End the Senseless Killing: Improving Memory Management for Mobile Operating Systems

Niel Lebeck, Arvind Krishnamurthy, Henry M. Levy, Irene Zhang

Paul G. Allen School of Computer Science & Engineering

Microsoft Research
Disclaimers

• I currently work at Google (not on Android)
• This work is not connected with Google
  • Research was done while I was a graduate student at UW
  • All data is from our experiments or open-source resources
  • All opinions are our own
This talk

• Motivation
• Key insight and Marvin
• Marvin’s mechanisms
• Marvin’s features
• Implementation and evaluation
Today’s mobile memory management is bad for users and applications

• Each app gets a fixed maximum memory budget
• Mobile OS kills apps when the device runs out of memory
  • Even if apps are not actively using their memory
• Restarting apps takes time
• Developers must optimize app memory usage
Traditional swapping is not a solution

• Not suited to managed languages (e.g., Java)
  • Garbage collection causes swapping, confuses working set estimation (WSE)
  • Page-granularity swapping and WSE do not fit variable-sized objects

• Not suited to latency-sensitive touch devices
  • On-demand swapping causes stuttering and delays
This talk

• Motivation
• Key insight and Marvin
• Marvin’s mechanisms
• Marvin’s features
• Implementation and evaluation
Key insight

• We can *co-design the runtime and OS* to improve mobile memory management

• Possible because modern mobile platforms require all apps to use the same runtime
Marvin

• Android memory manager co-designed with Android’s Java runtime
• Reintroduces swapping to the mobile environment
Marvin

- Marvin has three main features:
  - Ahead-of-time swap
  - Object-level working set estimation
  - Bookmarking garbage collector [Hertz 05]
This talk

• Motivation
• Key insight and Marvin
• Marvin’s mechanisms
  • Stubs
  • Reclamation table
  • Object access interposition

indirection layer between objects
allows the runtime and OS to coordinate
lets the runtime transparently take action

• Marvin’s features
• Implementation and evaluation
Stubs

• We need an indirection layer between objects referencing each other
  • Catch accesses to swapped-out objects
• **Stubs** provide that indirection layer
  • Small pseudo-objects that sit in the Java heap and point to the “real” object
  • Store copies of the real object’s references
Reclamation table

- We need a way for the runtime and OS to coordinate
  - Tell OS which objects can be reclaimed
  - Prevent OS from reclaiming an object being used by the runtime
- A shared-memory *reclamation table* allows that coordination
  - Stores an object’s location and size, and has metadata bits for locking

<table>
<thead>
<tr>
<th></th>
<th>address</th>
<th>size</th>
<th>res</th>
<th>app lock</th>
<th>kernel lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xc00d00a0</td>
<td>512</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0xc00e1410</td>
<td>128</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0xc0002320</td>
<td>8192</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Object access interposition

- The runtime needs a way to transparently act when app code accesses objects
  - Restoring swapped-out objects
  - Update working set metadata
- **Object access interposition** is a set of paired interpreter and compiler modifications implementing those actions
  - Interpreter directly acts when performing object accesses
  - Compiler generates additional ARM64 instructions around object accesses
This talk

• Motivation
• Key insight and Marvin
• Marvin’s mechanisms

• Marvin’s features
  • Ahead-of-time swap
  • Object-level working set estimation
  • Bookmarking garbage collector

• Implementation and evaluation
Ahead-of-time swap

• Runtime uses **object access interposition** to set dirty bit in object header
• Runtime clears dirty bit after saving an object
• Kernel checks dirty bit before reclaiming an object

```java
foo.setX(42);
```
Ahead-of-time swap

• Runtime uses **object access interposition** to set dirty bit in object header
• Runtime clears dirty bit after saving an object
• Kernel checks dirty bit before reclaiming an object
This talk

• Motivation
• Key insight and Marvin
• Marvin’s mechanisms
• Marvin’s features
  • Ahead-of-time swap
  • Object-level working set estimation
  • Bookmarking garbage collector
• Implementation and evaluation
Object-level working set estimation

- Runtime uses **object access interposition** to set access bits
- Runtime scans access bits and updates longer-term WSE metadata

```java
int x = foo.getX();
```
This talk

- Motivation
- Key insight and Marvin
- Marvin’s mechanisms
- Marvin’s features
  - Ahead-of-time swap
  - Object-level working set estimation
  - Bookmarking garbage collector
- Implementation and evaluation
Bookmarking garbage collector

- Runtime uses **object access interposition** to maintain stub references
- GC detects **stubs** and reads references without touching underlying objects

```java
foo.setMember(bar)
```

![Diagram of garbage collection process]
Bookmarking garbage collector

- Runtime uses **object access interposition** to maintain stub references
- GC detects **stubs** and reads references without touching underlying objects

![Diagram showing the garbage collector, Java heap, stub, bar, member, reclamation table, and reclaimable object space.](image-url)
This talk

• Motivation
• Key insight and Marvin
• Marvin’s mechanisms
• Marvin’s features
  • Ahead-of-time swap
  • Object-level working set estimation
  • Bookmarking garbage collector
• Implementation and evaluation
Marvin implementation

• We modified the Android Runtime (ART) to implement Marvin
• Default policy keeps the foreground app’s objects in-memory
• For experiments, we trigger reclamation in the runtime
Evaluation

• Experimental setup:
  • Pixel XL phones
  • Android 7.1.1 (or our modified build)

• Questions:
  • Does Marvin let users run more apps?
  • Does ahead-of-time swap work?
  • What is Marvin’s overhead?
Does Marvin let users run more apps?

• We periodically started instances of a benchmark app
  • Initializes a 220MB heap with a mix of 4KB and 1MB arrays
  • Deletes and re-allocates 20MB of those arrays every 5 seconds

• We measured the number of active apps: apps that are alive and making progress on their workloads
Does Marvin let users run more apps?

- Marvin can run over 2x as many apps as stock Android.
- On Android w/ Linux swap, a little allocation makes apps unusable.

Android w/ Linux swap consistently crashed early.
Does ahead-of-time swap work?

- Marvin reclaims memory much faster than Android w/ Linux swap

Marvin

- Memory reclaimed in ≈100ms

Android

- Memory reclaimed in ≈8 seconds
What is Marvin’s overhead?

- When objects are memory-resident, execution overhead depends on proportion of object accesses

- Overhead is reasonable (15%) on PCMark benchmark
Related work

Similarities with Marvin

- Acclaim [Liang 20]
- SmartSwap [Zhu 17]
- A2S [Kim 17]
- MARS [Guo 15]

*Policies distinguish between foreground and background apps*
Related work

Similarities with Marvin

Acclaim [Liang 20]

SmartSwap [Zhu 17]

A2S [Kim 17]

MARS [Guo 15]

Addresses incompatibility of garbage collection and kernel-level swap
Related work

Differences from Marvin

Acclaim [Liang 20]
SmartSwap [Zhu 17]
A2S [Kim 17]
MARS [Guo 15]

Perform swapping at the kernel level rather than the runtime level
Conclusion

• **Problem:** Today’s mobile memory management is inadequate

• **Insight:** We can co-design the runtime and OS to improve memory management

• **Solution:** Marvin improves mobile memory management with three co-design features
  • Ahead-of-time swap
  • Object-level working set estimation
  • Bookmarking garbage collection
Thanks!

• Marvin source code is available on GitHub: https://github.com/UWSysLab
• Contact: nl35@cs.washington.edu