CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Project 2 (document version 1.0)
Memory Management Simulation

Overview

- This project is due by 11:59:59 PM on Tuesday, April 26, 2019 with four extra late days built in.
- This project is to be completed either individually or in a team of at most three students. Teams may consist of both undergraduate and graduate students. Do not share your code with anyone else.
- Note that students registered for CSCI 6140 will be required to submit additional work for this project, as described on page 14.
- You **must** use one of the following programming languages: C, C++, Java, or Python.
- As per usual, your code **must** successfully compile/run on Submitty, which uses Ubuntu v18.04.1 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.
- Note that the `gcc`/`g++` compiler is version 7.3.0 (`Ubuntu 7.3.0-27ubuntu1~18.04`). 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.
- If you use Java, name your main Java file `Project2.java`. And note that the `javac` compiler is version 8 (`javac 1.8.0_191`).
- For Python, you must use `python3`, which is Python 3.6.7. Be sure to name your main Python file `project2.py`.
- For Java and Python, be sure no warning messages occur during compilation/interpretation. Also, please "flatten" all directory structures to a single directory of source files; and for Java, do not use the `package` directive.

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 implement a dynamic partitioning scheme.

Representing physical (main) memory

For both contiguous and non-contiguous memory schemes, to represent physical memory, use any 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 using the format below, which shows 32 frames per line and 256 frames (i.e., eight output lines). These two values (i.e., 32 and 256) will be given as the first two command-line arguments to your simulation.

```
================================
AAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBB..................
..................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 IDs. 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; the filename is specified as the third command-line argument. This input file is formatted as shown below. And 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.

```
pid1 p1_mem p1_arr_time_1/p1_run_time_1 ... p1_arr_time_a/p1_run_time_a
pid2 p2_mem p2_arr_time_1/p2_run_time_1 ... p2_arr_time_b/p2_run_time_b
...
pidN pN_mem pN_arr_time_1/pN_run_time_1 ... pN_arr_time_z/pN_run_time_z
```

Here, each `p#_mem` value specifies the required (fixed) number of memory frames for that process. And each `p#_arr_time_?/p#_run_time_?` pair specifies a corresponding pair of arrival and run times for the given process, in milliseconds. 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.

You may assume that processes will be given in alphabetical order in the input file. Below is an example input file (note that all 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
H 14 0/1400
I 24 100/380 500/475
J 13 435/815
Q 46 550/900
etc.
```

To simulate a 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.

Note that processes leave memory before other processes arrive. Further, any "ties" that occur should be resolved by 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. Of the four algorithms we have studied, you must simulate the *first-fit*, *next-fit*, and *best-fit* algorithms, each of which is described in more detail below.

Note that we will only use a dynamic partitioning scheme, 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.

```
================================
AAAAAAAABBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBB..................
..................DDDDDDDDD....
................................
................................
........HHHHHHHHHHHHHHHHHHHHHHH
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 `X` wishes to enter the system, memory must first be allocated dynamically. More specifically, a dynamic partition must be created from an existing free partition.

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

For the *next-fit* algorithm, process `X` 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, looping back to the "top" of memory if necessary.

For the *best-fit* algorithm, process `X` is placed in the smallest free partition available in which process `X` 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.

And if no suitable free partition is available, then an out-of-memory error occurs, at which point defragmentation might occur (see below).

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, 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 given as a command-line argument.

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

```
================================
AAAAAAAABBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBDDDDDDDDDHHHHHHHHH
HHHHHHHHHHHHHHHHHH..............
................................
................................
................................
................................
................................
================================
```

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`.

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.

And to place pages into frames of physical memory, use a simple first-fit approach.

Simulation configuration

As with Project 1, the key to designing a useful simulation is to provide a number of configurable parameters to the user. This allows you to simulate and tune for a variety of scenarios. Therefore, define the following simulation parameters as tunable constants within your code, all of which will be given as command-line arguments:

- `argv[1]`: The first command-line argument specifies the number of frames to show on a line. The examples show 32 frames per line. Note that this value might not be a power of two.
- `argv[2]`: The second command-line argument specifies the size of the memory, i.e., how many frames make up the physical memory. The examples show a memory consisting of 256 frames. Note that this value might not be a power of two.
- `argv[3]`: The third command-line argument specifies the name of the input file to read in for your simulation.
- `argv[4]`: The fourth command-line argument defines $t_{memmove}$, which is the time, in milliseconds, that it takes to move **one** frame of memory during defragmentation.

Simulation and output requirements

For the given input file, simulate each of the three placement algorithms, as well as the non-contiguous memory allocation scheme, 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
- Simulation end

As with the previous project, 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 to describe each event. Be sure to 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. 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.

Given the example input file at the bottom of page 3, your simulator output would be as follows, with a blank line separating each set of simulation output, i.e., between each pair of "Simulation end" and "Simulation start" event output lines.

```
bash$ ./a.out 32 256 sample.txt 1
time 0ms: Simulator started (Contiguous -- First-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:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBB
BBBBBBBBB.....................
..............................
..............................
..............................
..............................
..............................
================================
time 0ms: Process C arrived (requires 58 frames)
time 0ms: Placed process C:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCC...........................
..............................
..............................
..............................
================================
time 0ms: Process D arrived (requires 86 frames)
time 0ms: Placed process D:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDD.......
..............................
================================
time 0ms: Process E arrived (requires 14 frames)
time 0ms: Placed process H:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDHHHHHHH
HHHHHHH.......................
================================
```

```
time 100ms: Process I arrived (requires 24 frames)
time 100ms: Placed process I:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDHHHHHHH
HHHHHHHIIIIIIIIIIIIIIIIIIIIIII.
================================
time 350ms: Process A removed:
================================
................................
............BBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDHHHHHHH
HHHHHHHIIIIIIIIIIIIIIIIIIIIIII.
================================
time 400ms: Process A arrived (requires 45 frames)
time 400ms: Placed process A:
================================
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDHHHHHHH
HHHHHHHIIIIIIIIIIIIIIIIIIIIIII.
================================
```

```
time 435ms: Process J arrived (requires 13 frames)
time 435ms: Cannot place process J -- skipped!
time 450ms: Process A removed:
================================
................................
.............BBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDHHHHHH
HHHHHHHIIIIIIIIIIIIIIIIIIIIIII.
================================
time 480ms: Process I removed:
================================
................................
.............BBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDHHHHHH
HHHHHHH.........................
================================
time 500ms: Process I arrived (requires 24 frames)
time 500ms: Placed process I:
================================
IIIIIIIIIIIIIIIIIIIIIIII........
.............BBBBBBBBBBBBBBBBBB
BBBBBBBBBCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDHHHHHH
HHHHHHH.........................
================================
```

```
time 550ms: Process Q arrived (requires 46 frames)
time 550ms: Cannot place process Q -- starting defragmentation
time 736ms: Defragmentation complete (moved 186 frames: B, C, D, H)
time 736ms: Placed process Q:
================================
IIIIIIIIIIIIIIIIIIIIIIIBBBBBBBB
BBBBBBBBBBBBBBBBBBBCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDHHHHHHHHHHHHHHQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
================================
time 836ms: Process D removed:
================================
FFFFFFFFFFFFFFFFFFFFFFFBBBBBBBB
BBBBBBBBBBBBBBBBBBBCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCC.................
.............................
.............................
....HHHHHHHHHHHHHHQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
================================
...
time ####ms: Simulator ended (Contiguous -- First-Fit)
```

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

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

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

Error handling

If improper command-line arguments are given, report an error message to `stderr` and abort further program execution. In general, if an error is encountered, display a meaningful error message on `stderr`, then abort further program execution.

Error messages must be one line only and use the following format:

```
ERROR: <error-text-here>
```

Submission instructions

To submit your assignment (and also perform final testing of your code), please use Submitty, the homework submission server. Also use Submitty to define teams, including a team of one if you decide to work alone on this project.

Be sure to include all names and RCS IDs in comments at the top of each source file submitted.

Note that this assignment will be available on Submitty a minimum of three days before the due date. Please do not ask on Piazza when Submitty will be available, as you should perform adequate testing on your own Ubuntu platform.

That said, to make sure that your program does execute properly everywhere, including Submitty, consider using the techniques below.

First, as discussed in class (on 1/10), use the `DEBUG_MODE` technique to make sure you do not submit any debugging code. Here is an example in C:

```
#ifdef DEBUG_MODE
    printf( "the value of x is %d\n", x );
    printf( "the value of q is %d\n", q );
    printf( "why is my program crashing here?!" );
    fflush( stdout );
#endif
```

And to compile this code in "debug" mode, use the `-D` flag as follows:

```
bash$ gcc -Wall -Werror -D DEBUG_MODE p2.c
```

Second, as discussed in class (on 1/14), output to standard output (`stdout`) is buffered. To disable buffered output for grading on Submitty, use `setvbuf()` as follows in C:

```
setvbuf( stdout, NULL, _IONBF, 0 );
```

You would not generally do this in practice, as this can substantially slow down your program, but to ensure correctness on Submitty, this is a good technique to use.

Relinquishing allocated resources

Be sure that all resources (e.g., dynamically allocated memory) are properly relinquished for whatever language/platform you use for this assignment. Sloppy programming will potentially lead to grading penalties. Consider doing frequent code reviews with your teammates.

Graduate section requirements

For students registered for CSCI 6140, additional analysis is required.

Please answer the questions below by submitting a PDF file called `<userid>-p2-analysis.pdf` (e.g., `goldsd3-p2-analysis.pdf`). Answer all questions below in no more than five pages.

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

1. What are the advantages and disadvantages of each of the four algorithms covered in this assignment? Support your answer by citing specific simulation results.
2. Of the three simulated algorithms, which algorithm is the "best" algorithm? Support your answer by citing specific simulation results.
3. Describe some variations to these algorithms that you can come up with to address the various disadvantages of these algorithms.
4. Describe at least three limitations of your simulation, in particular how the project specifications could be expanded to better model a real-world operating system.

If you are registered for CSCI 4210, feel free to review these questions, but do not submit an analysis on Submitty.