Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Praktikum Systemprogrammierung

Versuch 4

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

4 Testtaskbeschreibung

4.1 Memtest

Der Testtask *Memtest* überprüft, ob der externe Speicher- und der externe Heaptreiber korrekt arbeiten. Der gesamte Testtask gliedert sich in fünf Phasen.

Erste Phase In der ersten Phase von *Memtest* werden die charakteristischen Werte des internen und externen Speichertreibers überprüft. Es wird sowohl das 1:2 Verhältnis von Map- zu Use-Bereich kontrolliert, als auch die Allokationstabelle auf eine entsprechende Mindestgröße überprüft. 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.

Zweite Phase Die zweite Phase überprüft die Korrektheit des externen Speichertreibers und lässt sich in vier Abschnitte unterteilen. Im Folgenden werden diese Abschnitte erläutert:
(**1. Double Access**) In diesem Abschnitt wird geprüft, ob nach einer Schreiboperation auf eine Adresse im externen Speicherbaustein das zuvor geschriebene Datenbyte ausgelesen werden kann. Dieser Vorgang wird für alle Speicheradressen wiederholt.
(**2. Address Bits**) In diesem Abschnitt wird das Anlegen der Adressbits an den Speicherbaustein überprüft. Dieser Abschnitt besteht aus einer statischen und einer dynamischen Phase. Die statische Phase wird vor jeder dynamischen Phase ausgeführt und prüft durch diverse Schreib- und Leseoperationen, ob die jeweils zu lesenden bzw. zu schreibenden Datenbytes korrekt im externen Speicherbaustein abgelegt werden. Beispielsweise überprüft der Testtask, ob nach dem Schreiben des Datenbytes `0xFF` an Adresse `0` und dem Schreiben des Datenbytes `0` an Adresse `0xFF` immer noch das Datenbyte `0xFF` von Adresse `0` gelesen werden kann. In der dynamischen Phase wird ein Adressbit einzeln gesetzt und geprüft ob auf der resultierenden Adresse ein eindeutiges Muster geschrieben und gelesen werden kann. Die dynamische Phase und damit auch die statische Phase, wird für jedes Adressbit zweimal durchgeführt.
(**3. Integrity**) Dieser Abschnitt ist in zwei Durchläufe gegliedert, um die Integrität des Speichers zu überprüfen. Im ersten Durchlauf wird der gesamte Speicher mit einem Muster beschrieben. Im zweiten Durchlauf wird dieses Muster erneut ausgelesen, um zu prüfen ob eine Speicherstelle doppelt bzw. gar nicht beschrieben wurde.

(**4. Inversions**) In diesem Abschnitt wird geprüft ob die Inversion eines Datenbytes, welches sich bereits im Speicher befindet, korrekt im Speicher abgelegt und wieder ausgelesen werden kann. Dafür wird für jede Speicheradresse ein Muster erzeugt welches abwechselnd unverändert und invertiert auf der entsprechenden Speicherstelle abgelegt wird. Entspricht das ausgelesene Muster nicht dem zuvor abgelegten Muster stellt dies einen Fehler dar. Insgesamt wird diese Phase für sechs verschiedene Muster- und Adressabfolgen durchgeführt.

Tritt in diesem Abschnitt des Testtasks ein Fehler auf, wird dies mittels entsprechender Fehlermeldung gekennzeichnet. Die dabei angegebene Zahl stellt die Anzahl der aufgetretenen Fehler dar.

Die dritte und vierte Phase sind bereits von dem Testtask *FreeMap* bekannt.

Dritte Phase In der dritten Phase des Testtasks wird mit Hilfe von `os_malloc` die ganze Allokationstabelle des externen Speichers belegt. Anschließend wird mit der unteren Treiberstruktur überprüft, ob die Speicherallokation gemäß Protokoll implementiert wurde. Treten hierbei Fehler auf, wird wieder eine Fehlerübersicht in Tabellenform angezeigt.

Vierte Phase In der vierten Phase werden Bytes am Anfang des Use-Bereichs alloziert, sodass 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.

Fünfte Phase Die fünfte Phase des Testtasks *Memtest* überprüft, ob der Lese- und Schreibvorgang des externen Speichertreibers mithilfe kritischer Sektionen realisiert wurde. Dazu werden vier zusätzliche Prozesse gestartet. Jeder Prozess schreibt sein eigenes Muster in einen eigenen Speicherblock in den externen Speicher. Beim nachfolgenden Lesevorgang wird überprüft, ob die gelesenen Werte noch mit dem geschriebenen Muster übereinstimmen. Ist dies nicht der Fall, so wird dies durch die Fehlermeldung `„Pattern mismatch"` auf dem LCD angezeigt. Nach einer kurzen Wartezeit wird dann angezeigt, wo es einen Bruch des Musters gab. Dazu wird in der ersten Zeile des LCDs der Prozess[1] angezeigt, der diesen Fehler bemerkte, gefolgt von der Speicheradresse, an der dieser Fehler auftrat. In der zweiten Zeile wird angezeigt, welches Byte gelesen wurde und welches Byte gelesen werden sollte. Mit dem Druck auf einen beliebigen Button wird die Bearbeitung des Testtasks fortgeführt und etwaige weitere Fehlermeldungen angezeigt. Werden alle drei Phasen bestanden, so wird `„ALL TESTS PASSED"` `„PLEASE CONFIRM!"` auf dem LCD ausgegeben. Nach Bestätigung mit einer beliebigen Taste terminiert der

[1] Hierbei wird angezeigt, der wievielte Schreibprozess er ist, nicht seine `ProcessID`.

Testtask. Traten Fehler auf, wird „`TEST FAILED`" ausgegeben.

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 beide Treiber korrekt von `os_lookupHeap` zurückgegeben wurden.
`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

In dieser Phase wird einerseits die aktuelle Fehleranzahl in jedem Test und andererseits am Ende die Anzahl der fehlgeschlagenen Tests ausgegeben.
`Test #`[1,4]`:`
`FEHLER:` *Anzahl*

`Failed tests`
`(Phase 2): #`[1,4]

Fehlermeldungen Phase 3

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 4

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.

Pattern mismatch

In Phase Fünf hat ein Prozess nach dem schreiben seines Musters nicht das korrekte Muster wieder auslesen können.

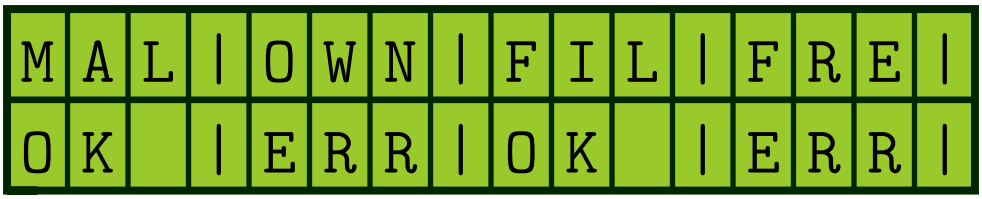

Abbildung 4.1: Es gab einen Fehler beim Setzen des Besitzers des Speicherblocks und beim wieder-freigeben des selbigen.

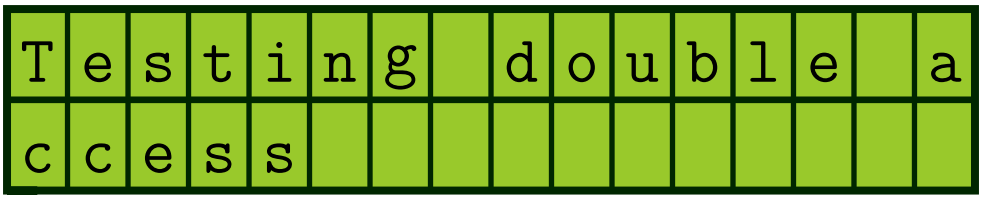

Abbildung 4.2: Memtest während einem Test in der zweiten Phase

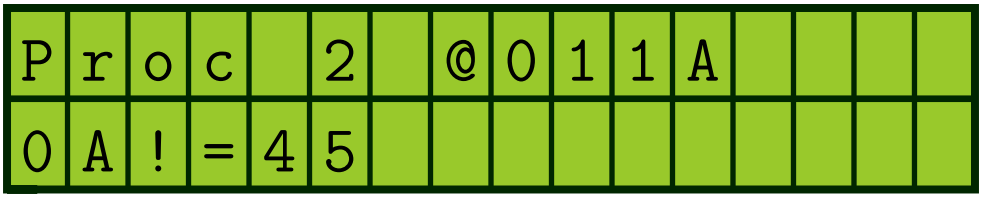

Abbildung 4.3: Fehlerausgabe von Memtest in der fünften Phase. Der zweite Schreibprozess hat einen Fehler an der Speicheradresse `0x011A` festgestellt. Er hat den Wert `0x0A` gelesen, obwohl dort der Wert `0x45` gelesen werden sollte.

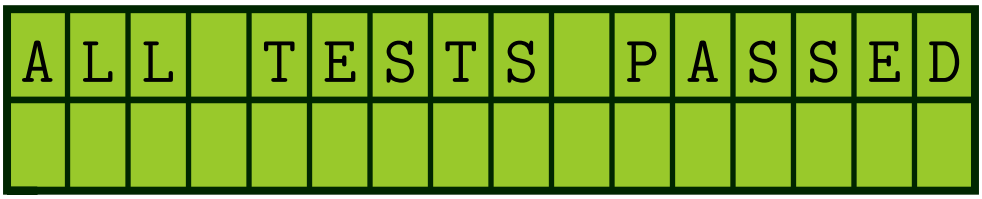

Abbildung 4.4: Memtest nach bestandenem Durchlauf aller fünf Phasen

4.2 Range

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 mithilfe 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. Bedenken Sie außerdem, dass der in diesem Versuch neu hinzugekomme externe SRAM auf Grund seiner Größe ein optimiertes Bereinigen des Heaps erfordert. Ohne diese Optimierung ist das absolvieren des Testtasks in der Präsenzeit nicht realistisch (Ein in Versuch 3 bereits erfolgreich absolvierter *Range*-Test führt nicht zur Anerkennung des Tests in Versuch 4).

In der zweiten Phase wird ein Speicherbereich alloziert und unmittelbar danach wieder freigegeben. Die Größe des Speicherbereiches wächst dabei bei jeder weiteren Allokation. Es ist zu beachten, dass die zur Speicherfreigabe verwendete Adresse nicht unbedingt auf den Anfang des Speicherbereichs 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.
Verlaufen alle Phasen dieses Tests erfolgreich wird „`ALL TESTS PASSED`" „`PLEASE CONFIRM!`" auf dem LCD ausgegeben. Nach Bestätigung mit einer beliebigen Taste terminiert der Testtask und es startet der Leerlaufprozess.

Fehlermeldungen

Could not alloc! (Phase [1,2])
In der jeweiligen Phase des Tests konnte zu einem Zeitpunkt kein Speicher alloziert werden.

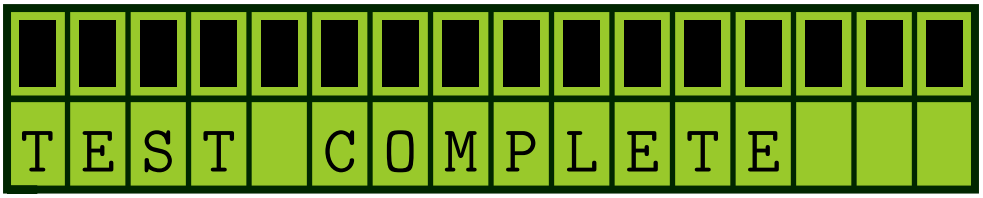

Abbildung 4.5: Range nach erfolgreichem Durchlauf.

4.3 Heap Cleanup

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 wird „`ALL TESTS PASSED`" „`PLEASE CONFIRM!`" auf dem LCD ausgegeben. Nach Bestätigung mit einer beliebigen Taste terminiert der Testtask und es startet der Leerlaufprozess.

Fehlermeldungen

Missing heap!
Es ist nur ein Heap definiert.

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.

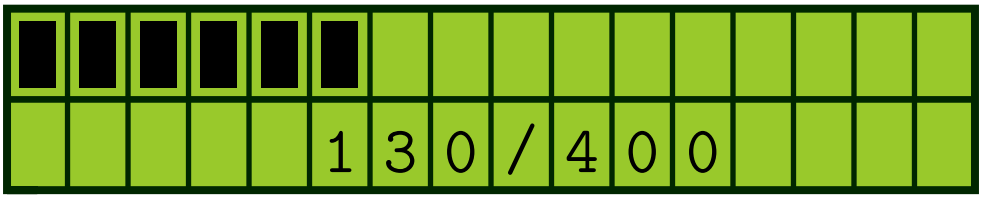

Abbildung 4.6: Heap Cleanup während des Tests.

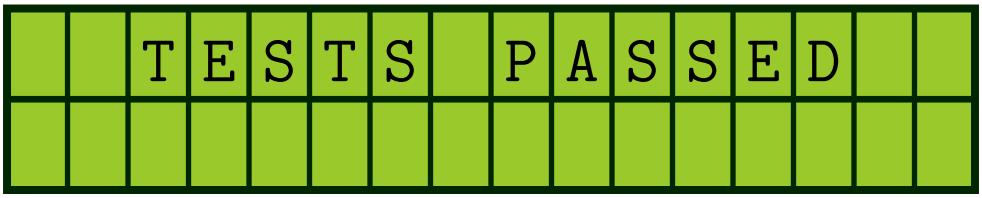

Abbildung 4.7: Anzeige, falls alle Phasen des Tests erfolgreich verliefen.

Nach Bestätigung mit einer beliebigen Taste terminiert der Testtask und es startet der Leerlaufprozess.

4.4 Alloc Strategies

Der Testtask *Alloc Strategies* überprüft die korrekte Arbeitsweise der vier Allokationsstrategien.

Der Testtask arbeitet analog zum Test der Allokationsstrategien aus Versuch 3. In diesem Versuch müssen darüber hinaus jedoch alle vier Allokationsstrategien überprüft werden. Im Laufe der Testtaskausführung wechselt der Testtask zur FirstFit Allokationsstrategie. Sorgen Sie deshalb zuerst für eine korrekte Implementierung der FirstFit Strategie.

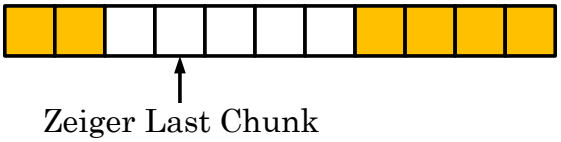

Abbildung 4.8: Beispiel für eine Speicherbelegung des Special NextFit-Tests.

4.5 Stability Private with Allocstrats

Der Testtask *Stability Private with Allocstrats* 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 LCDs die bisher verstrichene Zeit und in der unteren Zeile fortlaufend „`1a 2a 1b 1c 2b 3a 2c 2d 1d 3b...`" ausgegeben. Dabei gibt die Ziffer den Prozess und der Buchstabe die aktuelle Phase des Testtasks an, in der er sich befindet. Folglich müssen die Buchstaben a-d sowie die Zahlen 1-5 auftreten. Die Reihenfolge der Zahlen-Buchstaben-Paare ist hierbei irrelevant. Es wird sowohl der interne Speicher, als auch der externe Speicher getestet. 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. Nachdem in Phase *b* etwas Zeit verstrichen ist, wird in Phase *c* überprüft, ob die geschriebenen Werte im Speicherbereich noch mit den Originalwerten übereinstimmen. Ist dies nicht der Fall, wird die Fehlermeldung „`Pattern mismatch internal!`" bzw. „`Pattern mismatch external!`" angezeigt. Freigegeben wird der Speicherbereich in Phase *d*.
Während des Testtasks wird in regelmäßigen Abständen zwischen allen Allokationsstrategien gewechselt. Deshalb wird je nach ausgewählter Allokationsstrategie im Abstand von einigen Sekunden „`Change to f/n/b/w`" auf dem Display angezeigt. Im Anschluss daran wird zur ursprünglichen Anzeige zurückgekehrt.

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 (internal/external)
Die Daten im allozierten Speicher wurden verändert.

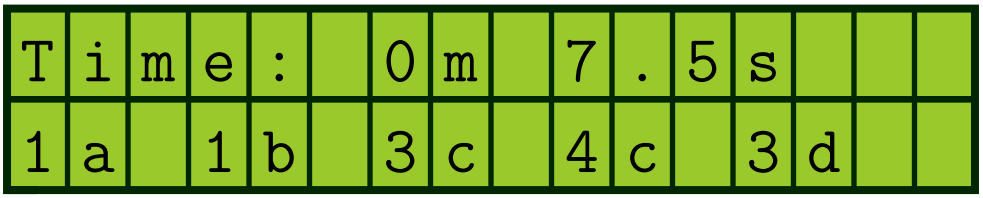

Abbildung 4.9: Durchlauf von Stability Private with Allocstrats nach sieben Sekunden.

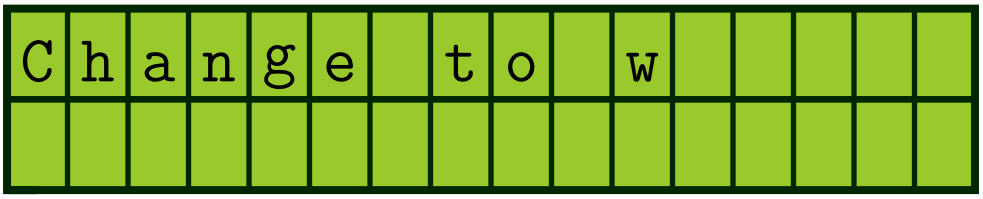

Abbildung 4.10: Wechsel der Schedulingstrategie in Stability Private with Allocstrats zu Worst-Fit.

4.6 Realloc

Der Testtask *Realloc* überprüft, ob die Reallokationsroutine korrekt implementiert wurde. Die Routine wird dabei parallel sowohl auf dem internen SRAM als auch auf dem externen SRAM getestet. Der Test unterscheidet dabei acht Phasen in denen jeweils ein Speicherblock realloziert werden muss:

1. **R. res.**: Der Speicherblock wird vergrößert. Hinter dem Speicherblock (rechts) ist ausreichend freier Speicher vorhanden. Es wird geprüft ob die Routine diesen benutzt, um den Speicherblock zu vergrößern.

2. **L. res.**: Der Speicherblock wird vergrößert. Es wird geprüft, ob die Routine den freien Speicher vor dem Speicherblock (links) nutzt, um diesen zu vergrößern und die Daten dabei konsistent mitverschiebt.

3. **L. res. frc.**: Der Speicherblock wird vergrößert. Dafür gibt es keinen freien Speicherbereich ausreichender Größe. Das einzig mögliche Vorgehen ist die Vergrößerung des Speicherblocks in den Bereich vor diesem. Auch hierbei sollen die Daten mitverschoben werden.

4. **LR res.**: Der Speicherblock wird vergrößert. Die Menge an freiem Speicher vor und hinter dem Speicherblock reicht aus, um den Speicherblock zu vergrößern. Es wird geprüft, ob dies erkannt und richtig darauf reagiert wird.

5. **LR res. frc.**: Der Speicherblock wird vergrößert. Die Menge an freiem Speicher vor und hinter dem Speicherblock reicht aus, um den Speicherblock zu vergrößern. Zusätzlich ist dies die einzige Möglichkeit, um die Reallokation durchzuführen, da

sonst kein freier Speicherblock ausreichender Größe vorhanden ist. Es wird geprüft, ob dies erkannt und richtig darauf reagiert wird.

6. **Move**: Der Speicherblock wird vergrößert. Der freie Speicher vor und hinter dem Speicherblock ist zusammen mit dem schon allozierten Speicherblock kleiner als die zu erreichende Größe. Der Block muss verschoben werden. Es wird geprüft, ob dies geschieht und ob die Daten dabei mitverschoben werden.

7. **Shrink**: Der Speicherblock wird durch die Routine verkleinert. Auch das Verhalten bezüglich der Daten innerhalb des nun freigegebenen Speichers wird überprüft.

8. **Keep**: Der Speicherblock wird auf die Größe, die er momentan hat, angepasst. Es wird überprüft, ob sich der Speicherzustand ändert.

Wurden alle Phasen ohne Fehler abgeschlossen, wird `„ALL TESTS PASSED"` `„PLEASE CONFIRM!"` auf dem LCD ausgegeben. Nach Bestätigung mit einer beliebigen Taste terminiert der Testtask und es startet der Leerlaufprozess. Traten hingegen Fehler auf, erscheint auf dem LCD der Text `„TEST FAILED!"`

Fehlermeldungen

malloc failure
Während des Tests konnte kein Speicher mittels `os_malloc()` alloziert werden.

realloc memcpy fail
Nach dem Verschieben eines Speicherbereichs wurde der Inhalt des Use-Bereichs nicht korrekt kopiert.

realloc allocation fail
Während des Tests hat `os_realloc()` die Adresse `0` zurückgegeben, obwohl eine Reallokation möglich war.

realloc map adapt. fail
Laut Allokationstabelle hat der reallokierte Speicherbereich nicht die gewünschte Größe.

realloc map damage
Nachdem ein Speicherblock reallokiert wurde, wurden die Einträge von anderen Speicherbereichen in der Allokationstabelle beschädigt.

realloc pattern broken
Nach der Nutzung des reallokierten Speicherbereichs, wurden Daten in anderen Speicherblöcken beschädigt. Dies bedeutet, dass der zurückgegebene Speicher mit bereits allokierten Speicherblöcken kollidiert.

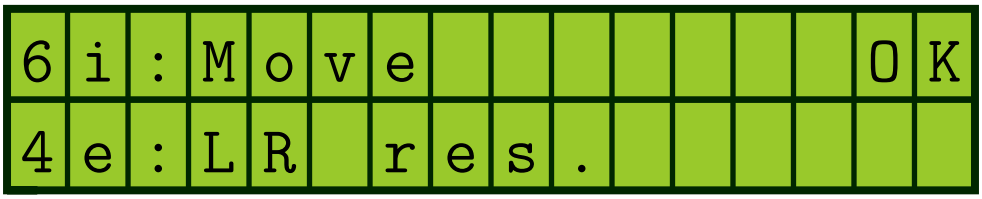

Abbildung 4.11: Realloc. Phase 6 wurde auf dem internen RAM erfolgreich abgeschlossen, auf dem externen RAM läuft gerade Phase 4.