Program 1, TCSS 380 Spring 2019

OBJECTIVE

The objective of this assignment is to apply your newly learned C knowledge to build a slightly larger application.

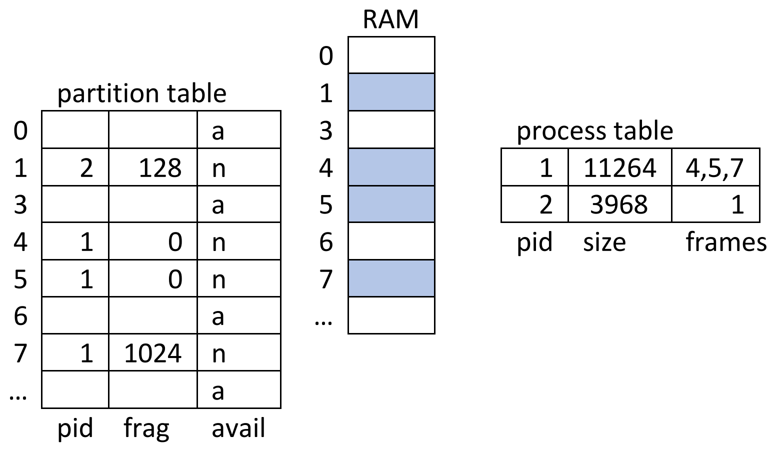
ASSIGNMENT DESCRIPTION

For this assignment you will write a program that models how the operating system keeps track of memory used by various processes. The general idea for our modeler is as follows: the memory, or RAM, is a number of contiguous memory locations and our operating system divides this memory into equal sized partitions. A process, depending on its size, may be assigned to one or more of these partitions. For example, if a partition size is 4096 bytes and the process size is 5000 bytes, then the process will occupy 2 partitions: one partition will be fully utilized, while the other will be underutilized, with only 904 bytes used. In our modeler, a partition cannot be used for more than one process, so in this case we are dealing with internal fragmentation, i.e. underutilization of one specific partition by 3192 bytes.

In order to keep track of all the currently running processes and where they are located in RAM, our OS keeps a couple of tables:

- the partition table keeps track of all partitions, whether they are available to new processes or not, and if they are not available, which process they belong to and the amount of internal fragmentation
- the process table that keeps track of all the processes and the frames they occupy

Each process in the program should be modeled by a structure that consists of a process id that will be generated by the program in a sequential fashion starting at a 1, and its size in bytes that will be entered by the user. Partition and process tables should be modeled by arrays of structures, where the structures should be defined in a way that makes it possible to store all the necessary information described above. The following diagram shows a conceptual picture of the model:



partition table

	pid	frag	avail
0			a
1	2	128	n
3			a
4	1	0	n
5	1	0	n
6			a
7	1	1024	n
…			a

process table

pid	size	frames
1	11264	4,5,7
2	3968	1

I/O: The input will be provided in an interactive fashion and the output should be generated to the screen. Take a look at a sample run for more details – user input is marked in bold and you should follow the exact same format (e.g. order, menu choices). You can assume that the user of your program will enter valid input, i.e. no error handling is required:

```
Enter the number of partitions:  10
Enter the size of each partition: 4096
Do you want to:
Add a process? Enter 1
Delete a process? Enter 2
Print values? Enter 3
Quit? Enter 4
```

```
1
Adding - enter process size:  5000
Do you want to:
Add a process? Enter 1
Delete a process? Enter 2
Print values? Enter 3
Quit? Enter 4
3
Printing –
--------------------------------------------------------------------------
Process table:
Process id          Frames
      1                   0, 1
--------------------------------------------------------------------------
Partition table:
Partition number Process id         Frag      Availability
0                1                  0         N
1                1                  3192      N
2                ?                  ?         Y
…
9                ?                  ?         Y
--------------------------------------------------------------------------
Do you want to:
Add a process? Enter 1
Delete a process? Enter 2
Print values? Enter 3
Quit? Enter 4
2
Deleting - enter process id:  1
Do you want to:
Add a process? Enter 1
Delete a process? Enter 2
Print values? Enter 3
Quit? Enter 4
4
```

PROGRAM SPECS

- The program should follow a multiple file structure (.h and .c files)
- The program should follow ADT principles (cohesion, information hiding, etc.) and use structures to define data types and their operations
- You are to provide a driver file that puts everything together – it should be named *driver.c*
- **You need to create a makefile for your code – the code will not be graded otherwise as there are different linking possibilities (should be called makefile)**
- Your executable file should be called *pr1.out*
- All program files must reside in one folder, called *pr1*, and the contents of the entire folder must be compressed using tar utility (instructions below)
- Your tar file should be called *pr1.tar*
- Your program has to follow basic stylistic features, such as proper indentation (use whitespaces, not tabs), meaningful variable names, etc.
- You need to comment your code
- You are not allowed to use global variables

- **Your program must be compatible with the required Linux VM or the cssgate server – programs that do not compile will receive a grade of 0.** Provide a comment at the top of *driver.c* that specifies which one we should use to run your code.

TAR INSTRUCTIONS

This part is to show you how to compress the entire *pr1* folder. To compress a folder, you have to be in a directory one level higher, i.e. if your *pr1* folder resides in *projects* folder, then to compress *pr1*, you have to be in *projects* folder.

```
tar -cf <name_of_a_new_tar_file> <directory_to_compress>
tar -cf pr1.tar pr1
```

To verify your tar file is correct before turning it in, move it to some other directory on your system and decompress

```
tar -xf <directory_to_decompress>
tar -xf pr1.tar
```

Verify you see another version of *pr1* folder with all its contents

EXTRA CREDIT

Extend the program in an interesting way or collect stats over multiple runs and add conclusions. For example, add a queue so that when RAM becomes full, new processes can wait to be scheduled in the queue. Generate stats into a csv file and analyze stats that show whether the partition size improves memory performance. If your extra credit results in more code, then you need to submit two files for grading: *pr1.tar* that contains the assignment and *pr1EC.tar* that contains the assignment with extra credit. At the top of *driver.c* explain what your extra credit is.

SUBMISSION AND GRADING NOTES

You are to submit your finished program through *Canvas*. This is an individual assignment – you are NOT allowed to work on it with any other student or share your code with others. The code will be graded based on its correctness by running it on instructor/grader designed test cases. Then, the code will be looked over for anything that violates good coding practices or the assignment specs (e.g. non-meaningful variable names, lack of code modularization). Note that you may turn in this assignment 7 days late, with no penalty. However, no submission past this deadline will be accepted even if you submit one minute after the deadline's timestamp – 7 days of lateness is more than enough.

Grading points will be distributed in the following fashion:

• Running program	35 pts
○ I/O	
○ Process table	
○ Frame table	
• Proper ADT modularization	10 pts
• Coding style and comments	5 pts
• Extra credit	5 pts