zu einem regulären Prozeduraufruf

Für die Behandlung asynchroner Unterbrechungen (Interrupts) gibt es mehrere Modelle, u.a. auch d

Vor welchem fehlerhaften Verhalten von Anwendungsprozessen schützen private virtuelle Adressräu

Warum ist es nur schwer möglich, schon während der Übersetzung eines Programms dafür zu sorgen

Warum ist eine optimale Pagingstrategie im Allgemeinen nicht erreichbar? Unter welcher speziellen

Wie viele Seitentabellen gibt es in einem anlaufenden Betriebssystem mit virtueller Speicherverwaltu

Was ist die Arbeitsmenge (Working Set) eines Prozesses? Wozu dient sie? Zu welchen Zeitpunkten b

Welche Aufgaben hat eine MMU? Wozu dient ein TLB (Translation Look-aside Buffer)? Wann wird

Seitentabellen können je nach Adressraumgröße sehr groß werden. Mit diesem Problem gehen mehrs

Wie groß ist die Seitentabelle? Wie groß ist die Seitenrahmen- bzw. Kachelgröße?

- Um welchen Faktor reduziert sich die Größe der Seitentabelle, die ständig im Hauptspeicher zu halten ist, bei einem zweistufigen Seitentabellenverfahren gegenüber einer einstufigen Seitentabelle?
- Welche Größe haben die Seitentabellen bei einstufigen bzw. zweistufigen Verfahren bei einer Eintragsbreite von jeweils 4 Byte?

Bei einer Seitengröße von 8 KiByte und einem virtuellen Adressraum der Größe 2^{64}: Wie groß wäre

Aufgabe 1: Systemaufrufe

Dienstleistungen des Betriebssystems (z. B. Erzeugen von Prozessen, Threads oder Dateien) werden unter Zwischenschaltung von Stellvertreterprozeduren aufgerufen (z. B. für Programme in der Sprache C finden diese sich bei Linux-Systemen meist in der C-Standardbibliothek libc), die dann über einen aufwändigeren als den Prozedurmechanismus den tatsächlichen Sprung ins Betriebssystem implementieren. a) Warum kann man aus einem Anwendungsprogramm nicht direkt

eine im Betriebssystem implementierte Prozedur aufrufen? Erläutern Sie die gängige alternative Verfahrensweise. Welche prinzipiellen Bestandteile enthalten die genannten Stellvertreterprozeduren?

b) In der Anlage zur Aufgabe finden Sie das Programm syscall.c; dort ist beispielhaft ein Systemaufruf manuell implementiert. Warum enthält das Unterprogramm Assemblerbefehle? Erweitern Sie nach obigem Muster Ihren ëigenen"exit()-Systemaufruf my_exit(), indem Sie das begonnene Fragment ergänzen. Demonstrieren Sie dessen korrekte Funktionsweise.
Hinweis: Mit dem Systemaufruf wait() kann man den Terminierungsstatus von Kindprozessen

überprüfen. Um das Programm zu kompilieren, auszuführen und seinen Rückgabewert anzuzeigen, können Sie alternativ das beigefügte Shell-Skript run.sh benutzen.

c) Vor der Einführung des Maschinenbefehls syscall in modernen 64-Bit-Architekturen musste zum Auslösen eines Systemaufrufs durch ein Anwendungsprogramm ein spezieller Interrupt (trap) ausgelöst werden. Recherchieren Sie, welche Vorteile die Verwendung von syscall gegenüber dieser konventionellen Verfahrensweise hat. Auch heute noch wird aus Gründen der Abwärtskompatibilität und zur Unterstützung spezieller Prozessorarchitekturen der trap-Mechanismus durch den Linux-Kernel unterstützt. Schaffen Sie es, Ihre my_exit()-Implementierung mittels Ersetzen der syscall Instruktion durch einen Trap-Interrupt (int$0x80) entsprechend zu portieren? Beachten Sie, dass dabei Systemaufruf-Nummern für 32-Bit-Hardwarearchitekturen und andere Register zu verwenden sind.

Tipp zur gesamten Aufgabenstellung: Unter Linux können Sie mit strace < Programm > u.a. die Systemaufrufe eines Programms verfolgen.

Aufgabe 2: Ereignismanagement mit Linux-Signalen

In Linux-Systemen kann mittels des Signalmechanismus eine Behandlung bestimmter Ereignisse auf Prozessebene stattfinden (aus der Vorlesung als Ïnterruptbehandlung auf Prozessebene"bekannt).
Dieses Prinzip soll nun mit mittels sogenannter Dämon-Prozesse (daemons) veranschaulicht werden. Dabei handelt es sich um Hintergrundprozesse, die i. d. R. im Verlauf des Boot-Vorgangs gestartet werden und keine direkte Benutzerinteraktion vorsehen. So kümmert sich beispielsweise der Line Printer Daemon (lpd) oder der Common Unix Printing System Daemon (cupsd) darum, Druckaufträge nebenläufig zu anderen Benutzerprozessen abzuwickeln. Eine typische Verwendung von Signalen ist beispielsweise, einem Dämon die Änderung seiner Konfigurationsdatei(en) zur Laufzeit mitzuteilen, woraufhin diese erneut eingelesen werden sollen. Hierfür hat sich die Benutzung des Signals SIGHUP etabliert.

a) Recherchieren Sie, welche Signale der für Unix-Systeme etablierte POSIX-Standard vorsieht und wie diese einem Prozess zugestellt werden. Demonstrieren Sie in der Übung, wie man auf der Kommandozeile einem beliebigen Prozess ein Signal (z. B. SIGHUP oder SIGKILL) senden kann. Informationen hierzu finden Sie wie immer in den Linux-Manpages oder im Handbuch zur C-Standardbibliothek libc.

In der Anlage zu dieser Aufgabe stellen wir Ihnen den Quellcode eines kleinen Dämon-Prozesses zur Verfügung. Zur besseren Demonstration haben wir darauf verzichtet, ihn als Hintergrundprozess zu initialisieren; er läuft daher wie ein Nutzerprozess und kann so seine Ausgaben auf der Kommandozeile sichtbar machen.

b) Starten Sie den mitgelieferten Dämon und demonstrieren Sie, wie dieser auf verschiedene Signale reagiert. Erklären Sie das Verhalten mithilfe Ihrer Recherche aus Teilaufgabe a).

c) Erweitern Sie den Dämon nun um die Fähigkeit, seine Konfigurationsdateien neu zu laden, wann immer er das Signal SIGHUP empfängt. Sie können dazu als Reaktion auf das Signal die bereits vorhandene Funktion $load_config()$ aufrufen.

d) Schaffen Sie es, durch Reaktion auf die Signale SIGTERM und SIGKILL ein explizites Terminieren des Prozesses durch den Benutzer zu verhindern? Demonstrieren und erklären Sie Ihre Ergebnisse.

Aufgabe 3: Virtuelle Speicherverwaltung von Linux-Systemen

Stellen Sie das virtuelle Speichermanagement von Linux-Systemen vor. Gehen Sie dabei insbesondere auf die Struktur der Seitentabellen und die Seitengröße ein. Virtuelle Speicherverwaltung bietet auch eine elegante Möglichkeit zur dynamischen Speicherverwaltung, d. h. je nach Bedarf den von einem Prozess belegten Speicherplatz zu vergrößern bzw. wieder zu verkleinern. Die Motivation zur Integration dieser Technik in den Linux-Kernel war die Einführung dynamischer Objekte in Programmiersprachen, die z. B. mittels new() während des Programmablaufs bei Bedarf neue Variablen, insbesondere große Arrays, erzeugen und auch wieder löschen können. Zum Ansprechen der entsprechenden Implementierungen in Linux, werden auf Nutzerebene die Funktionen malloc() ("memory allocation") und free() (Wiederfreigeben von Speicher) bereitgestellt. Beide Funktionen sind über Eintrittspunkte in die C-Standard-Bibliothek implementiert. Freier Speicher wird dabei vom Heap besorgt.

a) Erklären Sie diesen Begriff im Linux-Kontext. Stellen Sie dann die beiden obigen Funktionen vor und demonstrieren Sie ihre Benutzung.

Im Linux-Kern benutzen malloc() und $free()$ den Systemaufruf $brk()$ ("break"), der ebenso freien Speicher requiriert, aber für die Verwendung als Programmierschnittstelle nicht empfohlen wird. Dieser ändert den so genannten Programm-"break". b) Was bedeutet dies? Wie kann man

dessen aktuellen Wert sowie dessen Maximalwert feststellen?

c) Wodurch unterscheiden sich die Funktionen malloc()/$free()$ und $brk()$?