CS431 — Project 3

February 13, 2017

Due: Friday, February 24, 2017 (60 points)

Description

In this project you will be implementing swapping as described in the lectures using a linked list to track free/occupied memory.

Your program should compile and run with just the following commands:

```
$ javac SwapTest.java
$ java SwapTest
```

Linked List

Your program should contain the following class:

```
public final class Segment {

    private int pid;
    private int start;
    private int length;
    private Segment next;

    public Segment(int pid, int start, int length, Segment next) {
        this.pid = pid;
        this.start = start;
        this.length = length;
        this.next = next;
    }

    public int getPid() {
        return pid;
    }

    public int getStart() {
        return start;
    }

    public int getLength() {
        return length;
    }
    public Segment getNext() {
        return next;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

    public void setStart(int start) {
        this.start = start;
    }

    public void setLength(int length) {
        this.length = length;
    }

    public void setNext(Segment next) {
        this.next = next;
    }

    @Override
    public String toString() {
        return String.format("(%d %d %d)", pid, start, length);
    }
}
```

This class will act as the linked list of segments that track free and used memory. The pid variable corresponds to the id of the process in a given segment. If the value of pid is 0, this segment is a hole. Process ids should start from 1. The start variable corresponds to the address where this segment begins. The length variable corresponds to the length of the segment. The next variable corresponds to the next node in the list. Initially, your program should have a single Segment named `start` corresponding to the beginning of the list. Do not use a premade Linked List class such as the one in the Java collections API.

Jobs

A job or process in this project will contain two things: a pid and a size. You can use this class to represent jobs:

```
public final class Job {

    private final int pid;
    private final int size;

    public Job(int pid, int size) {
        this.pid = pid;
        this.size = size;
    }

    public int getPid() {
        return pid;
    }

    public int getSize() {
        return size;
    }

    @Override
    public String toString() {
        return String.format("[%d %d]", pid, size);
    }
}
```

Requirements

Your program will maintain the linked list of segments and a list of current existing jobs on the system. The list of existing jobs are not necessarily the ones currently in memory, it is simply the list of all jobs on the system.

Your program must implement the following:

1. First fit allocation: given a job, your program should perform the first fit algorithm to allocate space for the job in memory by adjusting the linked list as necessary. Return true if the allocation was successful and false otherwise.
2. Next fit allocation: given a job, your program should perform the next fit algorithm to allocate space for the job in memory by adjusting the linked list as necessary. Return true if the allocation was successful and false otherwise.
3. Best fit allocation: given a job, your program should perform the best fit algorithm to allocate space for the job in memory by adjusting the linked list as necessary. Return true if the allocation was successful and false otherwise.
4. Worst fit allocation: given a job, your program should perform the worst fit algorithm to allocate space for the job in memory by adjusting the linked list as necessary. Return true if the allocation was successful and false otherwise.
5. Deallocation: given a pid, your program should correctly deallocate the segment for that job. Remember, there are four possibilities here plus the two special cases of being either at the end or beginning of the list!

At the beginning the list of jobs should be empty and the linked list should have a single "hole" segment of size 100.

Interface

Your program should provide an interactive command line interface allowing the user to enter the following commands:

1. `add n s`

   This command adds job number `n` of size `s` to the list of jobs on the system. Both `n` and `s` represent integers.

2. `jobs`

   This command prints the list of jobs on the system.

3. `list`

   This command prints the current contents of the linked list of segments.

4. `ff n`

   This command invokes the first fit allocation for job number `n`.

5. `nf n`

   This command invokes the next fit allocation for job number `n`.

6. `bf n`

   This command invokes the best fit allocation for job number `n`.

7. `wf n`

   This command invokes the worst fit allocation for job number `n`.

8. `de n`

   This command invokes deallocation for job number `n`.

Note: your program should include proper error handling. If a job is already allocated, it should not be allocated again. If a job is not allocated, attempting to deallocate it should be handled gracefully (do not crash the program). Adding a job with a pid that already exists on the system should not be allowed.

If you want, you can extend the commands above such as allowing a command like:

```
ff 1 2 3
```

This would sequentially attempt to allocate first job 1, then job 2, then job 3 by first fit.

Sample Output

Every line beginning with a `>` is a prompt with the user's command after.

```
> jobs

> list
(0 0 100)
> add 1 5
> jobs
[1 5]
> ff 1
> list
(1 0 5) (0 5 95)
> add 2 17
> ff 2
> list
(1 0 5) (2 5 17) (0 22 78)
> de 2
> list
(1 0 5) (0 5 95)
```

```
> ff 2
> de 1
> list
(0 0 5) (2 5 17) (0 22 78)
> add 3 7
> ff 3
> list
(0 0 5) (2 5 17) (3 22 7) (0 29 71)
> add 4 4
> ff 4
> list
(4 0 4) (0 4 1) (2 5 17) (3 22 7) (0 29 71)
> de 2
> list
(4 0 4) (0 4 18) (3 22 7) (0 29 71)
>
```

Alternate sample output:

```
> add 1 5
> add 2 4
> add 3 3
> add 4 2
> add 5 1
> bf 1
> bf 2
> bf 3
> bf 4
> bf 5
> list
(1 0 5) (2 5 4) (3 9 3) (4 12 2) (5 14 1) (0 15 85)
> de 5
> de 3
> de 1
> list
(0 0 5) (2 5 4) (0 9 3) (4 12 2) (0 14 86)
> bf 5
> list
(0 0 5) (2 5 4) (5 9 1) (0 10 2) (4 12 2) (0 14 86)
>
```

Submission

1. On `https://codebank.xyz`, create a project named `CS431-P3`. Follow this naming convention precisely including case.
2. Create your own local repository by the following:
   (a) Navigate to your local directory for this project.
   (b) Run `git init` to initialize the repository.
3. Make sure you add the reference to the remote repository in your local repository with:

```
$ git remote add origin https://codebank.xyz/username/CS431-P3.git
```

4. Your project should have a main class named `SwapTest` in a file named `SwapTest.java`. You can have other files or classes, but it should successfully compile and run by simplying using:

   ```
   $ javac SwapTest.java
   $ java SwapTest
   ```

You will lose points if your submission is not correct (e.g., incorrect repository name, file names, class names, or package declaration that causes the above commands to fail to run).