Operating System Lab
Lab #11
Memory Management

Goal:

This lab requires you to fully understand, Linux memory monitoring command, the memory allocation methods, and then implement the method in C programming language.

Preparation:

Prepare the working environment

- Check out the repo from Github classroom
  - Use `git clone {repo URL}`

- Rebuild Docker image and run the container
  - In a Windows Powershell or MacOS/Liunx terminal, change the current folder to the "lab" folder (the one you check out).
    `cd {lab_folder}`
  - For **Mac** or **Linux**
    - You can execute the shell script "clean_start.sh" if you want to remove old image and create and run a new one.
      `./clean_start.sh`
    - Otherwise, you can just run the following "docker run" command to run a container.
  - For **Windows**
    - You need the following docker commands to run the container from the existing image, *ubuntu_os*.

```
docker run -itd -p8888:8888 --rm --name ubuntu-os-con -v"$PWD/share":/home/app ubuntu_os
```

    - If you do not have the image *ubuntu_os*, you need to rebuild the image

      `docker build -t ubuntu_os .`

  - To enter the "Bash Shell" of the Ubuntu container for compiling and testing codes

```
docker exec -it ubuntu-os-con /bin/bash
```

- Use VSCode or any text editors for development
  - In the host OS (Windows or MacOS), use terminal to change to the host's lab folder "{check out repo folder}/share" or VSCode to open that host's lab folder. Files you created or modified in the "{check out repo folder}/share" will be shown in docker container's "/home/app" folder.
  - Copy required files from "{check out repo folder}/share" to "{check out repo folder}/answer" and check in for submission.

Part I: Linux Commands for Tracing Memory Usage

There are variety of tools in a Linux system to trace memory usage of the system. This helps to determine the program that consumes all CPU resources or the program that is responsible for slowing down the activities of the CPU. In the following section, we introduce those tools. Detailed usage methods or examples require your research online.

- **top**

We learn `top` before. `top` command displays all the currently running process in the system. This command displays the list of processes and threads currently being handled by the kernel. top command can also be used to monitor the total amount of memory usage.

- **free**

It displays the amount of memory which is currently available and used by the system (both physical and swapped). `free` command gathers this data by parsing `/proc/meminfo`. By default, the amount of memory is display in kilobytes.

- **vmstat**

`vmstat` command is used to display virtual memory statistics of the system. This command reports data about the memory, paging, disk and CPU activities, etc. The first use of this command returns the data averages since the last reboot.

To-do1:

1. Use the script method to record the following commands. Create a file, called "*vmstat_output*" for saving the following practice. Run vmstat every 2 seconds for 8 times, and report the swap memory usage, and free memory. Answer the question - "*what is the meaning of si, so, bi, bo columns?*" under "*to-do1-1*" in the "answer.md"

2. Use `free` command to report amount of free space and swap size. Write a shell script program called "*mem_monitor.sh*" to show the memory usage in a human readable format every 2 seconds (do not use a loop).

   Example Execution

```
./mem_monitor.sh
Showing memory usage every 2 seconds (Ctrl+C to stop)
               total        used        free      shared  buff/cache   available
Mem:           7.8Gi       973Mi       5.0Gi       3.0Mi       1.8Gi       6.5Gi
Swap:          1.9Gi          0B       1.9Gi

               total        used        free      shared  buff/cache   available
Mem:           7.8Gi       973Mi       5.0Gi       3.0Mi       1.8Gi       6.5Gi
Swap:          1.9Gi          0B       1.9Gi

               total        used        free      shared  buff/cache   available
Mem:           7.8Gi       973Mi       5.0Gi       3.0Mi       1.8Gi       6.5Gi
Swap:          1.9Gi          0B       1.9Gi
```

Part II: Memory Allocation

Two memory allocations, continuous memory allocation and paging were discussed in the class. The following to-dos are exercises for understanding better about both approaches.

- **Continuous Memory Allocation**

Each process requires to be saved in memory to be executed. Therefore, the memory will be allocated for process to be run. There are three algorithms for allocating the memory for processes. They are first-fit, best-fit, and worse-fit.

In Figure 1, the memory is divided into multiple partitions. You can assume the red partition are all fully occupied, that is, no internal fragmentation. Table 1 is the process arrived for execution.

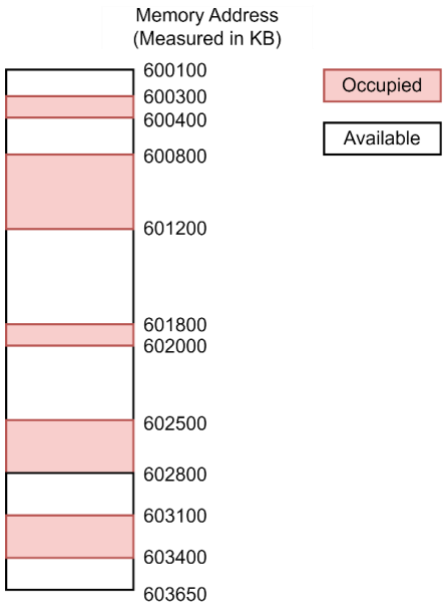

Figure 1 A Memory Map Presentation

Process	AT	Size
P1	1	357 KB
P2	2	210 KB
P3	3	468 KB
P4	4	491 KB

Table 1 Process List

To-do2:

See Figure 1 and Table 1. If a process is allocated to a partition, the partition is considered to be occupied. Even it is not fully occupied, the entire partition cannot be assigned to any other process. Also, consider the process will not be terminated.

1. Please use the first-fit algorithm to fill up the Table 2 below. Put your answer under "*to-do2-1*" in the "*answer.md*".
2. Similar to 1, please use the best-fit algorithm to fill up the Table 2 below. Put your answer under "*to-do2-2*" in the "*answer.md*".
3. Discuss any issue you see in 1 and 2. Put your answer under "*to-do2-3*" in the "*answer.md*".

Process	Starting Address	Ending Address
P1		
P2		
P3		
P4		

Table 2 Process memory allocation result

- **Paging**

During the paging, the following outlines the steps taken by the MMU to translate a logical address generated by the CPU to a physical address:

1. Extract the page number *p* and use it as an index into the page table.
2. Extract the corresponding frame number *f* from the page table.
3. Replace the page number *p* in the logical address with the frame number *f*.

As the offset *d* does not change, it is not replaced, and the frame number and offset now comprise the physical address.

The size of a page is a power of 2, typically varying between 4 KB and 1 GB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy. If the size of the logical address space is 2^m, and a page size is 2^n bytes, then the high-order m−n bits of a logical address designate the page number, and the n low-order bits designate the page offset.

Figure 2 A Memory Map Presentation

The page table maintain the mapping between page number and frame number. The CPU generated logical address will be used to look up the actually address in the physical memory.

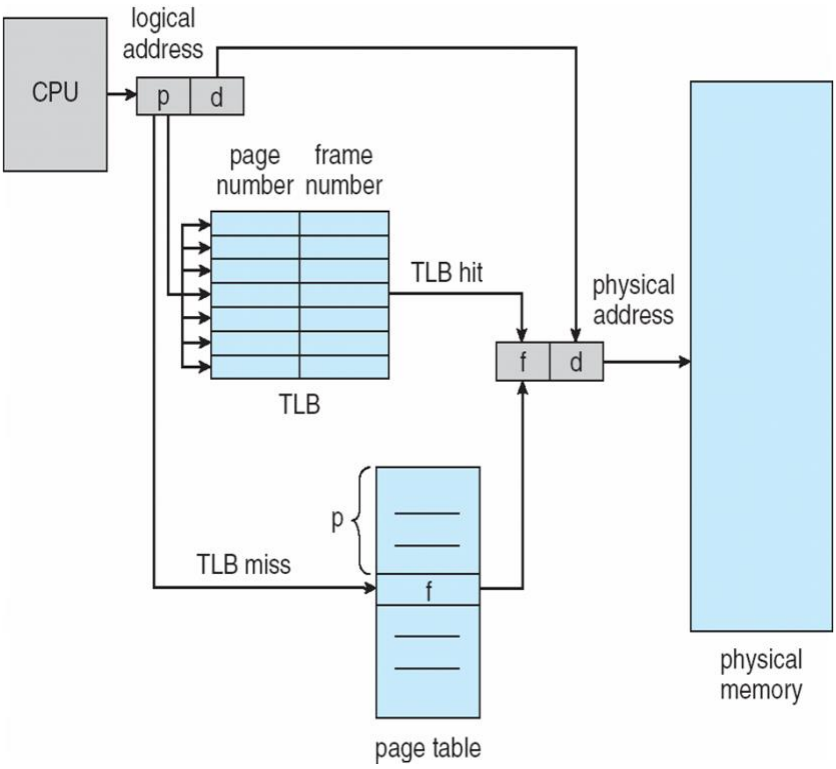

Figure 3 Page allocation mechanism with TLB

To-do3:

Given a 64bit machine with 16 terabytes RAM using 4kb page size. Please answer the following questions and please show how you get the result.

1. How many memory frames does the machine have? Put your answer under "*to-do3-1*" in the "*answer.md*".
2. How many bits are needed for displacement into a page? Put your answer under "*to-do3-2*" in the "*answer.md*".
3. Assume it takes 36 bits to index into the page table, how large can our virtual memory be in terms of bytes? Put your answer under "*to-do3-3*" in the "*answer.md*".

To-do4:

1. Read the "*contiguous_skeleton.c*" and implement the missing methods "bestFit" and "firstFit" and then name it "*contiguous.c*". Run your program to verify your answer in To-do2.

   Example input: (the italic font presents user inputs)

```
Enter number of partitions: 4
Enter partition sizes:
100
200
300
400
Enter starting memory address for the FIRST partition: 600300
Enter pre-allocated memory (0=available, >0=occupied) for each partition:
0
150
```

```
0
400
Enter number of processes: 2
Enter process sizes:
180
200
```

Example output from above input:

```
--- FIRST FIT ---

=== SEGMENTS INFO ===
Partition 0: Start=600300  Size=100  Occupied=NO
Partition 1: Start=600400  Size=200  Occupied=YES
Partition 2: Start=600600  Size=300  Occupied=YES
Partition 3: Start=600900  Size=400  Occupied=YES

Allocated Processes:
Process 0 (size=180) -> Partition 2 | Start=600600 End=600780

Unallocated Processes:
Process 1 (size=200) could NOT be allocated

--- BEST FIT ---

=== SEGMENTS INFO ===
Partition 0: Start=600300  Size=100  Occupied=NO
Partition 1: Start=600400  Size=200  Occupied=YES
Partition 2: Start=600600  Size=300  Occupied=YES
Partition 3: Start=600900  Size=400  Occupied=YES

Allocated Processes:
Process 0 (size=180) -> Partition 2 | Start=600600 End=600780

Unallocated Processes:
Process 1 (size=200) could NOT be allocated

--- WORST FIT ---

=== SEGMENTS INFO ===
Partition 0: Start=600300  Size=100  Occupied=NO
Partition 1: Start=600400  Size=200  Occupied=YES
Partition 2: Start=600600  Size=300  Occupied=YES
Partition 3: Start=600900  Size=400  Occupied=YES

Allocated Processes:
Process 0 (size=180) -> Partition 2 | Start=600600 End=600780

Unallocated Processes:
Process 1 (size=200) could NOT be allocated
```