Lab and Homework #5

Introduction to Operating Systems CS-UY 3224 | CS-UY 3224G

Mirna Džamonja, email md5961@nyu.edu

Due date for the Homework problems : October 9nd, 2023 by 5 PM, Paris time

Please hand in through the *Assignments* option on *Brightspace*.

Question 1: *Knowing your system.*

1. Check the page size of pages in the memory of your computer by typing

   `getconf PAGESIZE`.

   (The response is a number in bytes.) Familiarise yourself with `getconf` by reading its `man` page.

2. Why are the page size and the size of page in paging systems always powers of 2 ?

To do at home and hand in: The answer to item (2).

Question 2: *A bit of calculation.*

Assuming a 1KB page size, what are the page numbers and offsets for the following virtual address references, which are given in the decimal form:

1. 3085
2. 42095
3. 215201
4. 650000
5. 2000001 ?

To do at home and hand in: Your answers.

Question 3: *A bit of C.*

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.

To do at home and hand in: Your program.

Question 4: *Let's program an allocator !*

For this allocator we shall use the **best fit** strategy.

This program involves managing a contiguous region of memory of size m passed to the program at the beginning, where addresses may range from $0, \ldots, m-1$. Your program must respond to four different types of requests:

1. Request for a contiguous block of memory
2. Release; that is free, 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

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
```

The first parameter to the RQ command is the new process that requires the memory, followed by the amount of memory being requested.

When a request for memory arrives, allocator will allocate the memory from one of the available free spaces on your given slot of 1MB based on the best fit allocation strategy. If there is insufficient memory to allocate to a request, it will output an error message and reject the request.

At that point you can attempt compaction. As opposed to the ad hoc approach used in Homework 4 when we were doing this by hand, to program an allocator, we need an algorithm. The simplest compaction algorithm is to move all processes toward one end of memory; all free spaces move in the other direction, producing one large space of available memory.

The command for compaction is entered as:

```
allocator>C
```

A release will appear as:

```
allocator>RL P0
```

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

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] . . .

To do at home and hand in: The C program.