CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Project 2 (document version 1.4)
Contiguous and Non-contiguous Memory Management

Overview

- This project is due by 11:59:59 PM on Wednesday, December 13, 2017.
- This project will count as 10% of your final course grade.
- Note that students registered for CSCI 6140 will be required to submit additional work for this project, as described on page 14.
- This project is to be completed either individually or in a team of at most three students. Do not share your code with anyone else.
- You **must** use one of the following programming languages: C, C++, Java, or Python.
- Your code **must** successfully compile and run on Submitty, which uses Ubuntu v16.04.3 LTS.
- If you use C or C++, your program **must** successfully compile via `gcc` or `g++` with absolutely no warning messages when the `-Wall` (i.e., warn all) compiler option is used. We will also use `-Werror`, which will treat all warnings as critical errors.
- If you use Java, name your main Java file `Project2.java`.
- For Java and Python, be sure no warning messages occur during compilation and/or interpretation.
- The `gcc/g++` compiler is version 5.4.0 (`Ubuntu 5.4.0-6ubuntu1~16.04.4`). For source file naming conventions, be sure to use `*.c` for C or `*.cpp` for C++. In either case, you can also include `*.h` files.
- The `javac` compiler is version 8 (`javac 1.8.0_144`). Be sure to use `*.java` for your source file(s).
- For Python, you can use either `python2.7` or `python3.5`. Be sure to name your main Python file `project2.py`.

Project Specifications

In an operating system, each process has specific memory requirements. These memory requirements are met (or not) based on whether free memory is available to fulfill such requirements. In this second project, you will simulate both contiguous and non-contiguous memory allocation schemes. For contiguous memory allocation, you will use a dynamic memory allocation scheme.

Representing Physical (Main) Memory

For contiguous and non-contiguous memory schemes, to represent physical memory, use a data structure of your choice to represent a memory that contains a configurable number of equally sized "memory units" or frames.

As an example, memory may be represented as shown below (also see examples in the in-class notes). When you display your simulated memory, use the format below, showing exactly 32 frames per line. Also, as a default, simulate a memory consisting of 256 frames (i.e., eight output lines). These two values should be configurable and easily tunable.

```
================================
AAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBB.................
...................DDDDDDDDD....
................................
................................
........HHHHHHHHHHHHHHHHHHHHHHHH
HHH.............................
................................
================================
```

More specifically here, use `'.'` characters to represent free memory frames. Further, use ASCII characters in the range `'A'`-`'Z'` (i.e., uppercase letters only) to represent user process memory frames. Note that we will ignore operating system process memory frames.

Input File

For contiguous and non-contiguous memory schemes, details of user processes must be read from an input file, which is specified as the first command-line argument of your simulation. The input file is formatted as shown below. Any line beginning with a `#` character is ignored (these lines are comments). Further, all blank lines are also ignored, including lines containing only whitespace characters.

```
proc1 p1_mem p1_arr_time_1/p1_run_time_1 ... p1_arr_time_a/p1_run_time_a
proc2 p2_mem p2_arr_time_1/p2_run_time_1 ... p1_arr_time_b/p1_run_time_b
...
procN pN_mem pN_arr_time_1/pN_run_time_1 ... p1_arr_time_z/p1_run_time_z
```

Each `proc#` value specifies a single character in the range `'A'-'Z'` that uniquely identifies each given process. Further, each `p#_mem` value specifies the required (fixed) number of memory frames.

Each `p#_arr_time_?/p#_run_time_?` pair specifies a corresponding pair of arrival and run times for the given process. You may assume that each process has an increasing set of non-zero arrival times. You may also assume that each run time is greater than zero and does not overlap with the next arrival time. In other words, you do not need to validate these rules in your code.

Further, assume that the maximum number of processes in the simulation will be 26. You may also assume that processes will be given in alphabetical order.

Below is an example input file (note that values are delimited by one or more space or TAB characters):

```
A 45 0/350 400/50
B 28 0/2650
C 58 0/950 1100/100
D 86 0/650 1350/450
E 14 0/1400
F 24 100/380 500/475
G 13 435/815
J 46 550/900
etc.
```

For the contiguous memory management scheme, when defragmentation occurs (see below), these numbers must be automatically adjusted by extending all future arrival times accordingly. As an example, if defragmentation requires 300 time units, then all pending arrival times should increase by 300.

Also note that any "ties" that occur should be handled as follows. First, processes leave memory before other processes arrive. Second, further "ties" should be ordered using the alphabetical order of process IDs.

Contiguous Memory Management

For this portion of the simulation, each process's memory requirements must be met by the various contiguous memory allocation schemes that we have studied. The algorithms you must simulate are the *next-fit*, *first-fit*, and *best-fit* algorithms, each of which is described in more detail below.

Note that we will only use a dynamic partitioning scheme here, meaning that your data structure(s) must maintain a list containing (1) where each process is allocated, (2) how much contiguous memory each process uses, and (3) where and how much free memory is available (i.e., where each free partition is).

As an example, consider the simulated memory shown below.

```
================================
AAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBB.................
...................DDDDDDDDD....
................................
................................
........HHHHHHHHHHHHHHHHHHHHHHHH
HHH.............................
................................
================================
```

In the above example diagram, four processes are allocated in four dynamically allocated partitions. Further, there are three free partitions (i.e., between processes `B` and `D`, between processes `D` and `H`, and between process `H` and the "bottom" of memory.

Placement Algorithms

When a process `Q` wishes to enter the system, memory must first be allocated dynamically. More specifically, a dynamic partition must be created out of an existing free partition.

For the *next-fit* algorithm, process `Q` is placed in the first free partition available found by scanning from the memory unit just beyond the end of the most recently placed process.

For the *first-fit* algorithm, process `Q` is placed in the first free partition available found by scanning from the "top" of memory.

For the *best-fit* algorithm, process `Q` is placed in the smallest free partition available in which process `Q` fits. If a "tie" occurs, use the free partition closer to the "top" of memory.

For all of these placement algorithms, the memory scan covers the entire memory. If necessary (i.e., if the scan hits the "bottom" of memory), the scan continues from the "top" of memory.

If no suitable free partition is available, then an out-of-memory error occurs, at which point defragmentation may be required.

Defragmentation

If a process is unable to be placed into memory, defragmentation occurs if the overall total free memory is sufficient to fit the given process. In such cases, processes are relocated as necessary, starting from the top of memory. If instead there is not enough memory to admit the given process, your simulation must skip defragmentation and deny the request and move on to the next process.

Note that the given process is only skipped for the given requested interval. It may be admitted to the system at a later interval. For example, given process `Q` below, if the process is unable to be added in interval `200-580`, it might still be successfully added in interval `720-975`.

```
Q 47 200/380 720/255
```

Once defragmentation has started, it will run through to completion, at which point, the process that triggered defragmentation will be admitted to the system.

While defragmentation is running, no other processes may be executing (i.e., all processes are essentially in a suspended state) until all relocations are complete. Therefore, process arrivals and exits are suspended during defragmentation.

Note that the time to move **one** frame of memory is defined as `t_memmove` and is measured in milliseconds. Assume that the default value of `t_memmove` is `1`.

Given the memory shown previously, the results of defragmentation will be as follows (i.e., processes `D` and `H` are moved upwards in memory):

```
================================
AAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBDDDDDDDDDHHHHHHHH
HHHHHHHHHHHHHHHHHHH.............
................................
................................
................................
................................
................................
================================
```

Note that for the *next-fit* algorithm, after defragmentation completes, you should reset the last-placed pointer to the memory unit after the last process in memory. In the example above, process `Q` would be placed directly after process `H`.

Simulation and Output Requirements

For the given input file, simulate each of the three placement algorithms, displaying output for each "interesting" event that occurs. Interesting events are:

- Simulation start
- Process arrival
- Process placement in physical memory
- Process exit from physical memory
- Start of defragmentation
- End of defragmentation (i.e., process(es) successfully moved)
- Simulation end

As with previous projects, your simulator must keep track of elapsed time `t` (measured in milliseconds), which is initially set to zero. As your simulation proceeds, based on the input file, `t` advances to each "interesting" event that occurs, displaying a specific line of output describing each event. Further, reset your simulation for each of the placement algorithms.

Note that your simulator output should be entirely deterministic. To achieve this, your simulator must output each "interesting" event that occurs using the format shown in the examples further below. **(v1.4)** ~~In general, when memory changes, you must display the full simulated memory. Conversely, if memory does not change (e.g., a process is skipped), do not display the simulated memory.~~ To simplify processing, please display memory regardless of whether a process is placed or skipped.

Given the example input file at the bottom of page 3 and default configuration parameters, your simulator output would be as follows:

```
time 0ms: Simulator started (Contiguous -- Next-Fit)
time 0ms: Process A arrived (requires 45 frames)
time 0ms: Placed process A:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA...................
................................
................................
................................
................................
................................
................................
================================
time 0ms: Process B arrived (requires 28 frames)
time 0ms: Placed process B:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBBB
BBBBBBBBB.......................
................................
................................
................................
................................
................................
================================
time 0ms: Process C arrived (requires 58 frames)
time 0ms: Placed process C:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCC.............................
................................
................................
................................
================================
time 0ms: Process D arrived (requires 86 frames)
time 0ms: Placed process D:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDD.......
................................
================================
time 0ms: Process E arrived (requires 14 frames)
time 0ms: Placed process E:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEE.........................
================================
time 100ms: Process F arrived (requires 24 frames)
time 100ms: Placed process F:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEEFFFFFFFFFFFFFFFFFFFFFFFF.
================================
time 350ms: Process A removed:
================================
................................
.............BBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEEFFFFFFFFFFFFFFFFFFFFFFFF.
================================
time 400ms: Process A arrived (requires 45 frames)
time 400ms: Placed process A:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEEFFFFFFFFFFFFFFFFFFFFFFFF.
================================
time 435ms: Process G arrived (requires 13 frames)
time 435ms: Cannot place process G -- skipped!
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEEFFFFFFFFFFFFFFFFFFFFFFFF.
================================
time 450ms: Process A removed:
================================
................................
.............BBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEEFFFFFFFFFFFFFFFFFFFFFFFF.
================================
time 480ms: Process F removed:
================================
................................
.............BBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEE.........................
================================
time 500ms: Process F arrived (requires 24 frames)
time 500ms: Placed process F:
================================
................................
.............BBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEEFFFFFFFFFFFFFFFFFFFFFFFF.
================================
time 550ms: Process J arrived (requires 46 frames)
time 550ms: Cannot place process J -- starting defragmentation
time 760ms: Defragmentation complete (moved 210 frames: B, C, D, E, F)
================================
BBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDEEEEEEEEEEEEEEFFFFFF
FFFFFFFFFFFFFFFFFF..............
................................
================================
time 760ms: Placed process J:
================================
BBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDEEEEEEEEEEEEEEFFFFFF
FFFFFFFFFFFFFFFFFFJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
================================
time 860ms: Process D removed:
================================
BBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCC..........
................................
................................
............EEEEEEEEEEEEEEFFFFFF
FFFFFFFFFFFFFFFFFFJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
================================
...
time ####ms: Simulator ended (Contiguous -- Next-Fit)
time 0ms: Simulator started (Contiguous -- First-Fit)
time 0ms: Process A arrived (requires 45 frames)
time 0ms: Placed process A:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA...................
................................
................................
................................
................................
................................
................................
================================
...
time ####ms: Simulator ended (Contiguous -- First-Fit)

time 0ms: Simulator started (Contiguous -- Best-Fit)
time 0ms: Process A arrived (requires 45 frames)
time 0ms: Placed process A:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA...................
................................
................................
................................
................................
................................
................................
================================
...
time ####ms: Simulator ended (Contiguous -- Best-Fit)
```

Note that a blank line separates the above sets of simulation output, i.e., between each pair of "Simulation end" and "Simulation start" event output lines.

Non-contiguous Memory Management

Extend the above contiguous memory management simulation by next simulating a non-contiguous memory management scheme in which you use a page table to map each logical page of a process to a physical frame of memory. The key difference here is that defragmentation is no longer necessary.

To place pages into frames of physical memory, use a simple first-fit approach, as shown in the example below. Note that when you display the page table, display processes in alphabetical order; further, display at most 10 page-to-frame mappings per line.

```
time 0ms: Simulator started (Non-contiguous)
time 0ms: Process A arrived (requires 45 frames)
time 0ms: Placed process A:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA...................
................................
................................
................................
................................
................................
................................
================================
PAGE TABLE [page,frame]:
A: [0,0] [1,1] [2,2] [3,3] [4,4] [5,5] [6,6] [7,7] [8,8] [9,9]
[10,10] [11,11] [12,12] [13,13] [14,14] [15,15] [16,16] [17,17] [18,18] [19,19]
[20,20] [21,21] [22,22] [23,23] [24,24] [25,25] [26,26] [27,27] [28,28] [29,29]
[30,30] [31,31] [32,32] [33,33] [34,34] [35,35] [36,36] [37,37] [38,38] [39,39]
[40,40] [41,41] [42,42] [43,43] [44,44]
...
time 500ms: Process F arrived (requires 24 frames)
time 500ms: Placed process F:
================================
FFFFFFFFFFFFFFFFFFFFFFFF........
.............BBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEE.........................
================================
PAGE TABLE [page,frame]:
B: [0,45] [1,46] [2,47] [3,48] [4,49] [5,50] [6,51] [7,52] [8,53] [9,54]
[10,55] [11,56] [12,57] [13,58] [14,59] [15,60] [16,61] [17,62] [18,63] [19,64]
[20,65] [21,66] [22,67] [23,68] [24,69] [25,70] [26,71] [27,72]
C: [0,73] [1,74] [2,75] ...
```

```
D: [0,131] [1,132] [2,133] ...
E: [0,217] [1,218] [2,219] ...
F: [0,0] [1,1] [2,2] ...
time 550ms: Process J arrived (requires 46 frames)
time 550ms: Placed process J:
================================
FFFFFFFFFFFFFFFFFFFFFFFFJJJJJJJJ
JJJJJJJJJJJJJBBBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEE
EEEEEEEJJJJJJJJJJJJJJJJJJJJJJJJJ
================================
PAGE TABLE [page,frame]:
B: [0,45] [1,46] [2,47] [3,48] [4,49] [5,50] [6,51] [7,52] [8,53] [9,54]
[10,55] [11,56] [12,57] [13,58] [14,59] [15,60] [16,61] [17,62] [18,63] [19,64]
[20,65] [21,66] [22,67] [23,68] [24,69] [25,70] [26,71] [27,72]
C: [0,73] [1,74] [2,75] ...
D: [0,131] [1,132] [2,133] ...
E: [0,217] [1,218] [2,219] ...
F: [0,0] [1,1] [2,2] ...
J: [0,24] [1,25] [2,26] ...
...
time ####ms: Simulator ended (Non-contiguous)
```

Submission Instructions

To submit your assignment (and also perform final testing of your code), please use Submitty, the homework submission server. The URL is on the course website.

If you are submitting a team project, please have each team member submit the same submission (to be sure everyone gets a grade). Also be sure to include all names and RCS IDs in comments at the top of each source file.

Graduate Section Requirements

For students registered for CSCI 6140, additional analysis is required. Please answer the questions below and submit as a PDF file called `project2-analysis.pdf`. Use a paragraph to answer each question.

Note that each student registered for CSCI 6140 must write up his or her own answers (even if you are working on a team).

And undergraduates are encouraged to review the questions and think of answers here, but not required to submit your answers.

1. What are the advantages and disadvantages of each of the four algorithms covered in this assignment?
2. Which of the three contiguous memory schemes is actually the best? How do you quantify and support your choice?
3. Which of the three contiguous memory schemes is the worst? How do you quantify and support your choice?
4. Are there variations to these algorithms that you can come up with to address the various disadvantages of these algorithms?