5CCS2OSC Operating Systems & Concurrency
InfOS resit coursework
Stephen Kell
18th March 2024

Overview

This is a coursework assignment intended for those re-taking the coursework component of the 5CCS2OSC module.

It is marked out of 15, although a mark cap of 40% (6 out of 15) is likely to apply.

The task is using the InfOS teaching operating system which was introduced during the live large-group sessions and the screencasts.

It is *required* that you submit your work from the shared `5ccs2osc.nms` machine, using the `submit` program. You will make a single submission, with project number 7.

The task: an improved physical memory allocator

Your task is to improve the *frame allocator* in the InfOS kernel. Currently this code accepts requests to allocate one or more contiguous frames of physical memory. Its argument is an *order*, which is the base-2 logarithm of the number of pages: order 0 allocates a single page, order 1 allocates two, order 2 allocates four, and so on. The algorithm does a simple linear scan of an array of *frame descriptors*, looking for a contiguous range big enough. This can be slow if looking for a large range of free pages.

There are two ways you can improve this code: to make it simpler, or to make it faster. You will need to choose one of these options.

Making it simpler

You can score at most 6 out of 15 for completing this task.

InfOS is simple enough that there is no need to provide an allocator for large blocks of physical memory—just allocating one frame at a time is enough.

One way to start is, in `include/infos/mm/page-allocator.h`, to change

```
FrameDescriptor *allocate(int order);
```

to

```
FrameDescriptor *allocate(); // allocates one frame only
```

... and then make the necessary further changes to the entire codebase to restore InfOS to functionality.

An initial attempt, including the change outlined above, is available for you. To ensure the code continues to compile, the argument `order` still exists but defaults to 0 and an assertion will fail if a non-zero argument is passed. This means that currently, the kernel aborts early during the boot process. Your improved kernel will work normally, but will preserve the 'one frame only' simplification, by modifying all places in the code that try to allocate more than one frame. (There are not very many, but it's up to you to find them.)

You must pull changes from the `cw2024resit` branch and base your work on the commit c94576e.

One exception to 'one frame at a time is enough' rule is for the kernel's own `malloc` implementation. This does ask for large chunks of memory to parcel out to its clients, and for complex reasons, these must be physically contiguous even though they are addressed virtually. However, you can ignore this for the purposes of this assignment: the same commit mentioned above includes a modified version of `malloc` that sidesteps this problem.

Making it faster or more memory-efficient

You can score up to the full 15 out of 15 for completing this task. You should attempt *either* this task or the previous one, as they are disjoint.

You should work from the cw2024 branch not the resit branch. Your work should be based on commit 263c3b.

Although InfOS currently has no need to allocate larger-than-a-page blocks of physical memory, these would be useful in a slightly extended version of InfOS. For example, they are needed for DMA buffers; InfOS currently does not use DMA but could reasonably do so.

Therefore, in this task you will provide a more efficient implementation of frame allocation, in a way that continues to handle requests for any power-of-two number of frames.

The current algorithm does a linear search for a contiguous range of *frame descriptors* of sufficient size. These frame descriptors consume a significant amount of memory themselves, and the linear search is potentially very slow when searching for a larger block, especially under high (external) fragmentation.

How you go about improving the efficiency is up to you. However, the following are some ideas.

- You could replace the `FrameDescriptor` structure (currently three words per frame, or thereabouts) with a *bitmap* (one bit per frame). This will improve space-efficiency.
- You could implement a *buddy allocator* to replace the current simple frame allocator. A buddy allocator is so named because any large block can quickly be broken in half, yielding two smaller 'buddies', and conversely, any block has a neighbouring 'buddy' with which it can be combined, if both are free, to form a single large block. Such an allocator can quickly allocate contiguous regions of memory (here physical memory, in units of a page or any power-of-two number of pages), at the price of greater internal fragmentation

(every block it allocates has a power-of-two size). You will have to do your own reading on buddy allocators, which are extensively documented (e.g. on Wikipedia). The main objective here is to improve speed (time-efficiency).

- You could implement an alternative allocator of your own choosing, instead of a buddy allocator. There is a large design space, and the lecture materials covered some of this. The important thing is that it should be faster than the current allocator under plausible workloads.

Up to 6 marks are available for any competently executed refactoring of InfOS which preserves its functionality and is meaningfully connected with the task. The remaining 9 marks are for a minor improvement in time (up to 3 marks) and/or minor improvement in space overhead (up to 3 marks), major improvement in either time or space (up to 3 further marks, to a maximum of 6), and up to 3 marks for documentation, tests, comments and other good practice. If you attempt this task, your code will be subject to automated performance testing as well as manual inspection.

You can score up to the full 15 out of 15 for completing this version of the task. You should check whether a mark cap applies to you.

Submission

You need to submit project 7, repository `infos`, e.g. as follows.

```
/shared/CCS/CC/submit 7 ~/infos
```