Betriebssysteme

Wintersemester 2019/20

Inhaltsverzeichnis

1 Checkpoint 1

1.2 What is an Operating System?

Keine klare Definition vorhanden. Mögliche Antworten: Es ist eine Schicht zwischen Hard- und Software, welche Programme und Resourcen verwaltet.

1.3 Why did Operating Systems emerge?

Anforderungen an Computer haben sich mit der Zeit verändert. Jobverwaltung und bessere Auslastung wurde immer relevanter. Betriebsysteme wurden entsprechend zur Verwaltung verschiedener Aufgaben auf einem System.

1.4 Which of the following statements is false?

a: Operating System evolution required Hardware changes

b: Hardware evolution required Operating System changes

c: Hardware and OS evolved independently

d: There was strong influence in both directions

Lösung: Antwort c ist falsch.

1.5 Describe three critical early inventions in Operating Systems

mögliche Lösungen sind:

1. Time Sharing
2. (Pipelines)
3. Job Controll (mehrere Programme direkt hintereinander ausführen)
4. Multiprogrammierung (andere Programme laufen, während ein Programm auf IO Events wartet)

ToDo: auf entsprechenden Folien verweisen

1.6 What is UNIX?

Eine historische Betriebssystemfamilie, die auch heute noch immer aktuell ist.

1.7 What is POSIX?

Eine stark an UNIX angelehnter Quellcodestandard. Beschreibt wie ein System entwickelt werden muss, damit es POSIX Kompatibel ist. (Sich wie ein POSIX System verhält.) Es ist selbst kein Betriebssystem.

1.8 Discuss wether POSIX is still relevant today.

Keine falsche Antwort, da Diskussion. Entscheidend ist die Begründung.

1.9 Describe the relation between the terms Process, Program, Thread and File

Ein Prozess ist ein laufendes Programm, ein Programm ist eine ausführbare Datei und ein Thread ist ein Ausführungsstrang eines Prozesses. Ein Prozess kann mehrere Threads haben, welche alle im gleichen Kontext laufen. (Sie teilen sich Speicher etc.)

1.10 What is a shell?

Eine (Eingabe- und Ausgabe-) Umgebung in der eine Eingabe als Befehl interpretiert und entsprechend ausgeführt wird. (Kommandointerpreter)

1.11 Describe briefly what happens when you type 'ls' into a UNIX shell and press enter

Wenn Enter gedrückt wird, wird ein Interrupt erzeugt, welcher vom Betriebssystem abgefangen wird und dann als Eingabe an die Shell gesendet wird. Anschließend wird der Befehl 'ls' von der Shell als Programmaufruf interpretiert woraufhin die Shell sich selbst forkt und im Kind das Programm 'ls' im PATH sucht und ggf. startet. 'ls' nutzt die gleiche Ausgabe wie die Shell und zeigt so alle Dateien. Der Elternprozess wartet darauf, dass der Kindprozess terminiert.

1.11.1 What would have happened if you had typed 'cd' instead?

'cd' wird als aufruf eines Builtin interpretiert, weshalb die Shell in das HOME Verzeichnis des Nutzers wechselt. Es geschieht kein Fork-Exec.

1.12 In Pseudo-Code, write a program that executes 'ls' in a child process and waits in the parent process for the termination of the child process.

```
#include <...>

int main(){
    pid_t pid = fork();
    if(pid == -1){
        printf("error in fork\n");
        return -1;
    }else if(pid == 0){
        // CHILD
        char *const args = {"ls", NULL};
        int ret = execvp("ls", args);
        // ret == -1
        printf("error in exec; errno: %d\n", errno);
        return -1;
    }else{
        // PARENT
        wait(pid);
```

```
            return 0;
        }
    }
```

1.12.1 Which system calls would you use?

wait, fork, exec

1.13 Briefly describe five tasks of an Operating System

Resourcenverwaltung (Storage Managment (Haupt- und Plattenspeicher), Memory Managment, Scheduling (CPU Verwaltung, Prozessormanagment), Device Managment (Geräterverwaltung und Treiber, Interruptbehandlung)), Security- und Usermanagment (auch Prozessschutz voreinander)

1.14 Briefly describe three design goals of an Operating System

Portability, Maintability, Security, Performance, Responsive

1.15 Discuss the reasoning behind the seperation of User- and Kernel Mode. Identify advantages and disadvantages.

Schutz der Programme voneinander (Isolation), Benutzerverwaltung und Berechtigungssystem
Ein Nachteil: extra Performance durch ständige Wechsel über Syscalls

1.16 What is an Interrupt?

Eine Unterbrechung des aktuellen Programmablaufes (meist durch externe Geräte). Syncrone Interrupts entstehen durch Programmfluss (meist durch Exceptions) und Asyncrone Interrupts entstehen durch Hardwareereignisse.

1.17 What can cause an Interrupt?

Ein Interrupt kann zum Beispiel ausgelöst werden, wenn ein Prozess auf IO Zugriff warten muss.

1.18 Describe how the Operating System handles incoming Interrupts.

CPU hält die Ausführung des laufenden Prozesses an, die CPU sichert den Kontext des aktuellen Prozesses und behandelt anschließend den Interrupt entsprechend. (Meist ist der entsprechende Behandlungscode im Treiber) In der Interruptbehandlung wird dem Gerät auch mitgeteilt, dass es jetzt aufhören kann Interrupts zu senden. (Meist gleich als erstes) Anschließend wird das Programm fortgesetzt, indem sein Kontext wiederhergestellt wird. Während der Interruptbehandlung wird Interruptbehandlung weiterer möglicherweise eintretenden Interrupts deaktiviert. Entsprechend darf es während der Interruptbehandlung keine Programmierfehler geben. Interrupts können auch erst später behandelt werden, je nach Priorität.

1.19 What is the purpose of a System Call?

System Calls werden verwendet um von dem User Mode aktionen im Kernel Mode zu initialisieren.

1.20 Describe what happens when at runtime when a program uses the function 'getpid'.

Wir bekomme die PID (werden benötigt um Prozessen von einander zu entscheiden) des aktuellen Prozesses. Funktion ist in der User Mode Bibliothek beschrieben. Funktion wird aufgerufen und schreibt in ein Register die Syscall Nummer. Cpu wechselt in den Kernel Mode und die Routine für Systemaufrufe laden. Schaut die Syscall Nummer im Register nach und führt dann die entsprechende Funktion hier getpid() aus. Systemaufruf ist ein Synchroner Interrupt.
Funktion → User Mode Bibliothek → Syscall, Softwareinterrupt → Interraptionsroutine bestimmt Funktion an Hand der Nummer im Register → Funktion ausführen und Ergebniss nach Oben schleifen

1.21 What are Windows "Personalities"?

a: Independent operating modes of the CPU

b: Distinguished Engineers at the Microsoft Campus in Redmond

c: Seperate classes of applications, and their corresponding subsystems

d: Groups of Users in the System with different Privileges

Antwort C

1.22 Describe the anatomy of a Windows Subsystem.

Programme benutzen Subsystems an der Stelle von APIs. Die Subsysteme sind dokumentieren und greifen auf undokumentierte Windows System Service Calls zu.

1.23 Compare the three types of Subsystem Service Calls.

- vollständig im User Mode implementiert → Geometriefkt. PtInRect() IsRectEmpty()
- es werden ein oder mehrere Systemcalls für die Ausführung benötigt → ReadFile() WriteFile(); erstellt ind Subsystem Library
- benötigt die Umgebung des Subsystem Process → CreateFile(), durch IPC

1.24 Name three Operating System components usually located in User- and Kerner Mode (three each)

- User Mode
  - Commandointerpreter (Shell)
  - Subsysteme
  - Services, Dienste
- Kernel Mode
  - Grafikengine
  - Hardware Abstraction, Treiber, ISR
  - Kernel

– Memorymanagement

– ...

1.25 What is the role of a System Thread in Windows? Name two examples.

Thread ist ein Handlungsstrang eines Prozesses im User Mode.
Ein Systemthread ist eine nebenläufige Aktivität im Kernel Mode, welche eine spezielle Aufgabe erfüllt. Ein Systemthread wird durch das System nicht als Prozess abgebildet. Ein Systemthread hat priviligierten Zugriff auf das System.

Beispiele: Zero Page Thread (Nullt Speicher), eigene Threads von Treibern, Balance Set Manager (greift bei Speicherknappheit ein und verwaltet Prozessorspeicher), generell alle nebenläufigen Aufgaben im Kernel

1.26 What is a Servive / Daemon?

Hintergrundprozess im System, Kind von init, hat keine Ein- oder Ausgabe, vollführt Systemrelevante Aufgaben, üblicherweise ist er nicht-priviligiert

1.27 Compare the concepts "Microkernel" and "Monolithic Kernel".

Microkern: so viel Kernel Funktionalität wie möglich in den User Mode verlegen. Der Kernel enthält nur noch die wirklich relevanten Dinge. (Scheduling, Speicherverwaltung, Dispatch, Interprozesskommunikation) Meist aus Sicherheitsgründen. Entsprechend sind alle Treiber und die Kernel-Objektverwaltung im User Mode und werden isoliert. Zusammenfassung: kleiner Kern mit wenig Funktionen, vielen Diensten

Monolithic Kernel: mächtiger Kern mit vielen Funktionen im Kernel Mode, wenig Diensten

1.27.1 Which one would you use to describe Windows and which one for Linux?

Linux: Monolith Windows: (hybrider Kern), hauptsächlich Microkern

1.28 The Windows Kernel is Object Oriented. What is a Handle and what is its role?

Ein Handle ist ein Verweis auf ein existierendes Kernel Obejekt. Ein Handle gilt nur für jeweils einen Prozess. (Jeder Prozess hat eine eigene Handle Tabelle.) Ein Handle kann genutzt werden, um Systemfunktionen Kernel Objekten zu übergeben ohne das der Kernel diese Objekte "preisgeben" muss.

1.29 Can you share handles between separate Processes?

Nicht direkt. Aber ich kann in die Handle Tabelle eines anderen Prozesses eine weiteres Handle hinzufügen, welches auf das gleiche Kernel Objekt verweist. (Mit duplicateHandle oder so.)

1.30 Describe four types of Windows Kernel Objects.

File Object (einer geöffneten Datei), Port Object (kann genutzt werden um Nachrichten zwischen Prozessen zu verschicken), Thread Object, Process Object, Symbolic Link, Object directory (kann andere Objekte speichern um Hierarchie zu realisieren), Event Object, Semaphore Object, ... (siehe Unit 1, Vorlesung 3, Folie 30ff.)

1.31 What is the Basic Input/Output System (BIOS)?

Eine Software welches direkt auf der Hardware (der ROM (Read Only Memory)) liegt, welches das aktuell laufende Motherboard booten soll.

2 Checkpoint 2

2.2 What is Preemption?

In einem System in dem Prozesse in Zeitscheiben unterteilt werden, ist dies die faire Aufteilung dieser Zeitscheiben auf die Prozesse, durch das Unterbrechen des aktuellen Prozesses vom Betriebssystem. Jeder Prozess erhält ungefähr gleich viel Rechenzeit.

2.3 What new challanges did Preemption introduce when compared to cooperative multiprogramming?

Programme können an beliebigen Stellen unterbrochen werden. Dies wird zu einem Problem, falls eine Resource zwischen verschiedenen Programmen geteielt wird und beide Programme darauf zugreifen.

2.4 How is Preemption implemented in an operating system kernel?

Meist durch einen Timer Interrupt. (Eine Zeitscheibe endet, wenn die Clock ein Signal gibt.) Der Scheduler wählt ein neues Programm aus, welches Rechenzeit erhalten soll und der Dispatcher tauscht den aktiven Kontext aus, damit das neue Programm rechnen kann.

2.5 Compare Concurrency and Parallelism.

Concurrency: keine echte Gleichzeitigkeit, sondern eher ein konstantes abwechseln (immer nur aktiver Thread)
Parallelism: zwei Kerne, die echt Gleichzeitig rechnen

2.6 What is a Critical Section?

Ein Codesegment, in dem auf eine geteilte Resource zugegriffen wird.

2.7 What is the value of shared, and why?

In einer Nebenläufigen Ausführung von func_a und func_b:

```
int shared = 0;
void func_a(void)
{
    shared++;
}
```

```
void func_b(void)
{
    shared--;
}
```

Antwort: -1, 0 oder 1; je nachdem wie die Unterbrechung der Programme erfolgen, da das keine atomaren Anweisungen sind und wir uns somit temporäre Werte in Registern merken müssen, welche durch die Unterbrechung invalide werden.

2.8 Describe the three criteria that correct solutions of the Critical Section problem must fulfill.

- Progress: wenn mehrere Prozesse in die Critical Section eintreten wollen, dann darf die Entscheidung wer eintreten darf nicht beliebig lange hinausgezögert werden.

- gegenseitiger Ausschluss: wenn ein Prozess in der Critical Section ist, darf kein anderer Prozess in seiner Critical Section sein.
- Bounded Waiting: wenn ein Prozess in die Critical Section eintreten möchte, dann darf er nicht unendlich lange daran gehindert werden. (aber beliebig lange)

2.9 Why does this naive approach not solve the critical section problem? Outline a schedule.

```
shared int turn = 0;
do { // Code for T_i
     while (turn != i);
          critical section
     turn = j;
          remainder section
} while (1);
```

Problem: Progress wird verletzt: wenn ein Thread einen anderen "überholt", dann kann es passieren, dass dieser Thread am Eintreten der Critical Section gehindert wird, da der andere Thread noch in der remainder Section ist. Der erste Thread muss warten, bis der zweite seine Remainder Section und Critical Section durchlaufen hat, bis er selbst in die Critical Section eintreten darf.

Ein Beispiel: zwei Prozesse P0 und P1. turn ist auf 0 und P0 ist in der critical section und P1 in der remainder section. P0 verlässt nun seine critical section und turn wird dadurch auf 1 gesetzt. Wenn P0 nun vor P1 die remainder section verlässt, kann P0 nicht in die critical section eintreten, da turn immer noch auf 1 ist. P0 wird somit am Eintreten gehindert und Progress wird verletzt.

2.10 Name and describe one well known software algorithm that solves the critical section problem.

verschiedene Möglichkeiten:

Baker Algorithmus: Jeder Prozess zieht eine Nummer, wenn er in die critical Section eintreten will. Die gezogenen Nummer sind monoton steigend, aber nicht streng-monoton steigend. Es darf immer der Prozess eintreten, der die kleinste Nummer hat, oder bei Gleichstand zwischen zwei Prozessen der mit der kleinsten ID.

Peterson Algorithmus

Dekers Algorithmus **ToDo:** genauere Erklärungen

2.11 Discuss whether pure software algorithms are a good solutions to the critical section problem.

Nein, denn sie funktionieren nicht für echte Parallelität (Parallelism) und sie sind langsam mit vielen Threads.

2.12 To mitigate the problems of the software algorithms, hardware approaches are used instead. Which is not one of them?

a: test-and-set

b: outlook

c: exchange (compare and swap)

d: interrupt disabling

Antwort: d wird in Windows genutzt um die Critical Section vor Unterbrechungen abzusichern. a und c nutzen beide das Prinzip eine Variable atomar zu setzen und nur in die Critical Section einzutreten, wenn die Operation ein Erfolg war. Entsprechend ist c falsch.

2.13 What is a Semaphore?

Eine geteilte Integer Variable, deren Zugriff durch das Betriebssystem geschützt wird. (Ihre Operationen sind atomar.)

2.13.1 What operations are defined on Semaphores?

wait (Dekrementieren des Wertes, wenn die Semaphore > 0 ist), signal (Inkrementieren)

2.14 Identify the advantages and disadvantages of using Semaphores over native hardware approaches.

Vorteil: Semaphore ist Abstraktion, damit ist die zugrundeliegende Hardware / Betriebssystem irrelevant; die Semaphore kann mehrere Zustände haben; weniger Bussy Waiting als bei test-and-set
Nachteil: (Code) Overhead gegenüber test-and-set

2.15 What is a Deadlock?

Mehrere Prozesse warten auf eine Freigabe (von z.B. einer Semaphore), die durch einen jeweils anderen wartenden Prozess bereits blockiert ist. Entsprechend kann kein Prozess weiter arbeiten.

2.15.1 Describe how a Deadlock can be the result of careless use of Semaphores.

Beispiel: Zwei Prozesse versuchen zwei Semaphoren in unterschiedlicher Reihenfolge zu blockieren.

2.16 Given the following producer/consumer example program, how could you guard it with semaphores?

(Annahme: genau ein Producer und ein Consumer.)

```
void producer(void)
{
    int item;
    while(1) {
        produce_item(&item);
        if (count == N) suspend();
        insert_item(item);
        count = count + 1;
        if (count == 1) wake(consumer)
    }
}
```

```
void consumer(void)
{
    int item;
    while(1) {
        if (count == 0) suspend();
        remove_item(&item);
        count = count - 1;
        if (count == N-1) wake(producer
            );
        consume_item(item);
    }
}
```

Zwei Semaphoren einfügen: einen für den count und einen für N-count (freien Plätze). Der Producer wird dann die count Semaphore erhöhen und auf die free Semaphore warten, der Consumer wird genau das Gegenteil tun.

2.17 Describe the Role of the Dispatcher Objects.

Es sind Objekte (im Windows Kern) auf die man warten kann. Ein Dispatcher Objekt kann zwei verschiedene Zustände haben: signalisiert oder nicht signalisiert.

2.18 What does it mean for a Dispatcher Object to be Signaled, or Non-Signaled?

Bei nicht Signalisierten Objekten: wartende Threads werden blockiert.
Unsure: Bei signalisierten Objekten: ein wartender Thread kann weiterarbeiten.

2.18.1 Describe for one type of dispatcher object object what can cause it to change these states.

Bei Semaphoren: Status ist abhängig vom Wert der Semaphore.
Bei File Objekten: wir können mit einem wait auf das File Object explizit darauf warten, dass IO Operationen auf die Datei fertig werden.
Bei Prozess Objekten: wir können darauf warten, dass der Prozess terminiert.
Bei Timer Objekten: in periodischen Abständen signalisiert.

2.19 Name two functions of the Windows or UNIX API related to process synchronization and explain their purpose.

diverse Möglichkeiten: sem_wait + sem_signal (Semaphore Operationen), wait (warten auf Prozess), WaitForSingleObject (warten auf Dispatcher Objekt in Windows) pthread_join (warten auf Thread), ...

2.20 What is the IRQL?

Interrupt Request Level - beschreibt die Wichtigkeit des aktuell von der CPU behandelten Interrupts. Interrupts unter diesem Level werden später behandelt, Interrupts mit höherem Level sofort. (Ist somit ein Zustand der CPU.) Liegt zwischen 0 und 32.

2.21 Describe what tasks the operating system handles at various ranges of IRQLs.

0: normaler Zustand
1: (spezielle) APCs (Asyncronous Procedure Call)
2: Speicherverwaltung, Scheduling, ...
3+: diverse Geräte
in den oberen: PowerFail, Bluescreen

2.22 What is a Trap?

Der Mechanismus der bei einer einkommenden Unterbrechung den aktuellen Thread Zustand sichert. (Wir fangen den Thread ein.)

2.23 What kinds of event can cause a Trap?

Alle Interrupts, sysenter / syscall, Exceptions, Page Faults

2.24 What happens if during interrupt processing an interrupt of lower precedence arrives? What if the precedence is higher?

Wenn die Priorität kleiner ist, dann wird der einkommende Interrupt von der CPU ignoriert und werden behandelt, sobald die IRQL wieder sinkt.
Wenn die Priorität höher ist, wird der aktuelle Zustand mittels einer Trap gesichert und der einkommende Interrupt wird behandelt.

2.25 What are the roles of the Defered and Asynchronous Procedure Calls? (DPC and APC)

Beide werden verwendet, falls die Interrupt Service Routine lange dauert. Dann kann diese in eine DPC Objekt verpackt werden und später ausgeführt werden. (Wenn der IRQL wieder gesunken ist.) Damit werden andere Interrupts nicht unnötig blockiert uns eventuelle Verluste dieser Interrupts werden vermieden.
DPC sind dabei Prozessunabhängig, können also in jedem Thread ausgeführt werden, während APC zu einem Prozess zugehörig sind. (Entsprechend sind die Speicheradressen die vom Prozess und identisch zum Zeitpunkt des Interrupts.)

2.26 A Thread on Windows executes the Function ReadFile(). Describe the flow of activity from the application to the device and back.

Diese Frage muss in der Klausur nicht so ausführlich beantwortet werden.

Die ReadFile Funktion ruft die NTReadFile Funktion in Ntdll.dll auf, diese wird einen Systemaufruf zur Kernel Funktion NTReadFile ausführen. (Dieser Systemaufruf wird wiederum durch einen Softwareinterrupt realisiert, welche eine ISR aufruft.) Diese wird in der Handle Tabelle die entsprechende Datei zum Handle raussuchen. Anschließend wird der Treiber für das Gerät auf dem sich die Datei befindet geladen und ausgeführt. Der Interrupt ist nun fertig und es wird in den User Mode gewechselt. Der aktive Thread wird erst einmal warten. Sobald die Festplatte fertig ist mit dem Lesen der Datei, wird sie ein Interrupt schicken, welche einen DPC erzeugt, der die Daten aus der Platte in den Kernelspeicher kopieren wird. Anschließend erzeugt dieser DPC einen APC (im User Mode), welcher die Daten aus dem Kernelspeicher in den Prozessspeicher kopieren wird. Anschließend wird der APC den Prozess signalisieren, dass er weiterarbeiten kann, was dann wiederum die ReadFile Funktion beenden wird.

2.27 What is a pipe? Why is it needed?

Prozesse sind isoliert, müssen aber eventuell miteinander kommunizieren. Die Pipe ist ein Verbindungsstück zwischen zwei Prozessen.

2.28 What is the difference between a pipe and a socket?

Es sind beides Konstrukte, welche zu Prozessen gehören.
In der Pipe ist die Kommunikation nur in eine Richtung.
Ein Socket realisiert eine Server-Client Struktur. (Es gibt einen Server-Prozess welcher einen Socket öffnet und Client-Prozesse die sich mit diesem Socket verbinden können.) Mit einem Socket können wir in mehrere Richtungen kommunizieren.

2.29 Pipes and sockets are tied to the lifetime of a process. What other UNIX construct could you use that survives its creating process, and where is it persisted?

Named Pipes (FIFO): verhält sich ähnlich zu Dateien, ist aber als Puffer implementiert.

2.30 Write in Pseudocode (Windows or UNIX semantics) a program that launches two processes and connects them through a pipe.

Missing!

2.31 Name two functions of the Windows or UNIX API related to inter-process communication, and explain their purpose.

pipe: Erstellen einer Pipe, mkfifo: Erstellen einer Named Pipe, ...

3 Checkpoint 3

3.2 What is a Programm? What is a Process? What is a Thread?

Ein Programm ist eine ausführbare Datei.
Ein Prozess ist ein laufendes Programm.
Ein Thread ist ein Ausführungsstrang eines Programms.

3.3 What are the roles of the scheduler and the dispatcher?

Der Scheduler hat die Aufgabe den laufenden Prozessen ihre CPU Zeit zuzuweisen.
Der Dispatcher gibt die Kontrolle zum dem nächsten aktiven Thread(wiederherstellen des CPU Kontextes, vorbereiten des Adressbereiches für den nächsten Prozess).

3.4 Compare the long term scheduler to the short term scheduler.

Long-term scheduler(Steuert Grad der Multiprogrammierung, kann Prozesse schlafen legen, optional)
Short-term scheduler(Sucht zu jedem Quantum einen neuen Thread aus, Scheduler über den wir immer reden)

3.5 Describe 3 properties related to process and thread control objects.

- Prozess: Speicherbereich, Pfad der ausführbaren Datei
- Thread: CPU Kontext, Schedulingzustand
- Beides: Nicewert(Prioritäten), ID

3.6 Describe 3 properties related to the CPU context.

Prozessorstatuswort, Stackregister, Datenregister, Programmcounter

3.7 Outline the sequence of a context switch as performed by the dispatcher.

Aktuellen Thread unterbrechen → Aktuellen Zustand sichern im Hauptspeicher → Daten vom anderen Thread zurückschreiben → anderen Thread ausführen

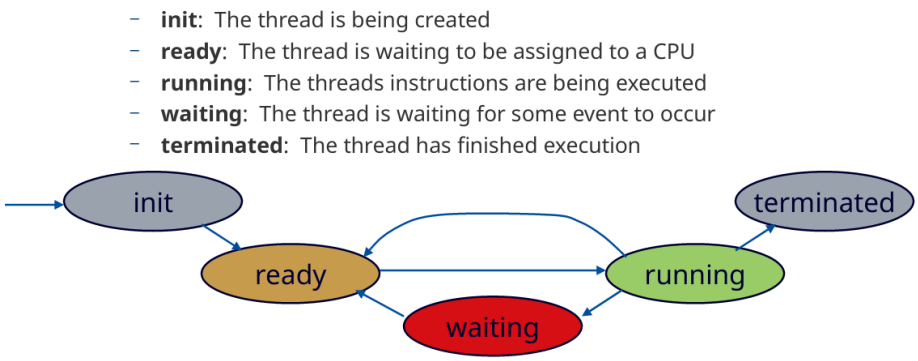
3.8 Compare cooperative to preemptive scheduling.

cooperative freiwillige Abgabe der CPU
preemptive Unterbrechung, wenn Quantum abgelaufen

3.9 What is a quantum? What effect does the length of a quantum have on the scheduler performance?

Ein Quantum ist eine feste Zeitscheibe, welche angibt wie lange ein Prozess rechnen darf, bis er unterbrochen wird. Längeres Quantum sorgt für eine höhere Performanz. Kürzeres Quantum ist responsiver.

3.10 Threads can have different states in relation to scheduling. Draw a diagram and Describe the purpose of the states and their transitions.



3.11 When are scheduling decisions made?

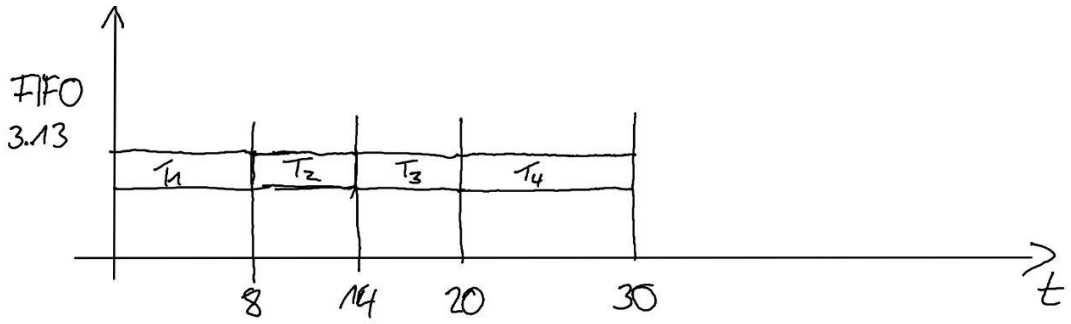
Am Ende von jedem Quantum. Welcher Thread als nächstes in **running** darf. Immer wenn ein neuer thread im system ankommt.

3.12 What are the optimiztion criteria of a scheduler?

- Resposivität
- Durchsatz
- Wartezeit
- Turn Arount Time(Zeit vom ersten Ausführen eines Threads bis zum letzten Ausführen)
- Auslast der CPU

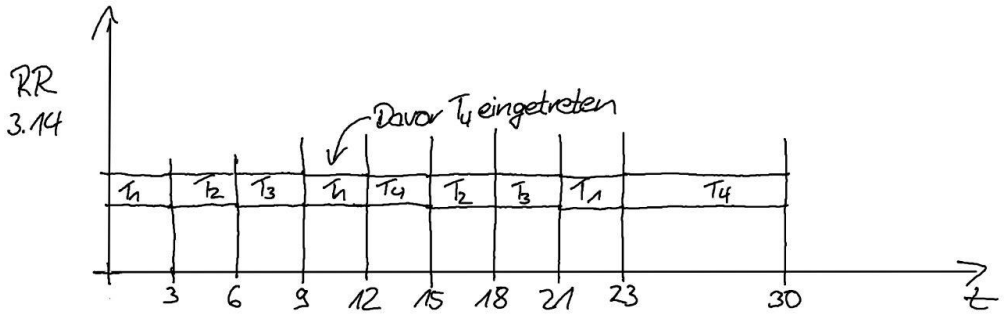
3.13 Draw a FIFO Gantt Chart for the given taskset. Assume a context switch takes no time.

Thread	Arrival Time	Burst Time
T_1	0	8
T_2	2	6
T_3	2	6
T_4	4	10



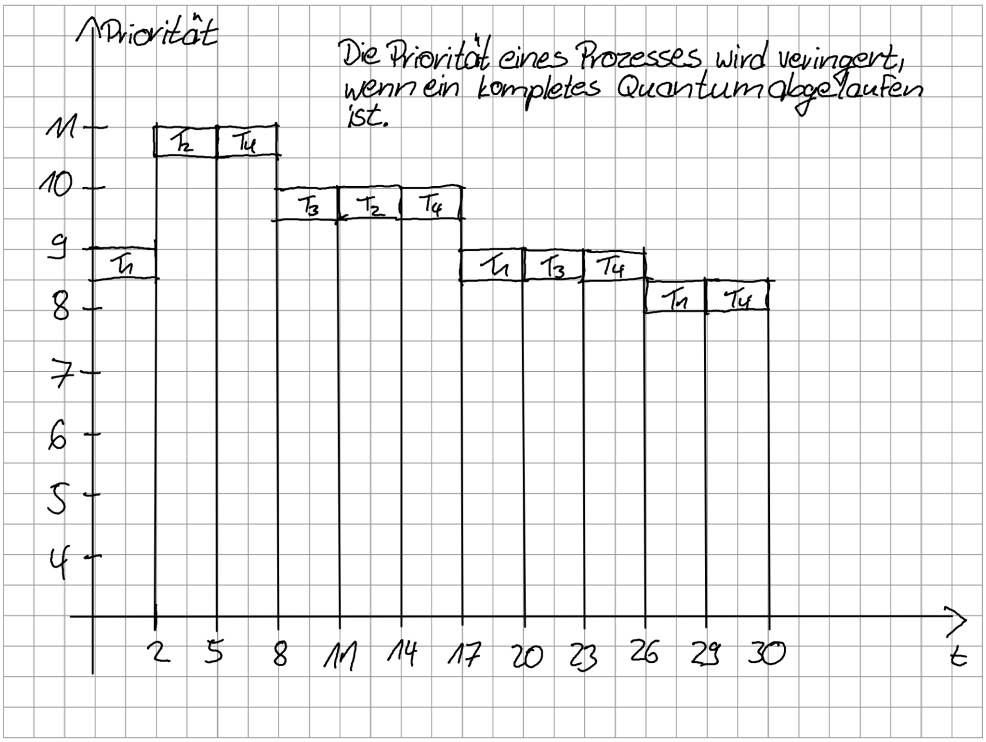
3.14 Draw a Round Robin Gantt chart for the given taskset. Assume a context switch takes not time, use quantum of length 3

Thread	Arrival Time	Burst Time
T_1	0	8
T_2	2	6
T_3	2	6
T_4	4 + 0	10



3.15 Draw a Round Robin Gantt chart for the given taskset. Assume a context switch takes not time, use quantum of length 3

Thread	Arrival Time	Burst Time	Priority
T_1	0	8	9
T_2	2	6	11
T_3	2	6	10
T_4	4	10	11



3.16 Calculate for each thread waiting time, turnaround time, throughput and compare

- **waiting time** eintreten in die Queue bis zur Ausführung der ersten Instruktion
- $TimeOfFirstInstruction - ArrivalTime = WaitingTime$

- FIFO
  - $T_1 = 0$
  - $T_2 = 8 - 2 = 6$
  - $T_3 = 14 - 2 = 12$
  - $T_4 = 20 - 4 = 16$
- RR
  - $T_1 = 0$
  - $T_2 = 3 - 2 = 1$
  - $T_3 = 6 - 2 = 4$
  - $T_4 = 12 - 4 = 8$
- RR + Prio
  - $T_1 = 0$
  - $T_2 = 0$
  - $T_3 = 8 - 2 = 6$
  - $T_4 = 5 - 4 = 1$

- **Turn Around Time** Zeit die der Task im System verbringt

- FIFO
  - $T_1 = Burst = 8$
  - $T_2 = Burst = 6$
  - $T_3 = Burst = 6$
  - $T_4 = Burst = 10$
- RR
  - $T_1 = 23$
  - $T_2 = 18 - 3 = 15$
  - $T_3 = 21 - 6 = 15$
  - $T_4 = 30 - 12 = 18$
- RR + Prio
  - $T_1 = 29$
  - $T_2 = 14 - 2 = 12$
  - $T_3 = 23 - 8 = 15$
  - $T_4 = 30 - 5 = 25$

- **response time** Eintritt in die Warteschlange bis zur erste Antwort des BS.
- **throuput** Menge der Threads pro Zeiteinheit. Hier: 4 Threads/30 Zeiteinheiten(auf Grund der Annahme, Kontextwechsel==0 !)

3.17 Describe the problem of thread starvation in relation to priority schedulers.

Ein Thread mit hoher Priorität hält einen Thread mit niedriger Priorität vom laufen ab.

3.18 What is a multilevel queue scheduler? Linux and Windows both contain multilevel queue schedulers. Name an example for a queue level and describe what properties threads in this queue have.

Scheduler der mehrere queues hat. In einander verschachtelt.
Windows: 16 Prios + 16 Realtime Prios (Algorithmen)
Linux: FIFO, RR + Batch, Idle
Missing!
Unsure:

3.19 Name 2 functions related to process or thread creation.

Linux: PThreadcreate(), fork()
Windows: CreateProcess(), CreateThread()

3.19.1 Name 2 functions related to process or thread termination.

Linux: exit(), pthreadexit()
Windows: kill(), abort()

3.20 Compare the Linux semantics of process creation (fork/exec) to the windows semantics of process creation (CreateProcess).

Linux fork(), exec() habe ich 2 Funktionen in den ich Sachen machen kann.
Windows in CreateProcess kann ich keine Sachen machen und ich habe nur eine Funktion. Parameterübergabe für Pipes als Argument integriert.

3.21 What are the advantages and disadvantages of multithreading in a programm?

Wir müssen unsere geteilten Resourcen vor einander schützen. Wir können eine höhere Auslastung des Systems erreichen.

3.22 Compare User Mode Threads to Kernel Mode Threads. When would you use which?

Kernel Mode Threads sind Threads die wirklich auf Threads im Kernel abbilden. Umgekehrt für User Mode Threads.

3.23 What are Fibers?

a: User Mode Threads on Windows
b: The Windows equivalent of UNIX pipes
c: Units of time counted in the Linux Kernel
d: Users in a traditional UNIX system

Antwort **A**

3.24 What are cgroups?

a: groups of capabilities of an executable
b: groups of redundant file system entries
c: isolated process control groups in Linux
d: groups of users in Windows

Antwort **C**: Prozessgruppen die voneinander isoliert werden. Bsp. Docker

3.25 Describe the specific challenges of real-time scheduling. What is the difference between soft and hard real-time?

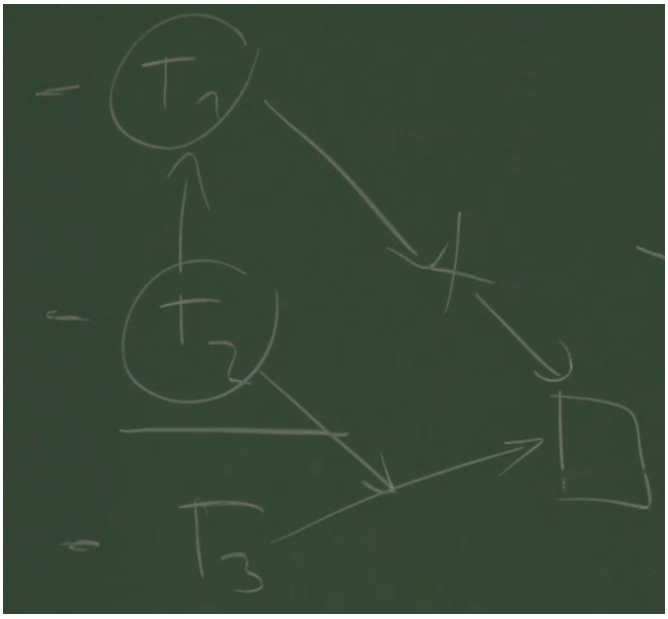
Software die mit Deadlines umgehen können muss. Soft: Dienst degradiert Bsp. Youtube, Hard: Autokontrolle, Flugzeugkontrolle → Unfall, Katastrophe
Sehr schwierig, weil man kann die Verzögerungen nicht im Vorraus kennen(Caches, Interrupts)

3.26 Describe one of the three discussed approaches of starvation avoidance.

PrioBoosting on Windows
PrioAgeing, Niceness on Linux
PrioBoosting: Thread bekommt kurz eine sehr hohe Prio sinkt dann aber wieder ab
PrioAgeing: Threads werden Stück für Stück abgesenkt

3.27 What is priority inversion?

Thread mit niedriger Prio hält Thread mit hoher Prio vom laufen ab → Locks



3.28 Which of the following is not a queue level in Completely Fair Scheduler (CFS)?

a: SCHED_BATCH
b: SCHED_HIGH
c: SCHED_RR
d: SCHED_FIFO

Antwort **b**: ist Ausgedacht von dem Dozenten

3.29 Describe the concept of niceness in relation to the Linux Completely Fair Scheduler (CFS).

Wir haben einen relativen Wert zwischen ±20. Der Wert beschreibt das Verhältnis Länge der Ausführungszeit, die die Threads zu einander bekommen mit einer Nicewert Differenz von ca 1,25 entsprechend der Laufzeit.
Ein Thread der 1 niedrigeren Nicewert hat als ein anderer Thread, wird 1,25 mal so lange ran kommen.

3.30 What is thread affinity? Compare hard affinity and soft affinity.

Bitmaske, welche auf einem Multicore System festlegt auf welchen Core ich laufen darf. Ich darf auf den Kernen laufen, auf denen meine Affinitätsmaske eine 1 hat.
Bei harter Affinität kann dies manuell eingestellt werden.
Bei softer Affinität wird dem Thread zu erst der Core zugewiesen, welcher am besten ist. Beim Fortsetzen wird versucht wieder diesen Core zu benutzen da hier alle Daten schon vorhanden sind.

3.31 On a high level, outline the similarities and differences of the Windows and Linux Scheduler.

Gemeinsamkeiten: Prioritäten
Linux: Nicewerte
Windows:
...
Missing!

4 Checkpoint 4

4.2 What is the role of the Memory Management Unit (MMU)

Sie übersetzt logische (Adresse im Programm) auf physikalische Adressen (Adresse auf Festplatte). (Passiert bei jedem Speicherzugriff neu.)

4.3 What is the difference between virtual / logical Adresses and physikal adresses?

Der physische Bereich existiert genau ein mal. Seine Größe ist abhängig von der Anzahl der Module.
Es gibt einen virtuelle Bereich für jeden Prozess. Diese sind eine Abbildung auf den physikalischen Bereich. Seine Größe ist abhängig von der Anzahl der Adressen.

4.4 What is Segmentation?

Einteilung des Speichers in abgeschlossene Segmente. Ein Segment ist charakterisiert durch seine Startadresse und seine Größe. Sie sind Speicherreservierungen für einzelne Prozesse. Damit können wir sichergehen, dass ein Speicherzugriff eines Prozesses immer in seinem Segment landet.

4.5 What are the main shortcommings of Segmentation?

externe Fragmentierung: Prozesse terminieren und ihr Speichersegment wird freigegeben. Damit entstehen aber Löcher zwischen Segmenten, die eventuell nicht gefüllt werden können.
interne Fragmentierung: Ein Prozess nutzt nicht das gesamte Segment, das er reserviert hat.
sehr unflexibel in Speichergröße: Häufig wird deswegen mehr Speicher reserviert, was aber zu mehr Fragmentierung führt.
Anzahl der Prozesse ist limitiert durch die Größe der reservierten Speicher.

4.6 Describe Paging. How is it different from Segmentation, and how does it address the shortcumings of Segmentation?

Ein Prozess kann mehrere Speicherbereiche (Pages) haben. (Anders als bei Segmentation.) Die Anzahl der Pages ist dabei dynamisch, also zur Laufzeit, festgelegt.
Damit gibt es keine externe Fragmentierung mehr, da Löscher zwischen den Pages wieder mit neuen Pages aufgefüllt werden können und es gibt weniger interne Fragmentierung da die Pages kleiner sind und wir uns bei Bedarf neue Pages dazu holen können. Da Pages ausgelagert werden können, ist die Anzahl der Prozesse auch nicht mehr limitiert.
(Page) Frame: ein physischer Speicherblock
page: ein virtueller Speicherblock; manchmal wird der Begriff auch für beides verwendet.

4.7 Describe the layout of a Page Table.

Eine Tabelle welche virtuelle Speicherbereiche auf physische Speicherbereiche abbildet. Eine Page Table ist zur Laufzeit veränderbar und wird von der MMU genutzt. In der Praxis ist es nur eine Liste mit physichen Page Frame (Nummern), und die virtuelle Page (Nummer) ist der Index.

4.7.1 Why do some architectures use multi-level page table structures?

Um Speicherplatz zu sparen, da eine Page Table sehr groß ist. Damit müssen wir nur die Page Table Einträge wirklich ausfüllen, welche wir auch verwenden. (Beispiel: Bei Intel 64-Bit Systemen gibt es vier Ebenen an Page Tables.)

4.8 Name two bits associated with each Page Table Entry and describe their purpose.

Dirty: wurde auf die Page geschrieben? Wenn ja, müssen wir sie abspeichern, bevor wir sie wegwerfen.
Read/Write/Execute: Berechtigungsbits
Valid: wenn die Page eine gültige Abbildung im physikalischen Speicher hat. (Wenn das Valid Bit nicht gesetzt ist, darf auf die Page nicht zugegriffen werden.)
(**ToDo:** Weitere siehe Folien)

4.9 What is the role of the Translation Lookaside Buffer (TLB)?

Es ist ein Cache für Seitenzugriffe: Es speichert kürzlich erfolgte Umwandlungen von virtuellen in physische Adressen. Damit sparen wir uns das Nachsehen in diversen Page Tables, wenn wir einen Cache Hit haben. Bei einem Miss wird das Ergebnis der Umwandlung der MMU in die TLB eingetragen. Es funktioniert mit Registern und ist entsprechend sehr schnell.

4.10 Assume a memory system that manages adresses of 26 bits length, with a page size of 256 bytes.

Die Reihenfolge wurde leicht geändert.

4.10.1 What is the size of the virtual adress space?

Adresslänge: 26 Bit
Größe des virtuellen Addressraumes (entspricht Anzahl aller möglichen Adressen): 2^{26} Byte = $2^6 * 2^{20}$ Byte = 64 MiB

4.10.2 How long are the adresses' page number and offset?

Page Size: 256 Byte = 2^8 Byte
$\Rightarrow$ Offset Länge (entspricht der Anzahl an Bits die wir benötigen, um alle Adressen in einer Page zu bilden): 8 Bit
Page Number Länge (Die verbleibenden Bits können dann für das Adressieren der Page Nummer in der Page Table genutzt werden.): 26 Bit − 8 Bit = 18 Bit

4.10.3 How many entries does a page table have?

Annahme: Single Level Page Table
Anzahl der Einträge in der Page Table (entspricht der Anzahl der Adressen die mit der Page Number Länge gebildet werden können): 2^{18}

4.13 *Same as above.* Each page table entry has two status bits.

4.13.1 What is the length of a page table entry?

Annahme: Addresslänge einer virtuellen Adresse ist identisch zur Adresslänge einer physischen Adresse.
Länge eines Page Table Eintrags: 18 Bit +2 Bit = 20 Bit

4.13.2 What is the total size of a page table?

Es kann auch sein, dass ein Eintrag auf Wort Länge (ist meist Addresslänge, also hier 26 Bit) aufgerundet wird. Hier ist das aber nicht der Fall.
Gesamtgröße (ist Länge eines Eintrages * Anzahl der Einträge):

$$\begin{aligned} 2^{18} * 20\,\text{Bit} &= 2^3 * 2^{15} * 20\,\text{Bit} \\ &= 2^{15} * 20\,\text{Byte} \\ &= 2^{10} * 2^5 * 20\,\text{Byte} \\ &= 2^{10} * 2^6 * 10\,\text{Byte} \\ &= 2^6 * 10\,\text{KiB} \\ &= 640\,\text{KiB} \end{aligned}$$

Zum Vergleich, wenn auf Addresslänge aufgerundet wird: 832 KiB
Wenn auf Bytelänge (also 24) aufgerunder wird: 768 KiB

4.14 Assume the described system has access to 24MiB of physical memory, half of which is reserved for the operating system. Of this half, 5MiB are reserved for process page tables, and are not pageable. How many processes can this system support?

Aufteilung ist also: 5 MiB für Page Tables, 7 MiB für andere Dinge vom Betriebssystem, 12 MiB für Prozesse.
Da jeder Prozess eine eigene Page Table hat, schauen wir, wie viele Page Tables wir in diese 5 MiB kriegen:

$$\begin{aligned} 5\,\text{MiB} / 640\,\text{KiB} &= 5 * 2^{10}\,\text{KiB} / 640\,\text{KiB} \\ &= 5 * 2^{10} / 640 \\ &= 2^{10} / 128 \\ &= 2^{10} / 2^7 \\ &= 2^3 = 8 \end{aligned}$$

Bei einer ungeraden Zahl müssen wir logischerweise abrunden.

4.15 What is the maximum size of a process working set, assuming that no reserved OS pages are part of the working set (4.19)?

Ein Prozess kann maximal 12 MiB an Speicher reserviert haben.

4.16 What is a Page Fault?

Zugriff auf eine Page auf der das Valid Bit nicht gesetzt ist. Ein Page Fault löst wiederum einen Interrupt aus.

4.17 What is the difference between a Soft Page Fault and a Hard Page Fault?

Soft Page Fault: Die Page bedindet sich noch im Cache und kann noch leicht zurück geholt werden.
Hard Page Fault: Die Page muss komplett neu in den Hauptspeicher (von der Platte) geladen werden.

4.18 Name two different reasons for which a memory access coul produce a Page Fault.

Daten sind bereits ausgelagert und befinden sich nicht mehr im Hauptspeicher, Wir schreiben auf eine Copy-On-Write Page, Zugriff auf reservierten aber nicht committeden Speicher (4.28), wir greifen auf eine Page zu, auf die wir keine Berechtigung haben, wir greifen auf Speicher zu an dem sich noch keine Page befindet

4.19 What is a Working Set?

Menge aller Hauptspeicherseiten auf die ein Prozess zugreifen kann, ohne ein Page Fault zu erzeugen.

4.20 What is the role of the Modified, Standby, Free, Zero and Bad Page Lists?

Die Listen ergänzen die Working Sets der Prozesse um eine genauere Einteilung der Pages zu ermöglichen.

4.20.1 What properties do pages in these lists have?

Modified: Seiten die noch zu einem Prozess gehören, bei denen aber das Dirty Bit gesetzt ist

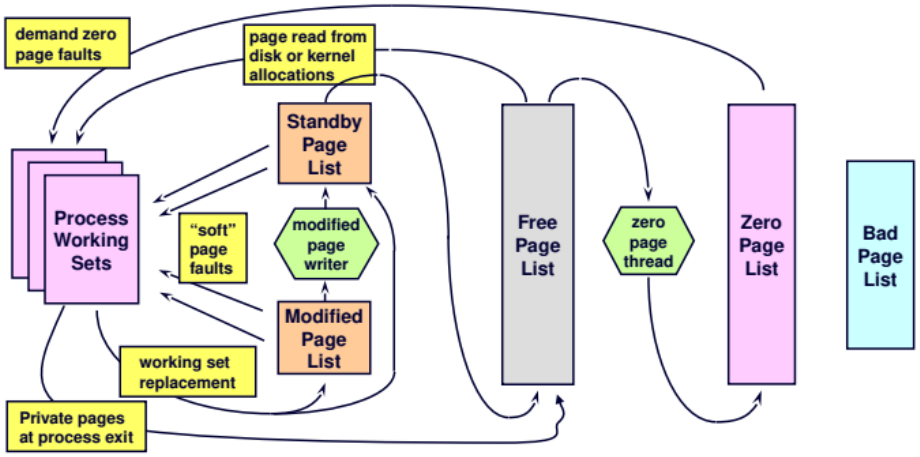
Standby: ähnlich, nur ist das Dirty Bit hier nicht gesetzt

Free: Seiten die zu keinem Prozess gehören, aber noch Restdaten enthalten

Zero: komplett genullte Seiten

Bad: defekte Seiten, werden vom System zur Boot Zeit beim Memory Check erkannt

4.20.2 How do pages transition between these lists?



Free nach Zero: Zero Page Thread (wird gemacht sobald Zeit ist; muss aber nicht gemacht werden, wenn die Seite sowieso überschrieben wird, zum Beispiel beim Laden einer Binary)
Working Set nach Modified / Standby: Balance Set Manager Thread, falls das System den Speicher für andere Prozesse braucht, oder falls Speicher lange nicht mehr genutzt wurde
Modified nach Standby: sobald Änderungen gesichert wurden
Working Set / Modified / Standby nach Free: wenn Prozess terminiert

4.21 What is the purpose of the Page Frame Number Database?

Struktur bei der für jede physikalische Page Frame ein Eintrag existiert.

4.22 Describe the page replacement algorithms First-In-First-Out (FIFO), Second Chance, and Least-Recently-Used (LRU).

Page replacement: wenn eine Page aus dem Working Set in die Modfied / Standby List verschoben wird.

FIFO: Die alteste Page Frame wird ausgelagert

Second Chance: FIFO über alle Seiten, die (seit dem letzten Page replacement) nicht verwendet wurden (in der Regel besser als FIFO)

LRU: Die am längsten nicht verwendete Page wird ausgelagert (die teuerste Methode von allen)

4.23 What is Béládys Anomaly, as found in the FIFO page replacement algoritm?

Mehr physikalische Pages können dazu führen, dass FIFO replacement schlechter funktioniert.

4.24 Assuming three frames of physical memory, and a memory access pattern to the virtual pages: 2,1,4,2,3,4,2,1,3,4,3. How many Page Fault would FIFO incur?

	2	1	4	2	3	4	2	1	3	4	3
1.	2	2	2	**2**	3	3	3	3	**3**	4	4
2.		1	1	1	1	1	2	2	2	2	3
3.			4	4	4	**4**	4	1	1	1	1

Unterstriche: Page Fault; Fett: Page Hit
Insgesamt 8 Page Faults (5 ohne die initialen)

4.25 How many Page Faults would Second Chance incur?

	2	1	4	2	3	4	2	1	3	4	3
1.	2	2	2	**2**	2	2	**2**	2	3	3	**3**
2.		1	1	1	3	3	3	1	1	1	1
3.			4	4	4	**4**	4	4	4	**4**	4

Fett: Access Bit wird gesetzt
Insgesamt 6 Page Faults.

4.26 How many Page Faults would LRU incur?

	2	1	4	2	3	4	2	1	3	4	3
1.	2	2	2	**2**	2	2	**2**	2	2	4	4
2.		1	1	1	3	3	3	1	1	1	1
3.			4	4	4	**4**	4	4	3	3	**3**

Fett: Page Hit
Insgesamt 7 Page Faults.

4.27 What is the difference between swapping and page replacement (paging)?

Swapping bedeutet eigentlich das gesamte Segment eines Prozesses auszulagern. (Damit ist der Prozess effektiv nicht mehr lauffähig.) Heute werden beide Begriffe für das gleiche Prinzip (page replacement) verwendet. Page replacement ist das auslagern einzelner (selten benötigter) Pages.

4.28 Describe the difference between reserved and committed memory.

Reservierter Speicher ist der als verwendbar markierte virtuelle Speicher. Committed Speicher ist der einem Prozess zur Verfügung stehende physikalische Speicher.

4.29 What is a Guard Page? What could it be used for?

Eine Page welche das Ende eines Buffers (Arrays) markiert und dafür sorgt, dass ein Interrupt erzeugt wird, sobald auf die Guard Page zugegriffen wird und in diesem Interrupt wird dann der Buffer um eine weitere Page verlängert (eine weitere Page committed), an dessen Ende wieder eine Guard Page steht. Eine Guard Page kann genutzt werden um Zugriff auf nicht reservierten Speicher oder Stack Overflows zu verhindern.

4.30 How do operating systems implement shared memory?

Ein Eintrag in der Page Table von zwei Prozessen zeigt auf die gleiche (physikalische) Page Frame.

4.31 What is Copy-on-Write (CoW) memory, and what is it used for?

Bei einem fork eines Prozesses, wird der Speicher in beiden Prozessen als CoW markiert. Die physikalische Page Frame ist dabei identisch. Sobald ein Prozess dann aber in diesen geteielten Speicher schreiben will, wird er kopiert, damit jeder Prozess seinen eigenen (unabhängigen) Speicher hat. Damit sparen wir uns das Kopieren von Speicher, den wir vielleicht nie verändern.