Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Praktikum Systemprogrammierung

Versuch 3

Testtaskbeschreibung

Lehrstuhl Informatik 11 - RWTH Aachen

17. April 2020

Inhaltsverzeichnis

Dieses Dokument ist Teil der begleitenden Unterlagen zum *Praktikum Systemprogrammierung*. Alle zu diesem Praktikum benötigten Unterlagen stehen im Moodle-Lernraum unter https://moodle.rwth-aachen.de zum Download bereit. Folgende E-Mail-Adresse ist für Kritik, Anregungen oder Verbesserungsvorschläge verfügbar:

support.psp@embedded.rwth-aachen.de

3 Testtaskbeschreibung

3.1 Schedulingstrategien

Mithilfe des Testtasks *Schedulingstrategien* können alle vorgegebenen Schedulingstrategien überprüft werden. Dazu werden diverse Prozesse gestartet und das korrekte Scheduling in vier Phasen überprüft:

Phase 1

In der ersten Phase werden die Prozess-IDs der Prozesse in der Reihenfolge ausgegeben, in welcher sie vom Scheduler Rechenzeit zugewiesen bekommen. Bei korrekter Implementierung der Schedulingstrategien ergibt sich beim Durchlauf der ersten Phase des Testtasks folgende Ausgabe, gefolgt von einem `„OK"`:

Even:	`12312312312312312312312312312`
Random: (Beispiel)	`13232322232223123133223331131321`
RoundRobin:	`11222223333333333333333311222223`
InactiveAging:	`13332333231323332331323332331323`
RunToCompletion:	`11111111111111111111111111111111`

Wird von einer Schedulingstrategie Prozess 0 ausgewählt, erscheint die Meldung `„Not impl. or idle returned"` und es wird mit der nächsten Strategie fortgefahren. Man beachte, dass die Strategien im Codegerüst, sofern nicht geändert, dieses Verhalten aufweisen. Wird ein Fehler erkannt, bleibt die Ausgabe sichtbar und die erste falsche Stelle wird markiert.

Phase 2
In Phase 2 wird geprüft, ob die Scheduling Strategien den Leerlaufprozess auswählen, wenn kein anderer Prozess ausgeführt werden kann. Es wird ggf. eine Fehlermeldung mit der entsprechenden Strategie ausgeben und die weitere Ausführung unterbrochen.

Phase 3
In der dritten Phase wird geprüft, ob alle Prozesse außer dem Leerlaufprozess mindestens ein mal von jeder Strategie ausgewählt werden. Da die Prozesse in diesem Testtask nicht terminieren, ist RunToCompletion von dieser Phase ausgenommen.

Phase 4
Diese Phase testet analog zur vorherigen, allerdings sind hier nicht alle Process Slots belegt.

Sollen nur bestimmte Strategien getestet werden, können die anderen dementsprechend in dem Array `strategies` in der Testdatei auskommentiert werden.

Fehlermeldungen

STRAT: x not schedulable
In der Strategie `STRAT` wurde Prozessslot x nicht ausgewählt.
STRAT: Idle not scheduled
In der Strategie `STRAT` wird der Leerlaufprozess nicht ausgewählt.

3.2 FreeMap

Der Testtask *FreeMap* überprüft, ob wichtige Eigenschaften des internen Heaps korrekt initialisiert wurden und, ob die Freigabe von alloziertem Speicher dem im Versuchsdokument beschriebenen Protokoll folgt.

Der gesamte Testtask gliedert sich in drei Phasen. In der ersten Phase von *FreeMap* findet eine Plausibilitätskontrolle des internen Speichertreibers statt. Es wird sowohl das 1:2 Verhältnis von Map- zu Use-Bereich kontrolliert, als auch die Allokationstabelle auf eine entsprechende Mindestgröße überprüft. Hier wird angenommen, dass die Größe der Allokationstabelle mindestens 100 Bytes beträgt. Des Weiteren werden die Rückgabewerte der Funktionen `os_lookupHeap` und `os_getHeapListLength` kontrolliert. Treten in dieser Phase Fehler auf, wird eine Fehlerübersicht in Tabellenform angezeigt, die angibt, welche Überprüfungen nicht erfolgreich waren. Hier sei neben der Auflistung unten auf die Kommentare innerhalb des Testtasks verwiesen.
In der zweiten Phase des Testtasks wird mit Hilfe von `os_malloc` die ganze Allokationstabelle belegt. Anschließend wird mit der unteren Treiberstruktur überprüft, ob die Speicherallokation gemäß Protokoll implementiert wurde. Nach Freigabe des Bereichs mittels `os_free` folgt eine erneute Überprüfung auf Korrektheit der Allokationstabelle. Treten hierbei Fehler auf, wird wieder eine Fehlerübersicht in Tabellenform angezeigt.
In der dritten Phase wird manuell das Ende der Allokationstabelle belegt und zusätzlich der Anfang des Use-Bereichs so beschrieben, dass insgesamt der Eindruck entsteht, als existiere ein allozierter Speicherbereich, welcher über die Allokationstabelle hinaus geht. Nach anschließender Freigabe mittels `os_free` wird überprüft ob der Speicherbereich in der Allokationstabelle freigegeben und die Nutzdaten nicht überschrieben wurden. Treten hierbei Fehler auf, wird wieder eine Fehlerübersicht in Tabellenform angezeigt.
Nachdem alle Phasen erfolgreich ausgeführt wurden, zeigt das LCD „`ALL TESTS PASSED`" an und der Testtask terminiert nachdem der Benutzer mit einem beliebigen Tastendruck bestätigt.

Fehlermeldungen

Fehlermeldungen Phase 1
Hier werden Fehler in einer Tabelle angegeben. Es gibt hierbei vier Spalten mit je einem

Titel und einem Testergebnis. Das Ergebnis kann entweder `ERR` oder `OK` lauten. Im ersten Fall ist der entsprechende Test fehlgeschlagen. Die vier Spalten sind:
`DRV:` `OK`, wenn der Treiber korrekt von `os_lookupHeap` zurückgegeben wird.
`MAP:` `OK`, wenn die Größe der Map plausibel ist.
`LST:` `OK`, wenn die Heap-Liste die korrekte Länge hat
`1:2:` `OK`, wenn das Verhältnis von Map- zu Use-Bereich korrekt ist

Fehlermeldungen Phase 2
Hier gibt es wieder vier Spalten. Diese sind:
`MAL:` `OK`, wenn das Allozieren des gesamten Use-Bereichs erfolgreich war. Ist das Ergebnis hier `ERR`, so werden die anderen Überprüfungen nicht durchgeführt und haben demnach keine Aussagekraft.
`OWN:` `OK`, wenn der Speicherblock den korrekten Besitzer hat.
`FIL:` `OK`, wenn die gesamte Map die korrekten Werte hat.
`FRE:` `OK`, wenn das Freigeben des Speicherblocks korrekt in der Map vermerkt wird.

Fehlermeldungen Phase 3
Hier gibt es nur zwei Spalten mit je einem Titel und einem Testergebnis. Diese sind:
`MAP:` `OK`, wenn die Map durch `os_free` korrekt verändert wurde.
`USE:` `OK`, wenn der Use-Bereich durch `os_free` nicht verändert wurde.

MAL	OWN	FIL	FRE
OK	ERR	OK	ERR

Abbildung 3.1: Es gab einen Fehler beim Setzen des Besitzers des Speicherblocks und beim wieder-freigeben des selbigen.

Achtung

Ein Testtask, welcher zuletzt „`WAITING FOR TERMINATION "` anzeigt, gilt erst als bestanden, wenn im Anschluss der Leerlaufprozess angezeigt wird.

3.3 Stability Private

Der Testtask *Stability Private* überprüft, ob die Allokation von dynamischen Speicherbereichen in beliebiger Größe möglich ist und ob geschriebene Datenmuster zu einem späteren Zeitpunkt korrekt ausgelesen werden können.

Bei einer fehlerfreien Implementierung wird in der oberen Zeile des LCD die bisher verstrichene Zeit und in der unteren Zeile fortlaufend „`1a 1b 1c 1d 2a 2b 2c 2d 3a 3b...`" ausgegeben. Dabei gibt die Ziffer den Prozess und der Buchstabe die aktuelle Phase des Testtasks an, in der man sich befindet. Folglich müssen die Buchstaben a-d sowie die Zahlen 1-3 auftreten. Die Reihenfolge der Zahlen-Buchstaben-Paare ist hierbei irrelevant. In Phase *a* des Testtasks wird Speicher alloziert und überprüft, ob der dabei erhaltene Adressbereich sich noch innerhalb der erlaubten Grenzen befindet. Ist dies nicht der Fall, wird die Fehlermeldung „`Address too small`" bzw. „`Address too large`" ausgegeben. Anschließend werden zuvor ermittelte Werte in den allozierten Speicherbereich geschrieben. In Phase *c* wird überprüft, ob die geschriebenen Werte im Speicherbereich noch mit den Originalwerten übereinstimmen. Ist dies nicht der Fall, wird die Fehlermeldung „`Pattern mismatch`" angezeigt. Freigegeben wird der Speicherbereich in Phase *d*, wohingegen in Phase *b* nur gewartet wird.
Zukünftig werden mit diesem Testtask noch weitere Allokationsstrategien getestet, weshalb im Abstand von einigen Sekunden „`Change to f`" auf dem Display angezeigt und im Anschluss daran zur ursprünglichen Anzeige zurückgekehrt wird. In dieser Version jedoch wird nur die Allokationsstrategie *FirstFit* benutzt.
Dieser Testtask gilt als bestanden, wenn über einen Zeitraum von mindestens drei Minuten keine Fehlermeldung ausgegeben wird.

Fehlermeldungen

Address too small
`os_malloc` hat eine zu kleine Adresse zurückgegeben.

Address too large
`os_malloc` hat eine zu große Adresse zurückgegeben.

Pattern mismatch
Die Daten im allozierten Speicher wurden verändert.

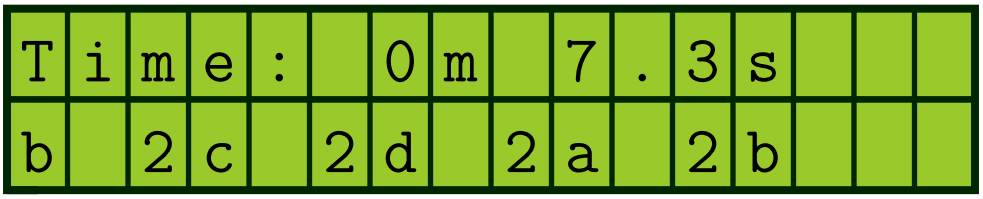

Abbildung 3.2: Durchlauf von Stability Private nach sieben Sekunden

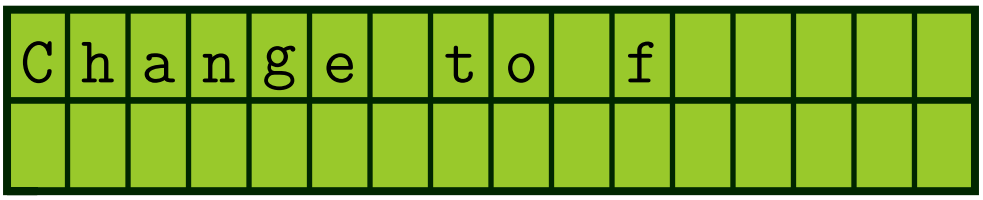

Abbildung 3.3: Wechsel der Schedulingstrategie in Stability Private zu First-Fit

3.4 Termination

Der Testtask *Termination* überprüft, ob das Betriebssystem terminierende Prozesse unterstützt. Voraussetzung für diesen Testtask ist, dass insgesamt mindestens fünf Prozesse gestartet werden können.

Phase 1
In der ersten Phase wird geprüft, ob beim Beenden eines Prozesses fremde kritische Bereiche verlassen werden. Dazu wird ein Programm innerhalb eines kritischen Bereichs ausgeführt und sofort gekillt. Anschließend wird geprüft, ob der kritische Bereich weiterhin offen ist und gegebenenfalls ein Fehler ausgegeben.

Phase 2
In der zweiten Phase werden wiederholt Prozesse erzeugt, die aus einer Programmfunktion bestehen, welche sofort terminiert. Dabei werden in zwei Durchläufen alle Schedulingstrategien (außer *RunToCompletion*) nacheinander geprüft. In der oberen Zeile des LCD wird die aktuelle Anzahl an ausgeführten Prozessen angezeigt. Dieser Wert sollte überwiegend 7/8 betragen. Daneben steht eine Abkürzung für die derzeit genutzte Schedulingstrategie (`EVEN` für *Even*, `RAND` für *Random*, `RORO` für *Round Robin* und `INAG` für *Inactive Aging*). In der unteren Zeile wird die Anzahl an Prozesserzeugungen seit dem letzten Strategiewechsel angezeigt. Diese Zahl sollte sehr schnell ansteigen. Als Richtwert für den Anstieg der *Spawns* kann eine Inkrementierung von 50 Programmaufrufen pro Sekunde herangezogen werden.
Der Testtask beendet erfolgreich, wenn das oben beschriebene Verhalten mit allen vorgestellten Schedulingstrategien (außer *RunToCompletion*) gegeben ist und keine Fehlermeldungen auftraten.

Fehlermeldungen

Max.Num.Proc < 5
Das Betriebssystem lässt nicht mehr als 5 Prozesse zu.

os_exec failed
Beim starten eines neuen Prozesses ist ein Fehler aufgetreten (unerwarteter Rückgabewert von `os_exec`). In der zweiten Zeile des LCDs wird der Rückgabewert von `os_exec` angezeigt.

Left crit. sec.
Während eines Aufrufs von `os_kill` wurden fremde kritische Bereiche verlassen.

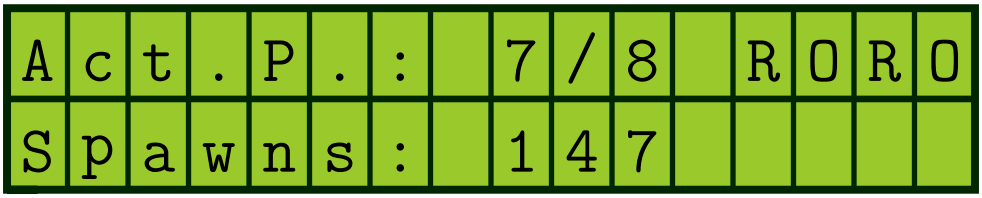

Abbildung 3.4: Termination-Testtask wird ausgeführt und die derzeitige Strategie ist *Round Robin*

3.5 HeapStart

Der Testtask *HeapStart* kontrolliert, ob die Überprüfung des Sicherheitsabstands zwischen Heap und den globalen Variablen in das Betriebssystem integriert wurde.
Hierzu wird eine globale Variable angelegt, die eine große Menge an Speicherplatz belegt und den gesetzten Sicherheitsabstand überschreitet.
Der Test gilt als bestanden, wenn auf dem LCD eine Fehlermeldung erscheint, die angibt, dass der Sicherheitsabstand überschritten wurde.

3.6 Heap Cleanup (in Versuch 3 optional)

Der Testtask *Heap Cleanup* überprüft, ob beim Terminieren von Prozessen der dynamische Speicher korrekt aufgeräumt wird. Dazu wird der Speicher in vier verschiedenen Phasen überprüft. In der ersten Phase wird überprüft, ob mehr Speicher alloziert werden kann, als es laut Allokationstabelle erlaubt ist. Ist dies der Fall, wird die Fehlermeldung „`Overalloc failure in`" gefolgt von dem Namen des entsprechenden Heaps ausgegeben.
In der zweiten Phase wird überprüft, ob der interne Heap korrekt initialisiert wurde. Dabei wird geprüft, ob sich der Use-Bereich des Heaps mit den Prozessstacks überschneidet. Tritt diese Überschneidung auf, so wird die Fehlermeldung „`Internal heap too large`" ausgegeben.
In der dritten Phase wird überprüft, ob es möglich ist, nach dem Aufräumen des Heaps den gesamten zur Verfügung stehenden Speicher zu allozieren. Dies geschieht für alle Memory-Strategien. Dazu wird immer erst ein Prozess gestartet, der Speicher alloziert und nicht mehr freigibt. Dieser Prozess wird dann terminiert und es wird überprüft, ob der gesamte Speicher frei ist. Ist dies nicht der Fall, ist das Aufräumen des Speichers fehlgeschlagen und es wird ein entsprechender Fehler ausgegeben.
In der vierten Phase wird das Aufräumen des Speichers noch tiefer gehend geprüft. Dazu wird durchgehend ein Prozess gestartet, der mehrere kleine Speicherbereiche alloziert

und danach sich selbst terminiert. Die Anzahl dieser Durchläufe wird bei einer fehlerfreien Implementierung als Fortschrittsbalken in der oberen Zeile des LCD verdeutlicht, während in der unteren Zeile ein entsprechender numerischer Wert angezeigt wird. Es wird hier erwartet, dass allozierter Speicher von terminierenden Prozessen freigegeben wird. Alle 50 Schritte wird nach Terminierung aller gestarteten Prozesse überprüft, ob der Speicher freigegeben wurde. Ist dies nicht der Fall, so wird eine Fehlermeldung ausgegeben.
Verlaufen alle Phasen dieses Tests erfolgreich erscheint auf dem LCD die Nachricht „`TESTS PASSED`" sowie „`PLEASE CONFIRM!`". Der Test kann nun mit einem beliebigen Tastendruck beendet werden. Im Anschluss startet der Leerlaufprozess.

Fehlermeldungen

Missing heap!
Es ist nur ein Heap definiert. In Versuch 3 ist diese Fehlermeldung durch das Setzen des Defines `TEST_FOR_ALL_HEAPS` auf `0` zu deaktivieren.

Overalloc failure in (extHeap/intHeap)
Es war möglich, mehr Speicher zu allozieren als im Use-Bereich des entsprechenden Heaps verfügbar ist.

(b/w/f/n):Huge alloc fail in (extHeap/intHeap)
Es war nicht möglich den gesamten Use-Bereich zu allozieren. Der erste Buchstabe dieser Fehlermeldung gibt die Allokationsstrategie an, mit der dies versucht wurde.

Heap not clean: (extHeap/intHeap)
Der Speicher eines terminierenden Prozesses wurde in Phase 4 nicht korrekt freigegeben.

Internal heap too large
Der Use-Bereich des internen Heaps ist so groß, dass er sich mit dem Stack überschneidet.

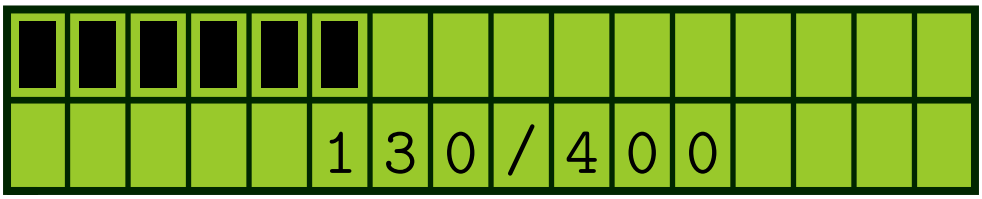

Abbildung 3.5: Heap Cleanup während des Tests.

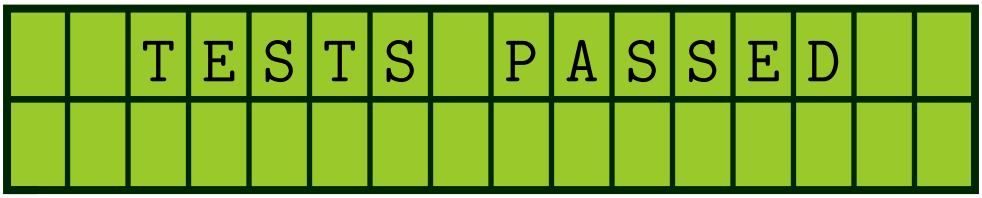

Abbildung 3.6: Anzeige, falls alle Phasen des Tests erfolgreich verliefen.

3.7 Range (in Versuch 3 optional)

Der Testtask *Range* testet eine Kombination aus Funktionalitäten, welche bereits von den beiden Testtasks *Heap Cleanup* und *Termination* getestet wurden. In Ergänzung zu diesen geht *Range* jedoch insbesondere auf die Allokation von Speicherblöcken mit variierender Größenanforderung ein.

Der gesamte Testtask lässt sich in zwei Phasen aufteilen, die mit Hilfe zweier Testprogramme realisiert werden. Dabei funktioniert die erste Phase wie folgt: Bei jedem Durchlauf wird eine bestimmte Menge an Speicherbereichen alloziert, die jeweils die gleiche Größe besitzen. Dabei verringert sich bei jedem weiteren Durchlauf die Menge an Speicherbereichen, während die Größe der Speicherbereiche steigt. Die gesamte Menge des allozierten Speichers ist so bei jedem Durchlauf identisch. So werden zum Beispiel beim ersten Durchlauf 100 Speicherbereiche der Größe 1 alloziert, beim nächsten Durchlauf dann 50 Speicherbereiche der Größe 2. Am Ende wird folglich ein Speicherbereich der Größe 100 alloziert. Ein Fortschrittsbalken in der oberen Zeile des LCD gibt dabei den Fortschritt der aktuellen Allokation an. Mit einem steigenden Index in der zweiten Zeile des LCD erhöht sich die Geschwindigkeit, mit welcher sich der Fortschrittsbalken füllt. Eine entsprechende Fehlermeldung wird ausgegeben, falls die Allokation des Speichers nicht gelingt. Es ist noch zu beachten, dass bei jedem Durchlauf der Testtask terminiert, sodass die *Heap Cleanup* für den korrekten Ablauf des Tests bereits integriert sein muss.
In der zweiten Phase des Testtasks wird ein Speicherbereich alloziert und unmittelbar danach wieder freigegeben. Die Größe des Speicherbereiches wächst dabei bei jeder weiteren Allokation. Zu beachten ist, dass bei der Freigabe des Speichers, die Adresse nicht unbedingt auf den Anfang des Speicherbereiches zeigt, der freigegeben werden soll. Analog wird hier in der oberen Zeile des LCD auch ein Fortschrittsbalken angezeigt. Mit dem steigenden Index in der zweiten Zeile des LCD sinkt hier jedoch die Geschwindigkeit, während Speicherbereiche freigegeben werden.
War der gesamte Testtask erfolgreich wird zum Abschluss für drei Sekunden `„TEST COMPLETE"` auf dem LCD angezeigt. Im Anschluss daran wird der Testprozess beendet - nachdem der Benutzer mit einem beliebigen Tastendruck bestätigt - sodass der Leerlaufprozess aktiv werden sollte.

Fehlermeldungen

Could not alloc!
Obwohl noch freier Speicher vorhanden ist, konnte keiner alloziert werden.

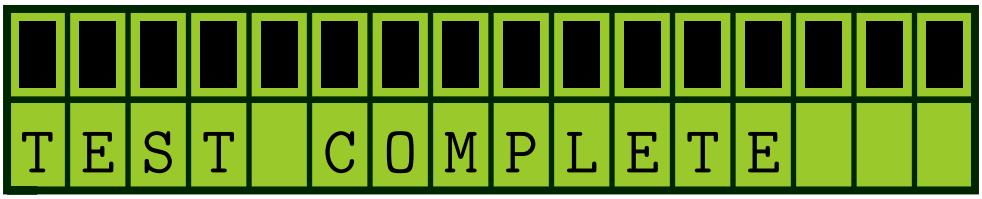

Abbildung 3.7: Range nach erfolgreichem Durchlauf

3.8 Allocation Strategies (in Versuch 3 optional)

Der Testtask *Allocation Strategies* ist nicht für das Bestehen des Versuchs erforderlich. Mit Hilfe des Testtasks können die in Versuch 3 optionalen Allokationstrategien *NextFit*, *BestFit* und *WorstFit* überprüft werden. Es wird vorausgesetzt, dass die beiden Funktionen `os_malloc` und `os_free` korrekt funktionieren.

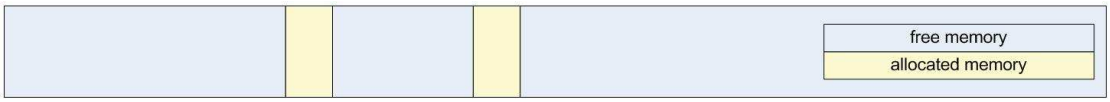

Abbildung 3.8: Absichtlich fragmentierter Speicher.

Zunächst wird überprüft, ob der Heap eine gewisse Mindestgröße aufweist. Nachdem geprüft wurde, ob die gesamte Allokationstabelle frei ist, wird mit jeder Strategie versucht ein Speicherbereich zu allozieren, der größer als der Use-Bereich ist. Ist dies der Fall, beendet der Test mit dem Fehler `Overalloc`. Im nächsten Schritt erzeugt der Test im Speicher ein Muster, sodass am Anfang des Heaps ein großer, freier Bereich von 15 Byte, ein kleinerer, freier Bereich von 10 Byte und der restliche freie Speicher existiert. Diese drei Bereiche werden durch zwei mit 5 Byte allozierten Speicherbereiche getrennt. Abbildung 3.8 zeigt die so entstandene Fragmentierung des Speichers schematisch. Anschließend werden die Allokationsstrategien nacheinander getestet. Dabei werden zwei Speicherbereiche von 10 Byte alloziert und direkt freigegeben. Im Anschluss daran wird die aktuelle Strategie überprüft, indem die beiden Speicheradressen miteinander verglichen werden. Bei *NextFit* wird. z.B. angenommen, dass beide Speicheradressen unterschiedlich sind. Des Weiteren sollte sich die Adresse des ersten allozierten Speicherbereiches im ersten freien Abschnitt befinden während die zweite Allokation im zweiten Abschnitt zu finden sein sollte. Anschließend werden alle allozierten Speicherbereiche wieder freigegeben.
Zusätzlich wird, falls Sie NextFit aktiviert haben, am Ende des Tests überprüft, ob mit NextFit auch dann noch Speicherstellen alloziert werden können, wenn der lastChunk-Zeiger in der Mitte eines freien Speicherbereichs liegt. Es ist möglich, jede Strategie einzeln zu testen. Dazu weisen sie jeweils den `#define`s `FIRST`, `NEXT`, `BEST` und `WORST` den Wert `1` zu. Ist ihre Implementierung erfolgreich, so wird nach dem Test `All passed` auf dem LCD ausgegeben. Andernfalls erscheint ein Error mit dem Namen der Strategie, die den Fehler verursacht hat. Zusätzlich wird am Ende `Check failed` ausgegeben. Zum Abschluss des Tests wird auf die Terminierung des Prozesses gewartet, sodass am Ende der Benutzer bestätigen kann und der Leerlaufprozess laufen sollte.

Fehlermeldungen

Heap too small.
Der Heap weist nicht die erforderliche Mindestgröße auf.

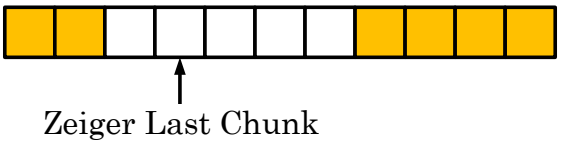

Abbildung 3.9: Beispiel für eine Speicherbelegung des Special NextFit-Tests.

Map not free.
Die Allokationstabelle enthält belegte Speicherbereiche.

Overalloc.
Es konnte ein Speicherbereich alloziert werden, der Größer als der Use-Bereich ist.

Error [Strategy].
Das Verhalten der Allokationstrategie `Strategy` entspricht nicht der Vorgabe.

Map not free afterwards
Das Aufräumen des Speichers mit Hilfe von `os_free` war nicht erfolgreich.

Check failed.
Mindestens eine der Allokationsstrategien hat nicht das gewünschte Verhalten aufgewiesen.