CSCI 340 - Operating Systems

Assignment 7 - Total Points 100

Objectives

The objective of this assignment is compare the effectiveness (in terms of external fragmentation, number of probes, and allocation failures) of three different dynamic partitioning placement algorithms (see pages 318-321 in the course textbook). You will be completing the `mem.c` file and the `main.c` file. You must also submit an appropriate `Makefile` to build this assignment.

A computer systems physical memory will be modeled by an array of unsigned integers. Each element in this array represents a single unit of memory. A value of a zero stored in a element represents a free unit (i.e. a unit that is not being used). A positive value stored in the element represents a unit in use. For reference purposes, Fig. 1 shows an example snap-shot of the memory array at some point in time during the simulation.

You will be individually simulating each of the three placement algorithms. Each placement algorithm will be simulated for a specified number of time steps. At each time step a random memory allocation request will be generated of the form, *(size, duration)* , where *size* is a random integer in the range [MIN_REQUEST_SIZE, MAX_REQUEST_SIZE] representing the number of contiguous units for this memory allocation request, and *duration* is a random integer in the range [MIN_DURATION, MAX_DURATION] that represents the amount of time this allocated memory will be in use (see `main.c` for the definition of these constants). For each time duration step, the placement algorithm simulation will:

- call `mem_allocate` and gather statistics on number of probes and number of allocation failures.
- call `mem_single_time_unit_transpired`

When the time iteration loop is completed, the external fragmentation value must then be computed using the `mem_fragment_count` function.

On the command line your program will be run with four arguments: 1) size of of physical memory, 2) number of runs, 3) time steps per run, and 4) random seed used by random number generator. You will be conducting "number of runs" experiments for each placement algorithm. For each placement algorithm you will average and display the external fragmentation results, number of probes, and number of allocation failures. Thus the output will consist of nine different lines.

Note: The random number generator seed value can be changed on the command line and should yield (slightly) different results. Because we are averaging across numerous runs, these results should not be drastically different. In addition, if the same seed is used in two different invocations of your main program, you will obtain the same results.

System and Standard Lib Functions

In this assignment you will choose the system and standard library functions needed to completed the programming assignment. In short, we want you to make the design and implementation decisions.

Provided Files

The three files listed below are provided to you.

- **mem.h**: Header file that defines the function prototypes used in this assignment. **Please note**: You may not modify or add new function definitions to this header file.
- **mem.c**: A file that provides the implementation of each function prototype listed in *mem.h*. You are required to fully implement all the functions except for `mem_init` and `mem_free`, which have been provided to you.
- **main.c**: Source code file that includes a stubbed out main function to be completed by you.

Todo

Many comments are provided above each function in the *main.c* and *mem.c* files. The provided comments give detailed instructions that are meant to guide you through this assignment. Please read them carefully and follow the instructions. For reference purposes, Fig. 1 shows an example snap-shot of the memory array at some point in time during the simulation.

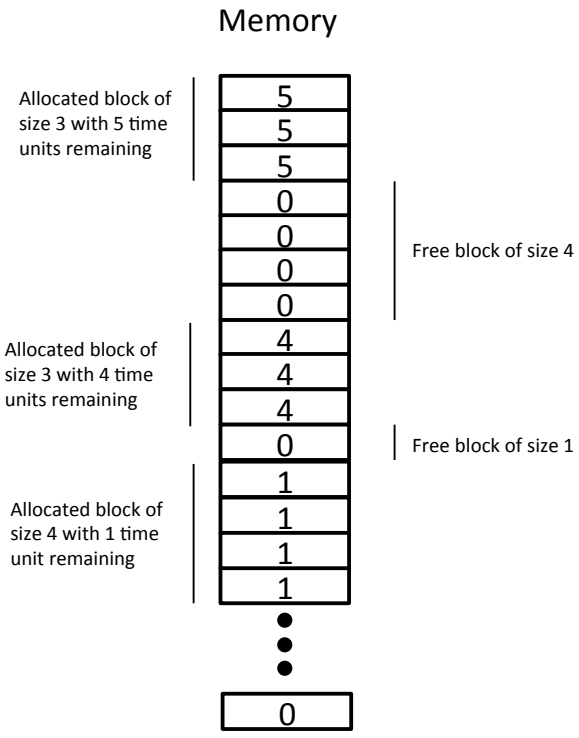

Figure 1: Example snap-shot of memory array at some point in time.

example snap shot of memory array

Collaboration and Plagiarism

This is an **individual assignment**, i.e. **no collaboration is permitted**. Plagiarism will not be tolerated. Submitted solutions that are very similar (determined by the instructor) will be given a grade of zero. Please do your own work, and everything will be OK.

Submission

Create a compressed tarball, i.e. *tar.gz*, that only contains the completed *main.c* , *mem.c*, and *Makefile*. The name of the compressed tarball must be your last name. For example, *ritchie.tar.gz* would be correct if the original co-developer of UNIX (Dennis Ritchie) submitted the assignment. Only assignments submitted in the correct format will be accepted (no exceptions). Submit the compressed tarball (via OAKS) to the Dropbox setup for this assignment. You may resubmit the compressed tarball as many times as you like, Dropbox will only keep the newest submission.

To be fair to everyone, late assignments will not be accepted. Exceptions will only be made for extenuating circumstances, i.e. death in the family, health related problems, etc. You will be given a week to complete this assignment. Poor time management is not excuse. Please do not email assignment after the due date, it will not be accepted. Please feel free to setup an appointment to discuss the assigned coding problem. I will be more than happy to listen to your approach and make suggestions. However, I cannot tell you how to code the solution. Additionally, code debugging is your job. You may use the debugger (gdb) or print statements to help understand why your solution is not working correctly, your choice.

Grading Rubric

For this assignment the grading rubric is provided in the table shown below.

Program Compiles	10 points
Program Runs with no errors	10 points
main() function implementation	15 points
mem_allocate() function implementation	20 points
mem_single_time_unit_transpired() function implementation	15 points
mem_fragment_count() function implementation	20 points
mem_clear() function implementation	10 points

In particular, the assignment will be graded as follows, if the submitted solution

- does not compile: 0 of 100 points
- compiles but does not run: 10 of 100 points
- compiles and runs with errors: 15 of 100 points
- compiles and runs without errors: 20 of 100 points
- All functions correctly implemented: 100 of 100 points