Programming Problems

9.28 Assume that a system has a 32-bit virtual address with a 4-KB page size. Write a C program that is passed a virtual address (in decimal) on the command line and have it output the page number and offset for the given address. As an example, your program would run as follows:

```
./addresses 19986
```

Your program would output:

```
The address 19986 contains:
page number = 4
offset = 3602
```

Writing this program will require using the appropriate data type to store 32 bits. We encourage you to use `unsigned` data types as well.

Programming Projects

Contiguous Memory Allocation

In Section 9.2, we presented different algorithms for contiguous memory allocation. This project will involve managing a contiguous region of memory of size *MAX* where addresses may range from 0 ... *MAX* – 1. Your program must respond to four different requests:

1. Request for a contiguous block of memory
2. Release of a contiguous block of memory
3. Compact unused holes of memory into one single block
4. Report the regions of free and allocated memory

Your program will be passed the initial amount of memory at startup. For example, the following initializes the program with 1 MB (1,048,576 bytes) of memory:

```
./allocator 1048576
```

Once your program has started, it will present the user with the following prompt:

```
allocator>
```

It will then respond to the following commands: `RQ` (request), `RL` (release), `C` (compact), `STAT` (status report), and `X` (exit).

A request for 40,000 bytes will appear as follows:

```
allocator>RQ P0 40000 W
```

The first parameter to the RQ command is the new process that requires the memory, followed by the amount of memory being requested, and finally the strategy. (In this situation, "W" refers to worst fit.)

Similarly, a release will appear as:

```
allocator>RL P0
```

This command will release the memory that has been allocated to process P0.

The command for compaction is entered as:

```
allocator>C
```

This command will compact unused holes of memory into one region.

Finally, the STAT command for reporting the status of memory is entered as:

```
allocator>STAT
```

Given this command, your program will report the regions of memory that are allocated and the regions that are unused. For example, one possible arrangement of memory allocation would be as follows:

```
Addresses [0:315000] Process P1
Addresses [315001: 512500] Process P3
Addresses [512501:625575] Unused
Addresses [625575:725100] Process P6
Addresses [725001] . . .
```

Allocating Memory

Your program will allocate memory using one of the three approaches highlighted in Section 9.2.2, depending on the flag that is passed to the RQ command. The flags are:

- F—first fit
- B—best fit
- W—worst fit

This will require that your program keep track of the different holes representing available memory. When a request for memory arrives, it will allocate the memory from one of the available holes based on the allocation strategy. If there is insufficient memory to allocate to a request, it will output an error message and reject the request.

Your program will also need to keep track of which region of memory has been allocated to which process. This is necessary to support the STAT command and is also needed when memory is released via the RL command, as the process releasing memory is passed to this command. If a partition being released is adjacent to an existing hole, be sure to combine the two holes into a single hole.

Compaction

If the user enters the `C` command, your program will compact the set of holes into one larger hole. For example, if you have four separate holes of size 550 KB, 375 KB, 1,900 KB, and 4,500 KB, your program will combine these four holes into one large hole of size 7,325 KB.

There are several strategies for implementing compaction, one of which is suggested in Section 9.2.3. Be sure to update the beginning address of any processes that have been affected by compaction.