Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Praktikum Systemprogrammierung

Versuch 4

Testtaskbeschreibung

Lehrstuhl Informatik 11 - RWTH Aachen

6. November 2023

Commit: 2f3d1172

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 Allocation Strategies

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

Der Testtask arbeitet analog zum Test der Allokationsstrategien aus Versuch 3. In diesem Versuch werden darüber hinaus jedoch alle vier Allokationsstrategien überprüft.

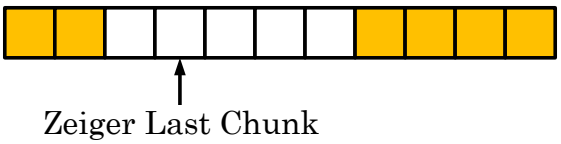

Abbildung 4.1: Beispiel für eine Speicherbelegung des Special NextFit-Tests.

4.2 Memtest

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

Phase 1 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 sichergestellt, dass der externe Heap die zur Verfügung stehende Größe des Speichers ausnutzt. 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.

Phase 2 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 *Free Map* bekannt.

Phase 3 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.

Phase 4 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.

Phase 5 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.

Nachdem alle Phasen erfolgreich ausgeführt wurden, zeigt das LCD „PRESS ENTER" an. Nachdem dies mit einem Tastendruck bestätigt wurde, wird „TEST PASSED" und „WAIT FOR IDLE" angezeigt und der Testtask terminiert. Im Anschluss muss der Leerlaufprozess ausgeführt werden.

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:
„LST": „OK", wenn die Heap-Liste die korrekte Länge hat
„DRV": „OK", wenn beide Treiber korrekt von `os_lookupHeap` zurückgegeben wurden.
„SIZ": „OK", wenn der externe Heap den vollen Speicher ausnutzt.
„1:2": „OK", wenn das Verhältnis von Map- zu Use-Bereich korrekt ist

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]"

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.

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

Phase 4 Hier gibt es nur zwei Spalten mit je einem Titel und einem Testergebnis:
„MAP": „OK", wenn die Map durch `os_free` korrekt verändert wurde.
„USE": „OK", wenn der Use-Bereich durch `os_free` nicht verändert wurde.

Phase 5
„Pattern mismatch": Ein Prozess nach dem schreiben seines Musters nicht das korrekte Muster wieder auslesen können.

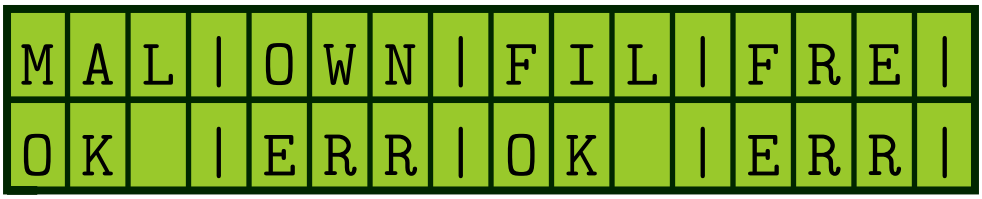

Abbildung 4.2: Es gab einen Fehler beim Setzen des Besitzers des Speicherblocks und beim wieder-freigeben des selbigen.

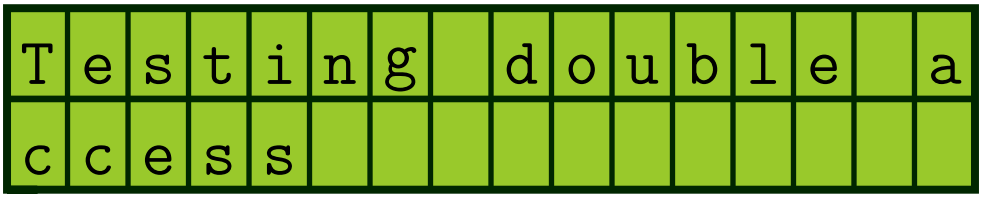

Abbildung 4.3: Memtest während einem Test in der zweiten Phase

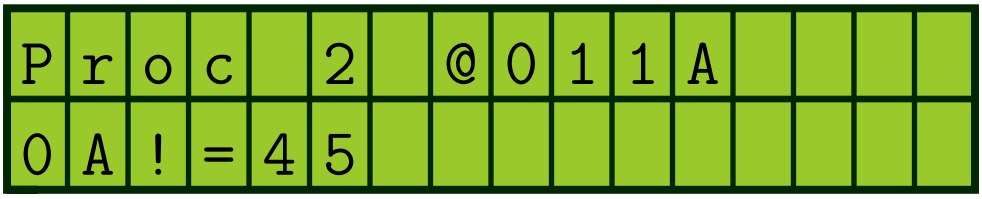

Abbildung 4.4: Fehlerausgabe von Memtest in der fünften Phase. Der zweite Schreibprozess hat einen Fehler an der Speicheradresse `0x011A` festgestellt. Er hat den Wert `0x0A` gelesen, obwohl dort der Wert `0x45` gelesen werden sollte.

4.3 Heap Cleanup

Der Testtask *Heap Cleanup* überprüft, ob beim Terminieren von Prozessen der dynamische Speicher korrekt aufgeräumt wird.

Der Testtask arbeitet analog zum optionalen Testtask aus Versuch 3. In diesem Versuch werden darüber hinaus jedoch sowohl der interne als auch der externe Heap getestet.

4.4 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. Mit diesem Testtask soll nur der interne Speicher getestet werden, da der externe Speicher zu groß ist um in der erwarteten Zeit getestet zu werden.

Der Testtask arbeitet analog zum optionalen Testtask aus Versuch 3.

4.5 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 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 drei Minuten keine Fehlermeldung ausgegeben und dann „`TEST PASSED`" angezeigt 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.

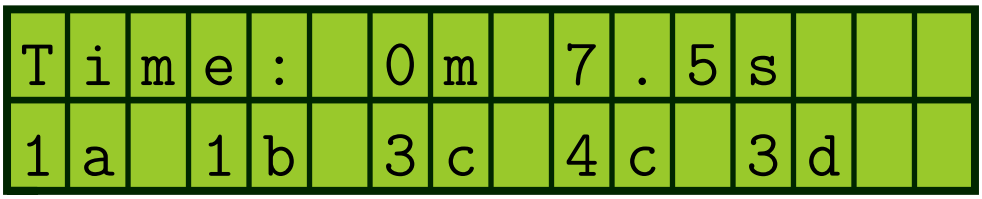

Abbildung 4.5: Durchlauf von Stability Private nach sieben Sekunden.

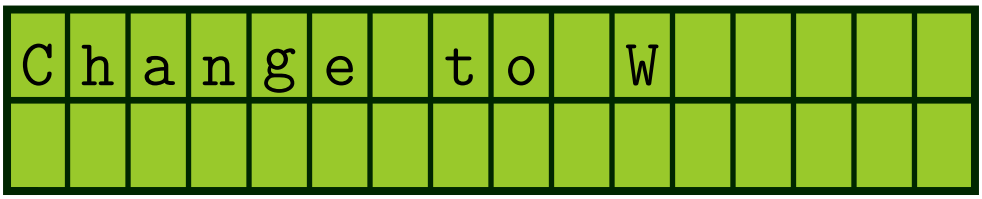

Abbildung 4.6: Wechsel der Schedulingstrategie in Stability Private zu Worst-Fit.

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 neun Phasen in denen jeweils ein Speicherblock realloziert werden muss:

1. **Owner**: Ein Prozess versucht den Speicherblock eines anderen zu reallozieren. Dies soll zu einem Fehler führen, der bestätigt werden muss. Zusätzlich wird überprüft, ob danach der Wert 0 zurückgegeben wird, um dem Aufrufer mitzuteilen, dass die Reallokation fehlgeschlagen ist.

2. **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.

3. **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.

4. **L. 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.

5. **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.

6. **LR. 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.

7. **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.

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

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

Nachdem alle Phasen erfolgreich ausgeführt wurden, zeigt das LCD „`PRESS ENTER`" an. Nachdem dies mit einem Tastendruck bestätigt wurde, wird „`TEST PASSED`" und „`WAIT FOR IDLE`" angezeigt und der Testtask terminiert. Im Anschluss muss der Leerlaufprozess ausgeführt werden.

Fehlermeldungen

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

No error (owner)
Das Reallozieren von fremden Speicherbereichen hat nicht wie erwartet zu einem Fehler geführt.

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

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

map adaptation
Laut Allokationstabelle hat der reallokierte Speicherbereich nicht die gewünschte Größe.

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

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.

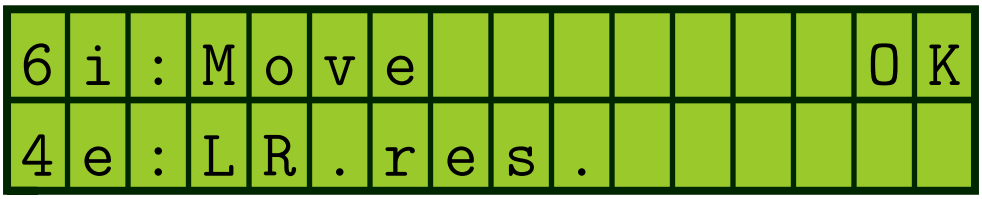

Abbildung 4.7: Realloc. Phase 6 wurde auf dem internen RAM erfolgreich abgeschlossen, auf dem externen RAM läuft gerade Phase 4.