Department of Computer Science

COS 222 Operating Systems

Optional Practical

Due date: 21 October 11:00 AM

A van Schalkwyk, H Janse van Vuuren
Marking

Completing this practical is optional. If you decide to complete it, it will count towards your final practical mark. The mark allocation to calculate your semester mark will be based on four prac. If you have completed five prac, your best four marks will be used in the calculation.

Do not confuse this practical with Practical 3. Both are about memory management, but Practical 3 is not optional.

• This practical will be assessed during a session where you are booked, provided that you have uploaded your work before the session.

• Upload screenshots of your work in a zip file before attending a lab session where you would like to be assessed. The upload slot will be available until 21 October 11:00 AM.

• Assessment may be requested on the following dates during the normal lab session times.

  Mondays  19 Oct or 26 Oct
  Tuesdays  20 Oct or 27 Oct
  Wednesdays  30 Sept or 21 Oct
  Thursdays  1 Oct or 22 Oct

  Assessment will be granted only if the time stamp of your upload is shown on the printed lab list and the work has not been assessed before.

• You may submit only your own work. Submitting a solution found online or from another student is not acceptable.

Instructions

This prac consists of two tasks, both of which require you to modify Minix source code. The topic of this prac is memory allocation algorithms.

• The tasks described below may be done independently, although it’s advised to do task 1 first.

• WARNING: Since both tasks require changing Minix source code, be sure to make backups of your Minix disk image, before task 1 and before task 2.

Introduction

One of the most important responsibilities of the Process Manager in Minix is to perform memory management. As a part of memory management, a list of available holes in
memory is maintained. This list is referred to simply as the hole list. The hole list is a singly linked list. Each hole in this linked list has a pointer to the next hole, the base address of the hole, and its length in clicks. When memory is requested (such as when a fork or exec occurs) the process manager searches the hole list to find an appropriate hole in memory. The first-fit algorithm is used to find this hole.

Here is a simplified illustration of the hole list:

Refer to page 431 of the textbook for more information.

**Task 1: Memory statistics [2 marks]**

**Remember to back up your Minix now.**

This task requires that you edit the `alloc_mem` function to show the user some statistics regarding memory allocation as it taking place. It would have been ideal to write these statistics to a file, but it is difficult to write to a file from a Minix server (you can read more about this online). For this reason the output should simply be printed to the terminal each time the `alloc_mem` function is called.

*Note: This will cause a lot to be printed to the terminal, especially during boot. Just type “root” or your username and password as usual to log in, even if some statistics output has cluttered the terminal.*

The information that should be displayed at every `alloc_mem` call is:

- The total number of holes in the hole list
- The number of iterations it took to find an appropriate hole
- The average hole size so far (in Bytes)

**NB:** Please take screen shots of your source code and some of the output before continuing to Task 2. You can run any shell command to get some memory statistics output. You will see that statistics are printed twice when running a shell command. You should know why.
Tips

- The size of a single phys_clicks is system dependent, but you may assume it is 256 Bytes.
- If you don’t see changes after running make, navigate to /usr/src, run make clean, and then make world.

Task 2: Changing the algorithm [3 marks]

Make another backup of your Minix before continuing.

For this task you are to change the algorithm that the Process Manager uses to find an appropriate hole in memory (an unmodified Minix uses first-fit). You may choose any one of the following:

- Best-fit
- Worst-fit
- Random-fit

The statistics should also be printed as with the task 2, displaying all of the same information.

NB: Please take screen shots of your source code and some of the output of task 2.

Tips

- Pay close attention to what you are coding when completing this task, a single mistake will make it impossible for Minix to boot. Continuously starting from scratch due to small avoidable mistakes can become frustrating.
- Wrap you head around how the first-fit algorithm is implemented before attempting to modify the code.
- Don’t copy online solutions. Not only does it take away from the learning experience, but most if not all of them are broken in some way.