barkhausen institut

Allocator

Microkernel-based Operating Systems—Exercise

Barkhausen Institut, Dresden (Germany)

Viktor Reusch, Martin Küttler

WiSe 26/27

Memory Allocation



- Ideally, we would implement libC's `malloc()` and `free()` using L4Re dataspaces and the region manager
- However, this intrusive change turned out to be unmaintainable
- Instead: C++'s allocator interface
- Allocator adapter already provided in `src/l4/pkg/alloc/include/alloc.h`
- You only have to fill in `src/l4/pkg/alloc/lib/pool.cc`
- The `Pool` is the underlying allocator that is used by `Allocator`s of various types
- You can use the C++ allocator like this

```
Alloc::Pool pool;
std::vector<int, Alloc::Allocator<int>> ints{pool};
ints.push_back(42);
```

- For the sake of the exercise, do not use dynamic allocations (`new`, `std::vector()`) in your allocator

Page Allocation



- You can get new pages from Moe:
  - Allocate a dataspace capability
  - Get a dataspace from Moe: `L4Re::Env::env()->mem_alloc()->alloc(size, ds);`
  - Attach dataspace to local address space:

    ```
    L4Re::Env::env()->rm()->attach(&addr, size, flags, ds);
    ```
- To free unused pages:

  ```
  L4Re::Env::env()->rm()->detach(addr, nullptr);
  L4Re::Util::cap_alloc.free(ds, L4Re::Env::env()->task().cap());
  ```
  - Alternatively use `L4Re::Rm::Unique_region` and `L4Re::Util::Unique_cap`.

Naïve allocate()



```
void *Pool::allocate(std::size_t bytes) noexcept {
  L4::Cap<L4Re::Dataspace> ds
    = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();
  if (!ds.is_valid()) return nullptr;

  long err = L4Re::Env::env()->mem_alloc()->alloc(bytes, ds);
  if (err) return nullptr;

  void *addr = nullptr;
  err = L4Re::Env::env()->rm()->attach(&addr, bytes,
    L4Re::Rm::Search_addr | L4Re::Rm::F::RW, ds);
  if (err) return nullptr;

  return addr;
}
```

Memory Management
Lists

- Idea:
  - Keep list of (address, size) pairs
  - In `allocate()`, search for an appropreate entry
- Problem: You'd need dynamic memory for that list.
- Typical Solution: Inlining
  - Put size and next-pointer directly into your memory
  - Do not hand out the memory where size is stored—it's needed for `deallocate()`.
  - That's what most libC implementations do.

Memory Management
Bitmaps

- Manage memory as pool of fixed-sized chunks.
- Use bitmap to store available chunks.



Memory Management
Bitmaps

- Manage memory as pool of fixed-sized chunks.
- Use bitmap to store available chunks.

- There are more options for the implementation. Come up with something yourself, or have a look in some book / the internet.

Assignment 5



- Implement `allocate()` and `deallocate()` using dataspaces and the region manager.
- Write your own little test program that uses the C++ `Alloc::Allocator` of the *alloc* library.
  - Remember to add the *alloc* package to `requires:` in your `Control` file.
  - Also add the *alloc* library to `REQUIRES_LIBS` in your `Makefile`.