Operating Systems Lab (CS330-2025)
Lab #4

General Instructions

- *Switch off* all electronic devices during the lab and store it in your bags.
- Read each part *carefully* to know the restrictions imposed for each question.
- *Do not modify* any of the test cases or evaluation scripts. Modify only the file(s) mentioned.
- Please take the help of the teaching staff, if you face any issues.
- Best of Luck!

Know your environment!

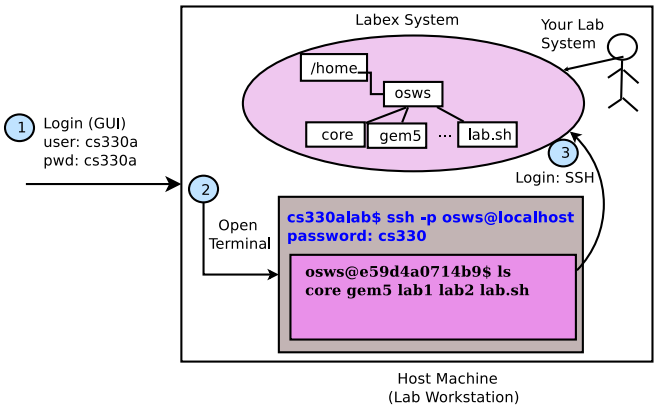

Figure 1: Overview of the Lab environment

When you login into the lab machine, you get a GUI desktop environment. Let us call this the "Host" machine. The lab exercise environment is in a separate sandbox environment in the same machine which you can treat as a separate computer system. Let us call this the "Labex" system. The host machine *is not* the lab setup, rather it hosts the lab exercise system. You may use the host machine to access `man` pages and open exercise document using GUI applications such as the PDF viewer.

Interacting with the "Labex" system

Login into the Labex system

1. Open the terminal application on the Host system. You can also use 'ALT+CTRL+t' and 'ALT+CTRL+n' to open a new tab in or a new terminal from an existing terminal. *Note that, this shell in this terminal refers to the Host system.*
2. On the terminal, execute `ssh -p 2020 osws@localhost`. It will prompt for the password. Once you have entered the password, you will get the shell to the Labex system.
3. Check the terminal heading or the shell prompt. If you see `osws@`, you are in the Labex system whereas if you see `cs330a@`, you are in the host system.

Exchange files between the Host and Labex systems

1. Open the terminal application on the Host system. Change your directory to "Desktop" (using `cd Desktop` if you want to download/transfer files stored in the host system Desktop.
2. To download `CS330-2025-Lab4.pdf` from Labex to the current directory on the host, run `scp -P 2020 -r osws@localhost:lab-4/labex/CS330-2025-Lab4.pdf ./`. It will prompt for the password. Once you have entered the password, you will see that the file is downloaded.
3. To upload 'abc.c' from the current directory in the host to the Labex system lab-3/labex directory, execute `scp -P 2020 -r abc.c osws@localhost:lab-4/labex/`. It will prompt for the password. Once you have entered the password, you will see that the file is uploaded.

Lab Actions

The current working directory *must be* the home directory of the Labex environment to execute different lab actions through the `lab.sh` utility. Executing `cd /home/osws` or simply `cd`) will take you to the home directory of the Labex environment. The usage semantic of `lab.sh` script is shown below.

```
USE ./lab.sh to initialize the session and get started.

usage:

./lab.sh --roll|-r  <roll1_roll2> --labnum|-n <lab number> --action|-a <init|get|evaluate|
                                                                        prepare|prepare-save|
                                                                        submit|save|reload|
                                                                        detach|swupdate|
                                                                        signoff>


Note your roll number string. Never forget or forge!

        [--action] can be one of the following

        init: Initialize the lab session
        get: Download the assignment
        evaluate: Evaluate the assignment
        prepare: Prepare a submission archive
        prepare-save: Prepare an archive to save
        submit: Submit the assignment. Can be perfomed only once!
        save: Save the assignment
        reload: Reload the last saved solution and apply it to a fresh lab archive
        detach: The lab session is detached. Can be reloaded using 'reload', if supported
        swupdate: Peform software update activities. Caution: Use only if instructed
        signoff: You are done for the lab session. Caution: After signoff, you will not be
                 allowed to submit anymore

EXAMPLE
=======
Assume that your group members have the roll nos 210010 and 211101.
Every lab will have a lab number (announced by the TAs).
Assume lab no to be 5 for the examples shown below.
```

```
Fresh Lab? [Yes]
{

    STEP 1        Initialize session  -->  $./lab.sh -r 210010_211101 -n 5 -a init
    STEP 2        Download the lab  -->  $./lab.sh -r 210010_211101 -n 5 -a get

   STEP {3 to L}   ----- WORK ON THE EXERCISE ------

   Completed? [Yes]

        STEP L     Evaluate the exercise --> $./lab.sh -r 210010_211101 -n 5 -a evaluate
        STEP L+1   Prepare submission -->  $./lab.sh -r 210010_211101 -n 5 -a prepare
        STEP L+2   Submit -->  $./lab.sh -r 210010_211101 -n 5 -a submit
        STEP L+3   Signoff --> $./lab.sh -r 210010_211101 -n 5 -a signoff

   Completed? [No]

        STEP L+1   Prepare to save -->  $./lab.sh -r 210010_211101 -n 5 -a prepare-save
        STEP L+2   Save your work  -->  $./lab.sh -r 210010_211101 -n 5 -a save
        STEP L+3   Detach --> $./lab.sh -r 210010_211101 -n 5 -a detach
}
Saved Lab? [Yes]
{
   STEP 1        Reload the lab   -->  $./lab.sh -r 210010_211101 -n 5 -a reload

   STEP {2 to L}   ----- WORK ON THE EXERCISE ------

   Completed? [Yes]

        STEP L    Evaluate the exercise --> $./lab.sh -r 210010_211101 -n 5 -a evaluate
        STEP L+1  Prepare submission -->  $./lab.sh -r 210010_211101 -n 5 -a prepare
        STEP L+2  Submit -->  $./lab.sh -r 210010_211101 -n 5 -a submit
        STEP L+3  Signoff --> $./lab.sh -r 210010_211101 -n 5 -a signoff

   Completed? [No]

        STEP L+1  Prepare to save -->  $./lab.sh -r 210010_211101 -n 5 -a prepare-save
        STEP L+2  Save your work  -->  $./lab.sh -r 210010_211101 -n 5 -a save
        STEP L+3  Detach --> $./lab.sh -r 210010_211101 -n 5 -a detach

}

***** IMPORTANT ****
- Check the evaluation output
- Make sure you submit before signing off
- Make sure you save the lab before detaching (if you want to continue next)
- Make sure you signoff (STEP L+3) or else you will not get marks and will not get
  the submissions emailed
- Make sure you logout from the system (not just the docker container)
```

```
***** CAUTION  *****
DO NOT DELETE lab.sh core or gem5
```

Testing

Manual Testing

You should use this approach to perform manual checks (with additional debug statements) while you are developing the required functionality. The steps for performing this kind of testing are as follows,

- Step 1: Ensure you are logged into the "Labex" environment (using SSH) using a terminal.
- Step 2: Change directory to lab-*/labex/Q_N using the `cd` command. Make modifications to the source code as per the specifications.
- Step 3: Compile the C code using `gcc`. Run the program with the test cases specified in the `testcases` directory.

Automated evaluation

Note: Remove additional debug prints before performing automated evaluation

This step can be performed either completing each part of the exercise or before making a final submission. To test a particular part (say Q1), change your current working directory to `lab-3/labex/Q1`. Execute `./run_tests.sh`. After completion of the script, the output produced will be stored in the `output` directory. To evaluate the complete exercise, use the usual procedure of executing an "evaluate" action using the `lab.sh` script.

For this lab, you can print debugging/status messages to `stderr`. We use Unix pipes for evaluation, which capture only `stdout`. For each question, the template has examples. *Do not clutter* `stdout`*'s contents.*

Exercise Overview

In this lab, we will explore the process address space and related system call API. The help material can be accessed using the following link: `http://172.27.21.215`

Q1: New memory management API's in town... [20 Marks]

In this part of the lab, you are given a C program `Q1/memMagic.c`. `Q1/memMagic.c` program currently uses malloc(), free() and realloc() library functions to dynamically allocate memory chunk, to free dynamically allocated memory chunk and to modify the size of dynamically allocated memory chunk. Your goal is to replace all malloc(), free() and realloc() library function calls with mmap(), munmap() and mremap() system calls. On successful replacement of above mentioned function calls, `Q1/memMagic.c` will print **'Program surgery successful!'** message.

Manual Build and Execution

```
// Current working directory should be lab-4/labex/Q1
$ gcc memMagic.c -o memMagic
$./memMagic
Program surgery successful!
$
```

Example

```
//allocate 4096 bytes with read and write permission
ptr = mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_ANONYMOUS|MAP_PRIVATE, 0, 0);

//change size of the allocated memory (pointed by
//the pointer 'ptr') from '4096' bytes to '8192' bytes
ptr = mremap(ptr, 4096, 8192, MREMAP_MAYMOVE);

//free allocated '8192' bytes memory
munmap(ptr, 8192);
```

Error handling

In case of any error, print **"Error"** as output.

System calls and library functions allowed

You **must only use** the below mentioned APIs in this question.

- `mmap`
- `munmap`
- `mremap`

Testing

Run `Q1/run_tests.sh` script to check whether your implementation passes the evaluation criteria or not.

Q2: What is my span! [40 Marks]

In this part, for a given virtual address *address*, you are required to find the length of the *continuous virtual address space range* that contains *address*. You are required to provide the implementation for the function `how_large` in `Q2/memspan.c`. If an address does not belong to any valid region, you should return 0.

Example

Consider the following state of address space for a process where '...' represent unallocated regions.

```
...[4096 - 16384)...[122880 - 245760)...

    how_large(10000) returns 12288
    how_large(150000) returns 122880
    how_large(45002001) returns 0
```

Notes

- You may use `mmap` with address hint (first argument to mmap) to check if the hint address is mapped or not. Refer to the man page of the `mmap` system call (*man mmap*).
- When you return from the function, you should make sure that the state of the address space is same as that when your function was called i.e., you should not have any additional allocations.

System calls and library functions allowed

You **must only use** the below mentioned system call in this question.

- `mmap`
- `munmap`

Manual build and execution

```
// Current working directory should be lab-4/labex/Q2
$ make
$ ./memspan <tc>
// tc: the testcase number
```

Automated Testing

Run the `Q2/run_tests.sh` script to check whether your implementation passes the test cases or not. A correctly implemented program would generate the following output:

Test case 1 passed
Test case 2 passed
...
...
Test case 10 passed

Q3: Relocate Binary, for fun! [40 Marks]

In this part of the lab, you are given a C program `Q3/relocate.c`. The objective of this exercise is to relocate the binary to a dynamically allocated memory area. In the template source file `Q3/relocate.c`, you are required to provide implementation for the designated parts (demarcated with `TODO` tags) of the `main` function and `relocate_textseg` function.

In the `main` function, you need to fill the `reloc_code_addr` with a properly mapped memory region that contains the relocated main function (and other functions) by calling the `relocate_textseg` function. You are also required to set the corresponding address of main (`reloc_main`) in the relocated executable area. Your task is to invoke the `relocate_textseg` function with proper arguments and do the required processing such that both values are correct.

The `relocate_textseg` function takes two unsigned long (64-bit) arguments i.e., `start` and `end` and returns a pointer to a memory region, containing the relocated content. You may need to use additional logic (in the `main` function) to set the relocated address of the main function.

Solution Strategy and Hints

Please note the following while thinking about a strategy.

- The relocation of only text segment *will not* suffice in many cases because of dependence of the instructions on the data segments and other constants. Therefore, you may map every valid address between *start of the code segment page* and *end of the data segment.*
- The `start_code` variable maintains the start of the code segment adjusting the the entry point address i.e., `_start`, to a page boundary.
- The end of the data segment maintained in the `end_data` variable.
- To make your job simple, you may map the relocated area with maximum permissions (`PROT_READ PROT_WRITE` and `PROT_EXEC`) and in a page aligned manner.
- You may need to check the validity of the source addresses before accessing them. While there are no system calls to check the validity of a memory page, it can be done with intelligent use of `mmap` and `munmap` calls.

Manual Build and Execution

```
// Current working directory should be lab-4/labex/Q3
$ gcc relocate.c -o relocate
$ ./relocate
##### First Call to main #####
Relocated Code Segment:nnnnn...nnnnn:0xhhhh...hhhh
Relocated Main function:nnnnn...nnnnn:0xhhhh...hhhh
##### Repeat Call to main #####
Now RIP:nnnnn...nnnnn:0xhhhh...hhhh
Folded number: nnnnn
$
```

Error handling

In case of any error, print **"Error"** as output.

System calls and library functions allowed

You **must only use** the below mentioned APIs in this question.

- `mmap`
- `munmap`

Testing

Run `Q3/run_tests.sh` script to check whether your implementation passes the evaluation criteria or not.

Test case 1 passed
Test case 2 passed