Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Lukas Werling, M.Sc.
Dr. Marc Rittinghaus

Musterlösung Hauptklausur
03.03.2022

Alle Punkteangaben ohne Gewähr!

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Namen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf den anderen Blättern (auch auf Konzeptblättern) Ihre Matrikelnummer ein.

  Please fill in your last name, your first name, and your matriculation number on this page and fill in your matriculation number on all other (including draft) pages.

- Die Prüfung besteht aus 25 Blättern: Einem Deckblatt, 24 Aufgabenblättern mit insgesamt 5 Theorieaufgaben und 3 Programmieraufgaben sowie 0 Seiten Man-Pages.

  The examination consists of 25 pages: One cover sheet, 24 sheets containing 5 theory assignments as well as 3 programming assignments, and 0 sheets with man pages.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed!

- Die Prüfung ist nicht bestanden, wenn Sie versuchen, aktiv oder passiv zu betrügen.

  You fail the examination if you try to cheat actively or passively.

- Sie können auch die Rückseite der Aufgabenblätter für Ihre Antworten verwenden. Wenn Sie zusätzliches Konzeptpapier benötigen, verständigen Sie bitte die Klausuraufsicht.

  You can use the back side of the assignment sheets for your answers. If you need additional draft paper, please notify one of the supervisors.

- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit mehreren widersprüchlichen Lösungen werden mit 0 Punkten bewertet.

  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).

- Programmieraufgaben sind gemäß der Vorlesung in C zu lösen.

  Programming assignments have to be solved in C.

Die folgende Tabelle wird von uns ausgefüllt!
The following table is completed by us!

Aufgabe	T1	T2	T3	T4	T5	P1	P2	P3	Total
Max. Punkte	9	9	9	9	9	15	15	15	90
Erreichte Punkte									

Aufgabe T1: Grundlagen

Assignment T1: Basics

a) Warum ergibt es Sinn, auch dann ein Betriebssystem zu verwenden, wenn auf einem System nur eine Anwendung ausgeführt werden soll? **1 pt**

Why does it make sense to use an operating system even if only one application is to be executed on a system?

Solution:

Possible answers **(1.0 P)***:*

- *Avoid engineering effort to support necessary devices and perform basic OS-tasks such as handling I/O, scheduling and dispatching threads, or memory management.*
- *Application is not restricted to run only on the particular system due to the OS abstractions (i.e., better compatibility)*

Other plausible answers are accepted.

b) Bei einem Systemaufruf können Parameter über den Stack oder Register übergeben werden. Erläutern Sie eine zusätzliche Maßnahme, die bei der Übergabe per Stack ergriffen werden muss, um die Sicherheit des Systems zu gewährleisten. **1 pt**

In a system call, parameters can be passed via the stack or registers. Explain one additional measure that must be taken when passing parameters by stack to ensure the security of the system.

Solution:

The parameters need to be copied from the user-mode stack of the thread to its kernel-mode stack before verification (check alone not sufficient as the same for registers) so that they cannot be manipulated by other user-mode threads that are running concurrently **(1.0 P)***. Alternative: The stack pointer and touched range must be sanitized to be entirely in user mode.*

Nennen Sie je einen Vorteil und einen Nachteil ohne Sicherheitsbezug für die Parameterübergabe mittels Registern. **1 pt**

Without taking security into account, name one advantage and one disadvantage of passing parameters via registers.

Solution:

Advantage: Saves memory accesses **(0.5 P)**
Disadvantage: Limited number of parameters **(0.5 P)**

Note: Saving of registers would be necessary anyway as they will be invalidated by kernel code execution. So even less registers need to be saved if some of them are used for parameter passing!

c) Ein `try`/`catch`-Block behandelt zwei Ausnahmen: *ArithmeticException* und *IllegalArgumentException*. Welche der Ausnahmen könnte ursprünglich von der CPU ausgelöst werden? Geben Sie ein Beispiel für eine solche Situation an. **1 pt**

A `try`/`catch` *block handles two exceptions:* ArithmeticException *and* IllegalArgumentException*. Which of the exceptions might originally be raised by the CPU? Give an example for such a situation.*

Solution:

`ArithmeticException`*, for example, due to a division by zero.*

d) Für die Synchronisation eines kritischen Abschnitts im Kernel wird ein Spinlock verwendet. Welche zusätzliche Maßnahme kann nötig sein, um eine korrekte Synchronisation zu erreichen und wann ist diese notwendig? **1.5 pt**

To synchronize a critical section in the kernel, a spinlock is used. What additional action may be necessary to ensure correct synchronization and when is it necessary?

Solution:

It might be necessary to also disable interrupts **(0.5 P)** *if the critical section accesses state that may also be touched by interrupt handlers* **(1.0 P)***.*

If the handler per-se does not acquire the spinlock, mutual exclusion is not given. If the interrupt handler tries to acquire the lock while a thread is in the CS, the interrupt handler will deadlock the system. Somehow preempting the spinlock is not a solution and equivalent to not acquiring the lock in the interrupt handler at all.

All solutions involving blocking are not accepted as the question states that we are using a spinlock, which per definition does not block a thread.

e) Die Tabelle zeigt die Sektionen einer ELF-Datei. Die zweite Spalte gibt die Position der Sektion in der Datei an. Die dritte Spalte spezifiziert, an welche Adresse die Sektion geladen werden soll. **2 pt**

Das Betriebssystem soll die in der letzten Spalte eingetragenen Zugriffsberechtigungen auf den virtuellen Speicher umsetzen. Zu welchem Problem kommt es bei seitenbasierter Speicherverwaltung mit 4 KiB Seiten? Passen Sie die Tabelle so an, dass die Rechte mit minimalem Ressourcenverbrauch umgesetzt werden können.

The table shows the sections of an ELF file. The second column is the section's position in the file. The third column specifies at which address it should be loaded.

The operating system shall apply permissions to the virtual memory as given in the last column. What problem might occur with page-based memory management and 4 KiB pages? Adjust the table so that the permissions can be applied with minimal resource consumption.

Name	File Offset	Virtual Address	Permissions[1]
`.text`	`0x01000`	`0x01000`	`R-X`
`.data`	`0x03c10`	`0x03c10` → **`0x04000`**	`RW-`
`.bss`	**`0x04f60`**	**`0x10000`**	**`RW-`**
`.rodata`	`0x04f60`	`0x04f60` → **`0x06000`**	`R--`

[1] R: read | W: write | X: execute

Instead of `"0x04f60"`, also `"0x0"`, and `"-"` and similar are accepted to indicate that `.bss` does not physically consume space in the file (except the ELF header itself). However, the file offset must not be outside the file size that can be deduced from the assignment.

Solution:

The sections are not page-size aligned so that protections cannot be set because they must be configured in the page table and apply for a whole page (e.g., 4KiB) **(1.0 P)**.

The next 4 KiB aligned address for `.data` *is* `0x04000` **(0.5 P)**. *For* `.rodata` *it is* `0x06000` *as the* `.data` *section is* `0x04f60 - 0x03c10 = 0x01350` *bytes in size* **(0.5 P)**.

Passen Sie die Tabelle so an, dass zwischen `.data` und `.rodata` in der Datei eine 10 KiB `.bss` Sektion liegt, die mit so geringen Berechtigung wie möglich an Adresse `0x10000` geladen wird. **1.5 pt**

Adjust the table so there is a 10 KiB `.bss` *section between* `.data` *and* `.rodata` *in the file, which is loaded at address* `0x10000` *with as few permissions as possible.*

Solution:

The `.bss` *section does not take up any space on disk. So the file offset of* `.rodata` *does not need to be adjusted* **(0.5 P)**. *The* `.bss` *section contains zero-initialized but writable data so it needs* `RW-` *permissions at minimum* **(0.5 P)**. *The other fields can be taken from the question* **(0.5 P)**.

Total: 9.0pt

Aufgabe T2: Prozesse und Threads

Assignment T2: Processes and Threads

a) Geben Sie einen anderen Namen für das hybride Threadmodell an. **0.5 pt**

Give an alternate name for the hybrid threading model.

Solution:

M:N model

b) Gegeben seien vier Prozesse auf einem Einprozessorsystem mit den angegebenen Ankunftszeiten (0 = Start) und Burst-Zeiten. Vervollständigen Sie die untenstehenden Scheduling-Pläne für die Strategie PSJF sowie die Strategie RR für die Zeitscheibenlänge 2 Zeiteinheiten. Bei der RR-Strategie hängt der Scheduler zuerst neue Prozesse (falls vorhanden) ans Ende der Ready-Queue und fügt dann den vorherigen Prozess ans Ende ein (wenn er noch lauffähig ist). Ein Kasten im Zeitplan stellt eine Zeiteinheit dar. Der Scheduler wird nach jeder Zeiteinheit ausgeführt. **4 pt**

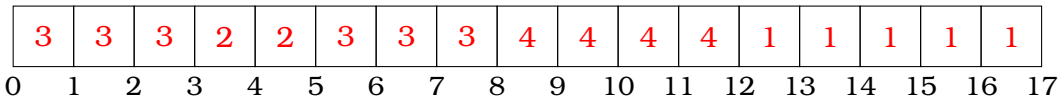
Consider four processes on a uniprocessor system with given arrival times (0 = start) and burst times. Complete the scheduling plans given below for the policy PSJF and the policy RR for a timeslice length of 2 units of time. For RR, the scheduler first adds new processes (if any) to the tail of the ready queue and then inserts the previous process to the tail (if it is still runnable). A box in the scheduling plan represents one unit of time. The scheduler is executed after each unit of time.

Process	Arrival Time	Burst Time
1	1.5	5
2	2.5	2
3	0	6
4	3.5	4

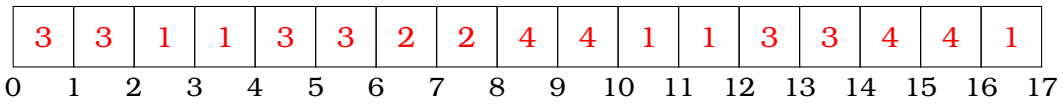
Solution:

*-**0.5 P** for each incorrect scheduling decision*

PSJF



RR (timeslice length 2)



Berechnen Sie für den obigen *PSJF-Scheduling-Plan* die Wartezeit der angegebenen Prozesse. **1 pt**

For the above PSJF scheduling plan, calculate the waiting time of each given process.

Solution:

The waiting time for each process can be calculated as $t_{wait} = t_{finish} - t_{arrival} - t_{burst}$

Process	*Finish time*	*Arrival time*	*Burst time*	*Waiting time*
1	*17*	*1.5*	*5*	*10.5*
3	*8*	*0*	*6*	*2*

Berechnen Sie für den obigen *RR-Scheduling-Plan* die Tournaround-Zeit der angegebenen Prozesse. **1 pt**

For the above RR scheduling plan, calculate the tournaround time of each given process.

Solution:

The turnaround time for each process can be calculated as $t_{turnaround} = t_{finish} - t_{arrival}$

Process	*Finish time*	*Arrival time*	*Turnaround time*
2	*8*	*2.5*	*5.5*
4	*16*	*3.5*	*12.5*

c) Füllen Sie die Lücken in dem untenstehenden Text aus. **2.5 pt**

Fill in the gaps in the text below.

Solution:

(0.5 P) *per gap*

Ein Prozess kann zu einem bestimmten Zeitpunkt verschiedene **Zustände** haben. Im Running -Zustand führt ein Prozess Instruktionen auf einem Prozessor aus. Wenn ein Prozess eine I/O-Anfrage initiiert, wird er eine Zeit lang blockiert (oder warten), sodass er nicht unnütz Rechenzeit verschwendet. Nachdem ein Prozess schließlich `exit` aufruft, bleibt er im Terminated (oder Zombie) -Zustand erhalten, bis sein Elternprozess seinen Exit-Status ausliest.

*A process can be in different **states** at a given time. In the running state, a process is executing instructions on a processor. When a process initiates an I/O request, it becomes blocked (or waiting) for a while so that it does not unnecessarily waste processing time. Finally, after a process calls* `exit`*, it sticks around in the terminated (or zombie) state until its parent process reads its exit status.*

Total: 9.0pt

Aufgabe T3: Speicher

Assignment T3: Memory

a) Erläutern Sie zwei Nachteile von einfachen Basis- und Limitregistern gegenüber Segmenttabellen. **2 pt**

Explain two disadvantages of simple base and limit registers compared to segment tables.

Solution:

- *With base and limit registers all memory of a process needs to be contiguous in physical memory, as there is only one pair of registers describing the virtual address space layout. Segments tables, on the other side, can map different areas of physical memory independently.*
- *Sharing memory is very difficult with base and limit registers as the memory regions have to overlap in physical memory. It becomes impossible to share memory between more than two processes without breaking isolation.*
- *Base and limit registers generally do not allow to assign specific protections to certain areas of a process.*

Other plausible answers are accepted.

*Note: Stating that memory **cannot** be shared is wrong as it is possible to share memory between two processes by overlapping the physical memory regions. The same is true for stating that the assigned memory **cannot** grow. This is possible at the end of the memory region if the following memory is not already allocated to another process.*

b) Weshalb kann der Heap konzeptionell als Stack fungieren, jedoch nicht umgekehrt? **1 pt**

Why can the heap conceptually act as a stack, but not vice versa?

Solution:

The lifetime of data in a stack frame is limited to the thread's execution in the corresponding function.

c) Nennen Sie zwei Gründe, warum es Sinn ergibt, in einem Speicherallokator CPU-lokale Listen für kürzlich freigegebene Speicherbereiche zu führen. **1 pt**

Give two reasons why it makes sense to maintain CPU-local lists for recently freed memory areas in a memory allocator.

Solution:

- *As lists are CPU-local they do not need synchronization when memory blocks are allocated from them, which allocation faster*
- *The memory areas are more likely to be hot in the local CPU cache so that accesses do not cause cache misses (even for the first writes)*

Other plausible answers accepted.

Note: Simply stating that finding free memory is faster is not accepted as the allocator might maintain O(1) free lists anyway. This question is about special free lists that keep recently freed memory local to a certain CPU.

d) Gegeben sei ein System mit 38-bit virtuellen Adressen. Die Seitengröße beträgt 2 KiB, jeder Seitentabelleneintrag umfasst 32 Bit. Vergleichen Sie rechnerisch den Speicherbedarf einer linearen mit dem einer hierarchischen Seitentabelle. Gehen Sie davon aus, dass der gesamte virtuelle Adressraum belegt ist. **2 pt**

Given a system with 38-bit virtual addresses. The page size is 2 KiB, each page table entry comprises 32 bits. Compare the memory requirements of a linear page table with those of a hierarchical page table via calculation. Assume that the whole virtual address space is in use.

Solution:

The address space consists of $\frac{2^{38}}{2^{11}} = 2^{27}$ *pages. Each page requires 4 bytes in the page table.*

***Linear Page Table**:*
$2^{27} * 2^2 = 2^{29}$ *Bytes* $= 512$ *MiB.* **(0.5 P)**

***Hierarchical Page Table**:*
Each page table has $\frac{2^{11}}{2^2} = 2^9$ *PTEs, so each level in the page table covers 9 bits in the virtual address plus the 11 bits for the offset. This gives* $27/9 = 3$ *levels. We have 1 page table at the top-most level,* 2^9 *in the second level, and* 2^{18} *in the third level.* $(1 + 2^9 + 2^{18}) * 2^{11} = 2^{11} + 2^{20} + 2^{29}$ *bytes.*

So the hierarchical page table consumes roughly 1 MiB more memory than the linear table.

Wieviel virtueller Speicher darf bei der hierarchischen Seitentabelle nicht eingeblendet werden, damit beide Seitentabellen in dem genannten Szenario den gleichen Speicherverbrauch haben? **2 pt**

How much virtual memory must not be mapped with the hierarchical page table so that both page tables consume the same amount of memory in the given scenario?

Solution:

The hierarchical page table consumes $2^{11} + 2^{20}$ *bytes more memory. If we want to save this amount of memory in page tables, we have to use* $1 + 2^9$ *less page tables. We cannot save the top-most page table. But if we save* 1 *page table in the second level, we can save* 2^9 *page tables in the third level.*

With 2^9 *page tables in the third level, we can map* $2^9 * 2^9 * 2^{11} = 2^{29}$ *bytes of memory. So if more than* $2^{29} = 512$ *MiB of memory are not mapped, the hierarchical page table begins to save memory compared to the linear page table.*

e) Erläutern Sie, welchen Vorteil Tagging beim TLB hat. **1 pt**

Explain the advantage of tagging in a TLB.

Solution:

With tagging support in the TLB, the OS does not have to flush the whole TLB when an address space switch occurs. This allows (some) translations to remain cached in the TLB and reduces the TLB miss rate (and thus memory access latency) when switching between address spaces.

Total: 9.0pt

Aufgabe T4: Koordination und Kommunikation von Prozessen

Assignment T4: Process Coordination and Communication

a) Erklären Sie die Bedingung "No preemption" für Deadlocks. **1 pt**

Explain the condition "No preemption" for deadlocks.

Solution:

"No preemption" means that resources can only be returned by tasks voluntarily but cannot be taken away forcefully **(1.0 P)**.

Wie nennt man Techniken, die eine solche Voraussetzung für Deadlocks negieren? **0.5 pt**

How are techniques called that negate such a precondition for deadlocks?

Solution:

Deadlock Prevention **(0.5 P)**.

b) Erklären Sie den Unterschied zwischen direktem und indirektem Message Passing. **1 pt**

Explain the difference between direct and indirect message passing.

Solution:

Direct message passing means that tasks are addressed directly, whereas indirect message passing operations target mailboxes **(1.0 P)**.

Wann und weshalb kann asynchrones Empfangen komplett ohne Puffer implementiert werden? **1.5 pt**

When and why can asynchronous receiving be implemented completely without a buffer?

Solution:

When sending is synchronous **(0.5 P)**. *In that case, the sender can hold the data until they can be copied to the receiver* **(1.0 P)**.

c) Weshalb ist die x86-Instruktion `add memory, 1`, die den Wert an der Speicheradresse `memory` inkrementiert, nicht atomar? **1 pt**

Why is the x86 instruction `add memory, 1` *which increments the value at the address* `memory` *not atomic?*

Solution:

Because the CPU first loads the value and then later stores the modified value **(1.0 P)**. *Other concurrent accesses between those two accesses cause race conditions.*

Note that all accesses are still part of the same instruction and the program cannot be preempted between the accesses, so there are no problems on uniprocessor systems.

d) Eine Anwendung schützt sämtliche von mehreren Threads genutzten Daten mit einem einzigen Mutex. Nennen Sie einen Vor- und einen Nachteil dieses Ansatzes gegenüber vielen Mutexes, die jeweils kleine Teilmengen der Daten schützen. **1 pt**

An application protects all data that is used by multiple threads with a single mutex. Name an advantage and a disadvantage of this approach compared to many mutexes which each protect a small subset of the data.

Solution:

Potential advantages **(0.5 P)***:*

- *Easier to implement*
- *Less memory required to store the mutexes*
- *No deadlocks*

Potential disadvantages **(0.5 P)***:*

- *Threads often have to wait even though the data they want to access is not used by any other thread.*

Gegeben sei eine Variable `x`, die durch einen Mutex `mutex` geschützt wird. Der folgende Code versucht, Berechnungen auf `x` außerhalb des kritischen Abschnittes durchzuführen. Diskutieren Sie die Lösung bezüglich des Auftretens von Race Conditions. **1.5 pt**

Assume a variable `x` that is protected by a mutex `mutex`. The following code tries to execute calculations on `x` outside of the critical section. Discuss the solution regarding the occurrance of race conditions.

```
while (1) {
    int old_value = atomic_load(&x);
    /* calculation outside of the CS
     * (no side effects, ONLY uses old_value) */
    int new_value = very_expensive_calculation(old_value);

    mutex_lock(&mutex);
    if (old_value == x) {
        /* only write the result if the variable did not change */
        x = new_value;
        mutex_unlock(&mutex);
        break;
    } else {
        /* try again */
        mutex_unlock(&mutex);
    }
}
```

Solution:

There are no race conditions **(0.5 P)***. If another thread changes `x` after the variable has been read in line 2, the check in line 7 detects the concurrent access and discards the result of the calculation instead of writing the result* **(1.0 P)***.*

Wie wirkt sich der Ansatz auf die Performance des Programms aus, wenn `lock` außer `x` keine weiteren Variablen schützt? Begründen Sie Ihre Antwort. **1.5 pt**

How does the approach affect the performance of the program if `lock` does not protect any other variables besides `x`? Justify your answer.

Solution:

Performance is commonly reduced **(0.5 P)** *as any concurrent access to* x *causes the calculation to be performed multiple times* **(1.0 P)**.

Answers which alternatively only argue that the approach improves the performance due to a reduced size of the critical section **(0.5 P)** *and therefore more parallelism for read accesses* **(0.5 P)** *are incomplete.*

Total: 9.0pt

Aufgabe T5: I/O, Hintergrundspeicher und Dateisysteme

Assignment T5: I/O, Secondary Storage, and File Systems

a) Nehmen Sie an, der Pfadname `/a/b/c/orig.txt` verweist auf eine normale Datei. Auf wie viele Ordner greift das Betriebssystem beim Öffnen dieser Datei zu? **0.5 pt**

Assume you have a regular file that is referred to by the pathname `/a/b/c/orig.txt`. How many directories will the operating system access when opening this file?

Solution:

4 directories: root, a, b, c

Gehen Sie nun davon aus, dass wir einen Hardlink sowie einen Symlink auf diese Datei wie unten angegeben anlegen. Auf wie viele Ordner greift das Betriebssystem beim Öffnen von `/hard.txt` bzw. von `/a/b/c/soft.txt` zu? **1 pt**

Now assume we create a hard link and a symlink to this file, as given below. How many directories will the operating system access when opening `/hard.txt` or `/a/b/c/soft.txt`?

```
$ ln /a/b/c/orig.txt /hard.txt
```

Solution:

1 directory: root **(0.5 P)**

```
$ ln -s /a/b/c/orig.txt /a/b/c/soft.txt
```

Solution:

8 directories: root, a, b, c for soft.txt, then root, a, b, c for orig.txt **(0.5 P)**

`orig.txt` wird gelöscht. Können wir weiterhin auf `hard.txt` zugreifen? Erklären Sie. **1 pt**

`orig.txt` is deleted. Can we still access `hard.txt`? Explain.

Solution:

A hard link is a separate directory entry that refers to the same inode **(0.5 P)**. *Deleting the directory entry `orig.txt` thus does not affect `hard.txt` and we can still access it* **(0.5 P)**.

b) Geben Sie zwei Beispiele von Dateitypen an, die üblicherweise in der Inode kodiert sind. **1 pt**

Give two examples for file types that are commonly encoded in the inode.

Solution:

regular file, directory, symbolic link, character/block device

Wie erkennt ein Unix-System ein Shellskript in einer ausführbaren Datei? **1 pt**

How does a Unix system detect a shell script in an executable file?

Solution:

By its content **(0.5 P)**: *A shell script starts with a shebang `#!`* **(0.5 P)**.

c) Füllen Sie die Lücken in dem untenstehenden Text aus. **2.5 pt**

Fill in the gaps in the text below.

Solution:

(0.5 P) *per gap*

__Chained Allocation__ ist eine **Dateiallokationsstrategie** bei der jeder Datenträgerblock einen Pointer zu dem nächsten Block der Datei enthält. Sie erreicht gute Performance für __sequenzielle__ Zugriffe, ist aber sehr langsam bei __wahlfreien__ Zugriffen. __FAT / Linked List Alloc.__ ist eine verbesserte Strategie, die alle Pointer in __den Hauptspeicher__ lädt.

__*Chained allocation*__ *is a* ***file allocation strategy*** *where each disk block contains a pointer to the next block in the file. It provides good performace for* _____ __*sequential*__ *accesses, but is very slow for* __*random*__ *accesses.* __*FAT / linked list alloc.*__ *is an improved strategy that loads all the pointers in* __*main memory*__*.*

d) Gegeben seien ein RAID-0-System mit 4 Festplatten sowie ein RAID-4-System mit 5 Festplatten. Welchen Nachteil hat das RAID-4-System? Erklären Sie. **1 pt**

Assume you have a RAID-0 system with 4 disks and a RAID-4 system with 5 disks. What disadvantage does the RAID-4 system have? Explain.

Solution:

The RAID-4 systems does not allow parallel ***write*** *accesses* **(0.5 P)** *because the parity disk forms a bottleneck* **(0.5 P)**.

Ein Programm schreibt auf 12 zufällige Blöcke. Wie lange muss es jeweils auf dem RAID-0 bzw. RAID-4-System insgesamt auf Festplattenzugriffe warten? Gehen Sie davon aus, dass die Schreibvorgänge soweit wie möglich gleichmäßig auf den Festplatten verteit sind. Jeder Lese- oder Schreibzugriff auf eine Festplatte benötigt D Zeiteinheiten. **1 pt**

A program writes to 12 random blocks. How long in total does it need to wait for disk accesses on the RAID-0 and RAID-4 system, respectively? Assume that these random writes are spread out evenly as much as possible across the disks. Every read or write access to a disk takes D time units.

Solution:

RAID-0: 12 writes distributed to 4 disks $\Rightarrow \frac{12}{4}D = 3D$ **(0.5 P)**

RAID-4: For every write, we need to read and write the parity block from the same disk (bottleneck!) $\Rightarrow 12 \times 2D = 24D$ **(0.5 P)**

Total: 9.0pt

Aufgabe P1: C Grundlagen

Assignment P1: C Basics

a) In dem untenstehenden Code haben sich 7 Fehler eingeschlichen. Markieren Sie die fehlerhaften Zeilen mit einem X und korrigieren Sie den Code. Gehen Sie von einem 64-Bit-System aus. **7 pt**

There are 7 errors in the code below. Mark the incorrect lines with an X and correct the code. Assume a 64-bit system.

```
#define CHUNK_SIZE 14
struct chunk {
    int32_t data[CHUNK_SIZE];
    struct chunk *prev;
};
```

```
struct head {
    size_t len;
    struct chunk *tail;
};
```

Solution:

```
int push(struct head *head, int32_t data) {
    struct chunk *tail = head->tail;
    if (head->len % CHUNK_SIZE == 0) {
        tail = malloc(sizeof(struct chunk)); /* mistake: missing 'struct' */
        if (tail == NULL) return -1;
        tail->prev = head->tail; /* mistake: tail.prev */
        head->tail = tail;
    }
    tail->data[head->len++ % CHUNK_SIZE] = data;
    return 0;
}

int pop(struct head *head, int32_t *data) {
    struct chunk *tail = head->tail;
    if (head->len == 0) return -1;
    *data = tail->data[--head->len % CHUNK_SIZE]; /* mistake: head->len-- */
    if (head->len % CHUNK_SIZE == 0) {
        head->tail = tail->prev; /* mistake: use-after-free */
        free(tail);
    }
    return 0;
}

int swap(struct head *head) {
    if (head->len < 2) return -1;
    struct chunk *prev = head->tail;
    size_t i = (head->len - 1) % CHUNK_SIZE;
    size_t j = (head->len - 2) % CHUNK_SIZE;
    if (i == 0) prev = prev->prev; /* mistake: i == 1 */
    int32_t tmp = head->tail->data[i]; /* mistake: int16_t too small */
    head->tail->data[i] = prev->data[j]; /* mistake: i and j swapped */
    prev->data[j] = tmp;
    return 0;
}
```

Welche Datenstruktur implementiert der Code? **0.5 pt**

Which data structure does the code implement?

Solution:

The code implements a stack (or LIFO).

Was gibt das Programm (nach Korrektur aller Fehler) bei Ausführung der untenstehenden main-Funktion aus? **1 pt**

What does the program (after correcting all errors) print when executing the main function below?

```
int main() {
    struct head head = {0};
    int32_t i, x;
    for (i = 0; i < 5; i++) {
        push(&head, i);
        swap(&head);
    }
    while (pop(&head, &x) != -1)
        printf("%" PRId32 "␣", x);
    printf("\n");
}
```

Solution:

0 4 3 2 1

Warum ist die `CHUNK_SIZE` (= 14) sinnvoll gewählt? **1 pt**

Why is the `CHUNK_SIZE` (= 14) sensibly chosen?

Solution:

With `CHUNK_SIZE` = 14, `struct chunk` has size $14 \times 4 + 8 = 64$ *bytes* **(0.5 P)**, *which is exactly one cache line* **(0.5 P)**.

b) Was ist der Wert der Ausdrücke nach Ausführung des Programmschnipsels? **2 pt**

What is the value of the expressions after execution of the given code snippet?

```
uint32_t a[] =
  { 0x00112233, 0x44556677,
    0x8899aabb, 0xccddeeff,
    0xff00ff00, 0x11221100 };

uint16_t b = a[0] + a[1];
uint32_t *p0 = a;
uint32_t *p1 = 2 + p0;
// <-- Evaluate expression here
```

Solution:

Expression	*Value*
`a[3] & a[4]`	*0xcc00ee00*
`~a[5] >> 16`	*0xeedd*
*b	10*	*0x88aa*
**p1*	*0x8899aabb*

c) Ergänzen Sie die Abbruchbedingung der Schleife unten, sodass die Funktion die Länge des Strings `str` berechnet. Lassen Sie den übrigen Code unverändert. **1 pt**

Complete the condition of the loop below so that the function calculates the length of the string `str`. Do not modify other parts of the code.

Solution:

```
size_t my_strlen(char *str) {
  size_t len = 0;

  while (*str++)
    len++;
  return len;
}
```

d) Die Funktion `g()` des untenstehenden C-Programms wird zu der Assembly rechts kompiliert. Ergänzen Sie die `push`-Instruktionen, sodass die Parameter gemäß cdecl übergeben werden. **0.5 pt**

The function `g()` in the C program below compiles to the assembly on the right. Complete the `push` instructions so that the code passes the parameters according to cdecl.

main.c

```
int f(int, int);

int g() {
  return f(3, 5);
}
```

main.S

Solution:

```
g:
        push    5
        push    3
        call    f
        add     esp, 8
        ret
```

Auf welchem Weg wird das `int`-Ergebnis zurückgegeben? **0.5 pt**

How is the `int` result returned?

Solution:

The `int` result is returned in the `eax` register.

e) Geben Sie für alle Felder des unten stehenden `struct mystruct` die Größe des Feldes und die Größe des Paddings *nach* dem Feld in Bytes an. Schreiben Sie „0“, falls kein Padding eingefügt wird. Gehen Sie von einem 32-Bit-System aus. **1.5 pt**

For each field of the `struct mystruct` below, give the field's size and the size of the padding after *the field in bytes. Write "0" if the compiler does not insert any padding. Assume a 32-bit system.*

Code	Field size [Bytes]	Padding size [Bytes]
`struct mystruct {`	—	—
`  int64_t a;`	8	0
`  char b;`	1	3
`};`	—	—

(0.5 P) for two field sizes, **(0.5 P)** per padding size

Total: 15.0pt

Aufgabe P2: Build-System

Assignment P2: Build System

Sie sollen eine stark vereinfachte Version des `make`-Tools zum Übersetzen mehrerer Quellcodedateien schreiben. Dieses Programm nimmt als ersten Kommandozeilenparameter eine Datei, die eine Liste von Pfaden wie die folgende enthält:

```
file1.c
file1.o
file2.c
file2.o
```

Das Programm soll für je zwei aufeinanderfolgende Zeilen prüfen, ob die erste Datei neuer ist als die zweite. Falls ja, oder falls die zweite Datei nicht existiert, soll es `gcc` so ausführen, dass die erste Datei in eine Objektdatei übersetzt und an dem zweiten Pfad gespeichert wird:

```
gcc -c file1.c -o file1.o    # Falls file1.c neuer als file1.o
gcc -c file2.c -o file2.o    # Falls file2.c neuer als file2.o
```

- Sie müssen keine C-Header inkludieren.
- Sie müssen nur Fehlerbehandlung implementieren, wenn es explizit gefordert wird.
- Geben Sie im Hauptprozess jegliche angeforderten Ressourcen wieder frei. In Kindprozessen müssen Sie keine Ressourcen freigeben.

You shall write a vastly simplified version of the `make` tool which compiles multiple source code files. As its first command line parameter, this program takes a file which contains a list of paths such as the following:

```
file1.c
file1.o
file2.c
file2.o
```

For each pair of files, the program shall check whether the first file is newer than the second. If so, or if the second file does not exist, the program shall execute `gcc` to compile the first file and to store the resulting object file at the second path:

```
gcc -c file1.c -o file1.o    # If file1.c is newer than file1.o
gcc -c file2.c -o file2.o    # If file2.c is newer than file2.o
```

- *You do not need to include any C headers.*
- *You only have to implement error handling if doing so is explicitly requested.*
- *Free all resources allocated in the main process. You do not need to free resources in child processes.*

```
/* no lines are longer than the following constant (including '\n'): */
#define PATH_MAX 4096 /* maximum path length (including '\0') on Linux */
```

a) Vervollständigen Sie die Funktion `need_rebuild()`, die prüft, ob die zweite Datei (`path_b`) neu aus der ersten (`path_a`) erstellt werden soll. **3 pt**

- Vergleichen Sie die Zeiten der letzten Änderung. Ist die letzte Änderung von `path_a` jünger als die von `path_b`, so soll die Funktion 1 zurückgeben, ansonsten 0.
- Falls die Datei `path_b` nicht existiert oder die Änderungszeit von `path_b` aus einem anderen Grund nicht ermittelt werden kann, soll die Funktion ebenfalls 1 zurückgeben.

Complete the function `need_rebuild()` which checks whether the second file (`path_b`) shall be rebuilt from the first file (`path_a`).

- *Compare the time of the last modification. If the last modification to `path_a` is more recent than that of `path_b`, the function shall return 1, else it shall return 0.*
- *If the file `path_b` does not exist or the modification time cannot be determined for some other reason, the function shall also return 1.*

Solution:

```
int need_rebuild(const char *path_a, const char *path_b) {
        int status;
        struct stat a, b;
        stat(path_a, &a);
        status = stat(path_b, &b);
        if (status != 0) {
                return 1;
        }
        return a.st_mtime > b.st_mtime;
}
```

Call `stat()` twice with correct parameters **(1.5 P)**, *compare the modification time* **(1.0 P)**, *return 1 on error* **(0.5 P)**.

b) Vervollständigen Sie die Funktion `run_gcc()`, die den Befehl `gcc -c <input> -o <output>` ausführt, wobei `<input>` und `<output>` durch die jeweilgen Funktionsparameter ersetzt werden. Die Funktion soll erst zurückkehren, wenn der Kindprozess beendet wurde. **4 pt**

- Verwenden Sie `fork()` und einen der `exec`-Systemaufrufe.

Complete the function `run_gcc()` which executes the command `gcc -c <input> -o <output>`, where `<input>` and `<output>` are replaced by the respective function parameters. The function shall only return once the child process has completed.

- *Use `fork()` and one of the `exec` system calls.*

Solution:

```
void run_gcc(const char *input, const char *output) {
        int pid = fork();
        if (pid == 0) {
                execlp("gcc", "gcc", "-c", input, "-o", output, NULL);
        } else {
                waitpid(pid, NULL, 0);
        }
}
```

Call `fork()` to create a child process **(0.5 P)**, *call a suitable variant of `exec`* **(0.5 P)** *in the child process* **(0.5 P)**, *specify the correct `argv[0]`* **(0.5 P)** *and correct values for the remaining command line arguments* **(0.5 P)**, *terminate the argument list with `NULL` if necessary* **(0.5 P)**, *and wait for the process* **(1.0 P)**.

The code needs to call either `execlp()` or `execvp()` as the operating system has to search for `gcc` in the configured search paths for executables. `execl()` or `execp()` are therefore incorrect.

c) Die Dateipfade sollen aus einer Datei gelesen werden. Vervollständigen Sie die Funktion `read_line()`, die so lange Daten aus dem Dateideskriptor `file` in den Puffer `buffer` liest, bis sie auf einen Zeilenumbruch trifft. **4.5 pt**

- Der Zeilenumbruch soll nicht nach `buffer` kopiert werden.
- `buffer` soll durch die Funktion immer nullterminiert werden.
- Der Puffer hat die Länge `buf_len`. Passt die Zeile nicht in den Puffer, so soll die Funktion eine Fehlermeldung ausgeben und das Programm unmittelbar mit `exit()` beenden.
- Wenn die Funktion auf das Ende der Datei trifft, soll sie 0 zurückgeben. Ansonsten soll sie 1 zurückgeben, um gültige Daten in `buffer` zu signalisieren. Sie dürfen davon ausgehen, dass die Datei mit einem Zeilenumbruch endet, wie es vom POSIX-Standard vorgeschrieben wird.

The file paths shall be read from a file. Complete the function `read_line()` which reads data from the file descriptor `file` into the buffer `buffer` until it hits a line break.

- *The line break shall not be copied to `buffer`.*
- *`buffer` shall always be null-terminated by the function.*
- *The buffer has the length `buf_len`. If the line does not fit into the buffer, the function shall print an error message and shall immediately quit the program via `exit()`.*
- *If the function meets the end of the file, it shall return 0. Else, it shall return 1 to signal that `buffer` holds valid data. You may assume that the file ends with a newline character as stipulated by the POSIX standard.*

Solution:

```
int read_line(int file, char *buffer, size_t buf_len) {
    size_t total = 0;
    for (total = 0; total < buf_len; total++) {
        ssize_t bytes_read = read(file, &buffer[total], 1);
        if (bytes_read == 0) {
            /* not required, the POSIX standard says that files end with a
             * newline character - a simple "return 0" is enough here */
            buffer[total] = 0;
            return total != 0;
        }
        if (buffer[total] == '\n') {
            buffer[total] = 0;
            return 1;
        }
    }
    /* no '\n' found, line too long */
    printf("line too long\n");
    exit(-1);
}
```

Read bytes **(1.0 P)** *until there is a newline character* **(0.5 P)**, *zero-terminate the buffer* **(0.5 P)**, *return 1 if a newline character is found* **(0.5 P)** *and 0 if the end of the file has been reached* **(0.5 P)**. *Stop when the end of the buffer is reached* **(0.5 P)**, *print an error message* **(0.5 P)** *and stop the program* **(0.5 P)**.

d) Vervollständigen Sie die `main()`-Funktion des Programms, die die von `fd` referenzierte Datei zeilenweise einliest. **3.5 pt**

- Nachdem je zwei Zeilen eingelesen wurden, soll die Funktion die durch die zweite Zeile bestimmte Datei aus der durch die erste Zeile bestimmte Datei neu erstellen, falls dies wie oben beschrieben notwendig ist.
- Unter Linux bezeichnet `PATH_MAX` die maximale Länge von Pfaden in Byte, inklusive abschließendem Nullbyte. Ihr Programm soll Zeilen dieser Länge einlesen können.

Complete the `main()` *function of the program which reads the file specified by* `fd` *line by line.*

- *Whenever two lines have been read, the function shall rebuild the file specified by the second line from the file specified by the first line if doing so is necessary as described above.*
- *On Linux,* `PATH_MAX` *specifies the maximum length of paths in bytes, including the terminating null byte. Your program shall be able to read lines of this length.*

Solution:

```
int main(int argc, char **argv) {
        const char *list_file = argv[1];
        int fd = open(list_file, O_RDONLY);

        char input_buf[PATH_MAX];
        char output_buf[PATH_MAX];

        while (1) {
                if (read_line(fd, input_buf, PATH_MAX) == 0) {
                        break;
                }
                if (read_line(fd, output_buf, PATH_MAX) == 0) {
                        break;
                }
                if (need_rebuild(input_buf, output_buf)) {
                        run_gcc(input_buf, output_buf);
                }
        }

        close(fd);
        return 0;
}
```

Allocate two buffers for two lines **(0.5 P)**, *loop until the end of the file* **(1.0 P)**, *read the two lines* **(1.0 P)**, *rebuild the file* **(0.5 P)** *if necessary* **(0.5 P)**. **Total: 15.0pt**

Aufgabe P3: Speicherallokator

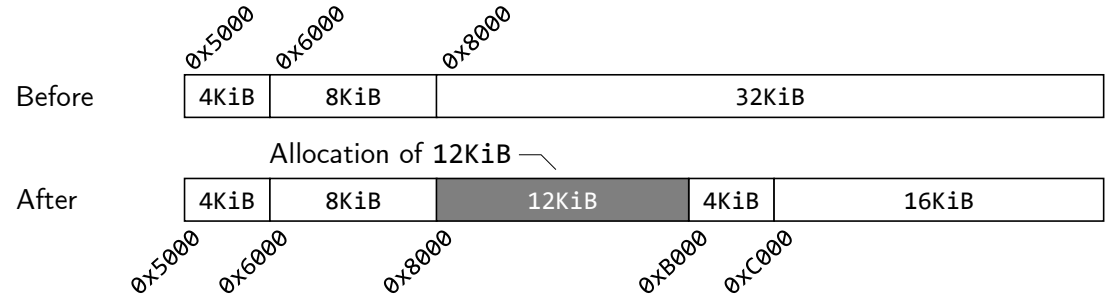
Assignment P3: Memory Allocator

Im Folgenden entwickeln Sie einen Speicherallokator. Der Allokator hält Listen freier Speicherblöcke vor (`free_list[]`), wobei alle Blöcke der Liste mit Index l die Größe 2^{l+12} haben und ihre Startadresse an ihrer Größe ausgerichtet ist. Die minimale Allokationsgröße entspricht demnach 4 KiB.

Bei einer Speicheranfrage sucht der Allokator nach einem möglichst kleinen freien Block, der die Anfrage erfüllt. Ist der gefundene Block größer als die angefragte Speichermenge, wird der überschüssige Platz als freier Speicher den entsprechenden Listen zugewiesen. Die Abbildung zeigt den Zustand vor und nach einer Allokation von 12 KiB. Der Allokator nimmt 12 KiB vom Anfang des 32 KiB-Blocks. Der restliche Speicher des 32 KiB-Blocks wird gemäß der Größen- und Ausrichtungskriterien aufgeteilt und an die Listen freier Blöcke zurückgegeben.

In the following, you develop a memory allocator. The allocator holds lists of free memory blocks (`free_list[]`), where all blocks in the list with index l have the size 2^{l+12} and their starting address is aligned with their size. Thus, the minimum allocation size is equal to 4 KiB.

Given a memory request, the allocator searches for the smallest possible free block that satisfies the request. If the block found is larger than the requested amount of memory, the excess space is assigned as free memory to the corresponding lists. The figure shows the state before and after an allocation of 12 KiB. The allocator takes 12 KiB from the beginning of the 32 KiB block. The remaining space of the 32 KiB block is split according to the size and alignment constraints and returned to the free lists.




```c
// Metadata for free block.
// Stored in the free memory itself at the beginning of each block.
typedef struct block {
    struct block *next;   // Next free block of same size. NULL for last.
    size_t size;          // Size of block in bytes.
} block;

#define MIN_ORDER 12                    // OTS(MIN_ORDER) = 4 KiB
#define MAX_ORDER 30                    // OTS(MAX_ORDER) = 1 GiB

#define OTL(order) (order - MIN_ORDER)  // Order to index into free list.
#define OTS(order) (1ULL << order)      // Order to size in bytes.

block *free_list[OTL(MAX_ORDER)]; // Lists of free blocks. NULL if empty.
```

a) Vervollständigen Sie die Funktion `largest_order()`, welche die größte Zweierpotenz berechnet, an der die Adresse `base` ausgerichtet ist und die kleiner oder gleich der Länge `len` ist. Die Funktion gibt den Wert des Exponenten (Order) zurück. **3.5 pt**

- Gehen Sie davon aus, dass für alle erwartbaren Eingabewerte gilt, dass der Exponent im Intervall $[\texttt{MIN\_ORDER}, \texttt{MAX\_ORDER})$ liegt.

Complete the function `largest_order()` which calculates the largest power of two to which the address `base` is aligned and which is smaller or equal the length `len`. The function returns the exponent (order).

- *Assume that for every input value to be expected the exponent is in the interval $[\texttt{MIN\_ORDER}, \texttt{MAX\_ORDER})$.*

Solution:

```
unsigned int largest_order(uintptr_t base, size_t len) {
    unsigned int o = MIN_ORDER;

    while ((base & (1UL << o)) == 0)
        o++;

    while (OTS(o) > len)
        o--;

    return o;
}
```

Introduce variable and set sensible start value **(0.5 P)**, *calculate order based on `base`* **(1.5 P)**, *calculate order based on `len`* **(1.5 P)**

b) Vervollständigen Sie die Funktion `link()`, welche den freien Speicherbereich an der Adresse `base` mit der Länge $2^{\texttt{order}}$ an den Anfang der entsprechende Liste freier Blöcke (`free_list[]`) einhängt. **2.5 pt**

- Gehen Sie davon aus, dass `order` im Intervall $[\texttt{MIN\_ORDER}, \texttt{MAX\_ORDER})$ liegt.
- Legen Sie die Struktur `block` an der Adresse `base` ab und initialisieren Sie alle Felder.

Complete the function `link()` which appends the free memory area starting at address `base` and with length $2^{\texttt{order}}$ to the beginning of the corresponding free list (`free_list[]`).

- *Assume that `order` is in the interval $[\texttt{MIN\_ORDER}, \texttt{MAX\_ORDER})$.*
- *Store the structure `block` at the address `base` and initialize all fields.*

Solution:

```
void link(uintptr_t base, unsigned int order)
{
    unsigned int l = OTL(order);
    block *b = (block *)base;

    b->next = free_list[l];
    b->size = OTS(order);
    free_list[l] = b;
}
```

(0.5 P) *for every (equivalent) statement*

c) Vervollständigen Sie die Funktion `free()`, welche den freien Speicherbereich an der Adresse `base` mit der Länge `len` derart aufteilt, dass alle entstehenden Blöcke an ihrer Länge ausgerichtet sind und die Anzahl der Blöcke minimal ist - also möglichst große Blöcke entstehen (siehe Abbildung auf Seite 22). Die entstehenden Blöcke sollen der passenden Liste freier Blöcke hinzugefügt werden. **2.5 pt**

- Gehen Sie davon aus, dass `len` ein Vielfaches von 4 KiB ist und `base` mindestens an 4 KiB ausgerichtet ist.
- Nutzen Sie die Funktionen `largest_order()` und `link()`.

Complete the function `free()` which splits the free memory area starting at address `base` and with length `len` so that all resulting blocks are aligned to their size and the number of blocks is minimal - that is the blocks are as large as possible (see the figure on page 22). The resulting blocks shall be added to the corresponding free lists.

- *Assume that `len` is a multiple of 4 KiB and that `base` is at least aligned to 4 KiB.*
- *Use the functions `largest_order()` and `link()`.*

Solution:

```
unsigned int largest_order(uintptr_t base, size_t len);
void link(uintptr_t base, unsigned int order);

void free(uintptr_t base, size_t len)
{
    while (len > 0) {
        int o = largest_order(base, len);
        size_t s = OTS(o);

        link(base, o);

        base += s;
        len -= s;
    }
}
```

(0.5 P) *for every (equivalent) statement*

d) Vervollständigen Sie die Funktion `malloc()`, welche einen Speicherbereich der Länge `len` alloziert und einen Zeiger darauf zurückgibt oder `NULL`, wenn die Anfrage nicht erfüllt werden kann. **6.5 pt**

- Gehen Sie davon aus, dass `len` ein Vielfaches von 4 KiB ist und $\texttt{len} < 2^{\texttt{MAX\_ORDER}}$.
- Die Funktion wählt den ersten freien Block aus der nicht leeren Liste freier Blöcke mit der kleinstmöglichen Größe, um die Anfrage zu erfüllen.
- Nutzen Sie zum Aushängen die Funktion `try_unlink()`, welche `NULL` zurückgibt, wenn die Liste leer ist.
- Die Blockstruktur liegt direkt am Beginn innerhalb des ausgewählten freien Speicherbereichs selbst.
- Rufen Sie die Funktion `free()` auf, um überschüssigen Platz am Ende des Blocks wieder freizugeben.

Complete the function `malloc()` *which allocates a memory area of length* `len` *and returns a pointer to it or* `NULL` *if the request cannot be satisfied.*

- *Assume that* `len` *is a multiple of 4 KiB and* `len` $< 2^{MAX\_ORDER}$.
- *The function selects the first free block from the non-empty free list with the smallest block size to statisfy the request.*
- *To unlink a block use* `try_unlink()` *which returns* `NULL` *if the list is empty.*
- *The block structure is stored directly at the beginning within the selected free memory area itself.*
- *Call* `free()` *to release excess space at the end of the block.*

Solution:

```
block *try_unlink(unsigned int order);
void free(uintptr_t base, size_t len);

void *malloc(size_t len) {
    block *b;
    unsigned int o;

    for (o = MIN_ORDER; o < MAX_ORDER; o++) {
            if (OTS(o) < len)
                    continue;

            b = try_unlink(o);
            if (b)
                    break;
    }

    if (b == NULL)
            return NULL;

    if (b->size > len)  /* optional */
            free((uintptr_t)b + len, b->size - len);

    return b;
}
```

Iterate over orders/levels to find out suitable order/level **(1.5 P)**, *ensure that order/level is large enough to hold* `len` *bytes* **(1.0 P)**, *try to take a block from the free list and cancel search if successful* **(1.5 P)**, *return* `NULL` *if no block can be found* **(0.5 P)**, *call* `free` *for the unused memory (free call* **(0.5 P)**, *cast of* `b` **(0.5 P)**, *first argument* **(0.5 P)**, *second argument* **(0.5 P)**)

Total: 15.0pt