CSOPESY Lecture – An Overview of CSOPESY OS Emulator

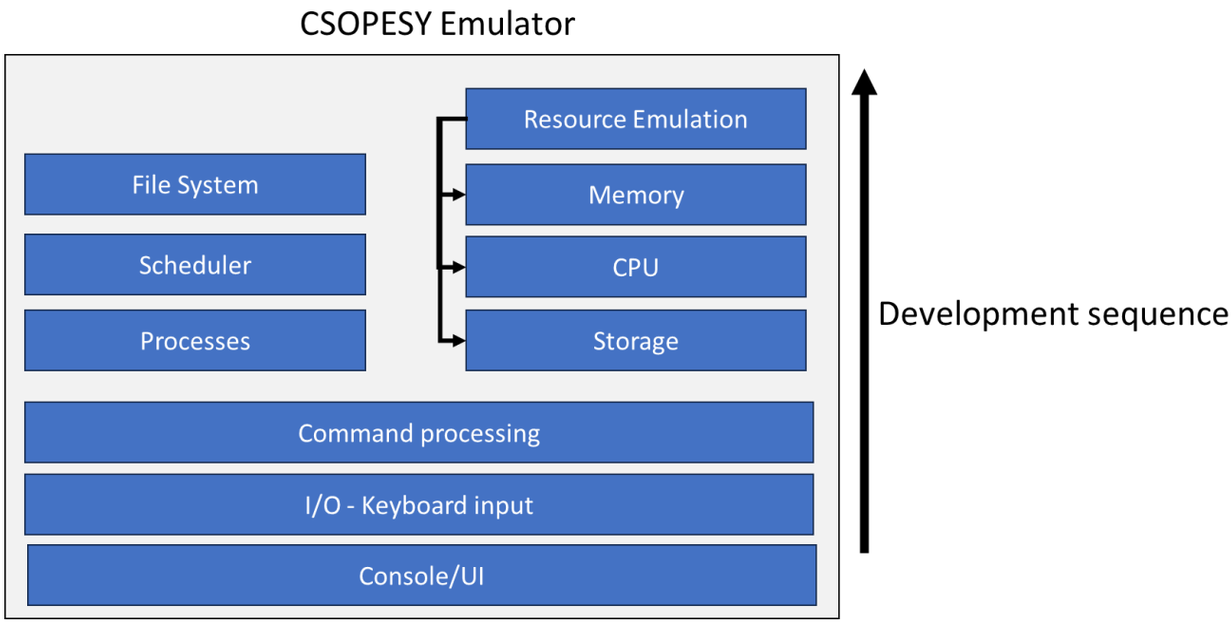
Instructor: Neil Patrick Del Gallego

Background

- What will you work on for most of the term – an OS emulator.
- Features of the CSOPESY OS emulator
- An overview of its proposed design of systems to get you started.
- Preview of assessments for your problem sets.

The CSOPESY emulator

- An emulator of different OS concepts – we will code the following important features: I/O interface, keyboard input, drawing/refreshing the screen, file directory system, process scheduling, and memory management.



A typical sequence of how an OS is loaded and run

```cpp
#include <iostream>

// Function prototypes
void bootstrap();
void initializeKernel();
void startSystemServices();
void enterMainLoop();
void shutdownAndCleanup();

// Define the main function
int main() {
    // Step 1: Bootstrapping
    bootstrap();

    // Step 2: Kernel Initialization
    initializeKernel();

    // Step 3: Start System Services
    startSystemServices();

    // Example: Print a message indicating successful boot-up
    std::cout << "Operating System Booted Successfully!\n";

    // Step 4: Enter Main Loop
    enterMainLoop();

    // Step 5: Shutdown and Cleanup
    shutdownAndCleanup();

    // Note: The OS typically never reaches this point
    return 0;
}
```

1. Bootstrapping:
    - Perform low-level initialization (hardware setup, memory initialization, etc.)
    - Load the kernel into memory and start executing it
    - Example: Perform hardware initialization

2. Kernel initialization
    - Initialize data structures (process table, file system, etc.)
    - Set up interrupt handlers and device drivers
    - Initialize memory management and scheduling algorithms
    - Example: Initialize process table and memory manager (e.g. memory allocation algorithm, demand paging)
3. Start system services
    - Start essential system services (file system, networking, etc.)
    - Launch system daemons and background processes
    - Example: Start file system service and network service
    - Observation: These are processes that are persistent in the OS – always alive/periodically invoking functions
4. Enter main loop
    - Continuously handle interrupts and system calls
    - Dispatch user processes and manage their execution
    - Handle user input and manage I/O operations
    - Example: Enter an infinite loop to handle system events

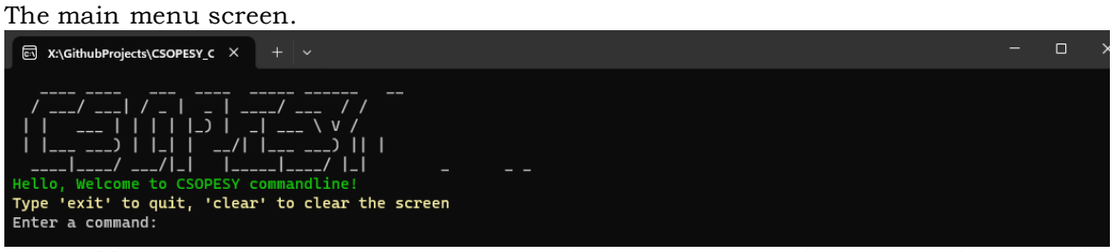
```
737  // Step 4: Enter Main Loop
738 void enterMainLoop() {
739     // Example: Enter an infinite loop to simulate OS operation
740     while (<user hasn't initiated shutdown>) {
741         // Continuously handle interrupts and system calls
742         // Dispatch user processes and manage their execution
743         // Handle user input and manage I/O operations
744         // Example: Enter an infinite loop to handle system events
745     }
746 }
```

5. Shutdown and cleanup
    - Gracefully terminate running processes and services
    - Clean up allocated resources and release memory
    - Halt or reboot the system
    - Example: Clean up resources and prepare for shutdown

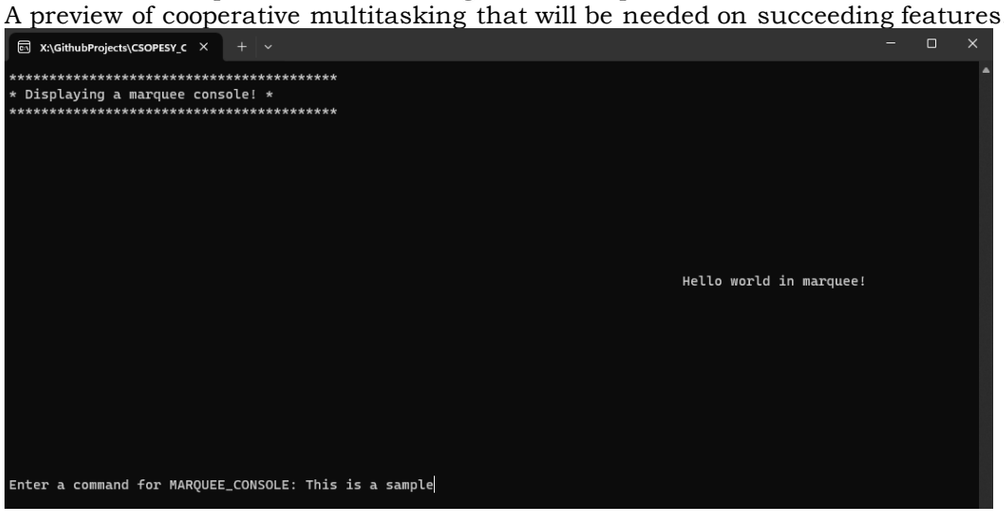
A walkthrough of its features

A command-line interface type of application (CLI) → akin to Windows Powershell, Bash, DOS, and other related command-line applications.

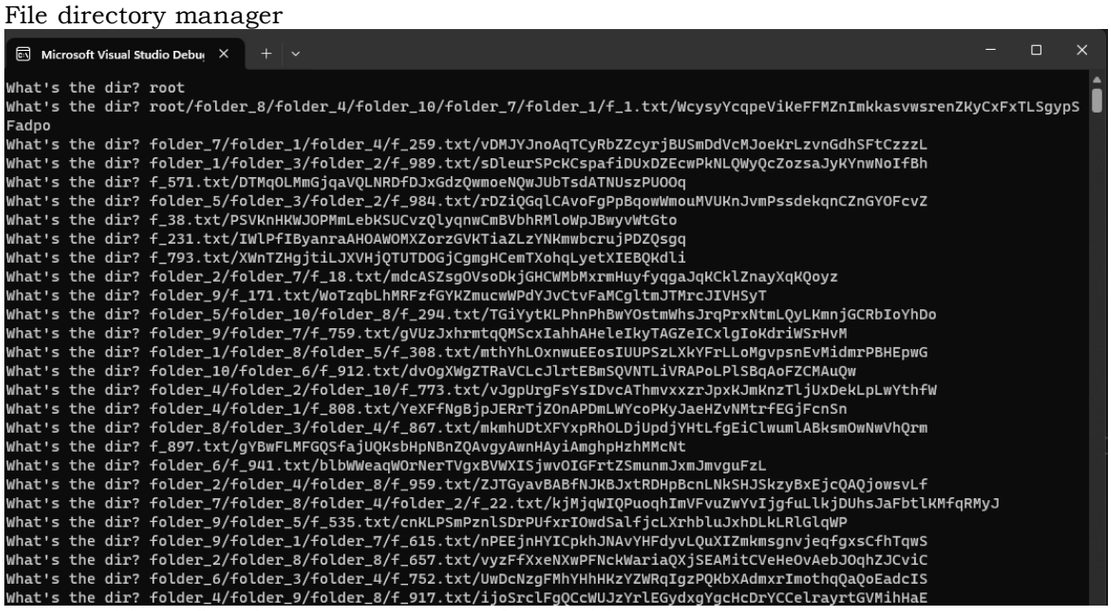
- The main menu screen.



    - Main menu interface accepts commands that transition the user to other console windows. E.g., file directory viewer, marquee console, process scheduler, memory emulator.
    - Drawing of new console layout is done in the same window. No pop-up.
- Marquee console
    - This is simply a demonstration of creating an immediate-mode application → consistently listens to user inputs while refreshing the screen per frame.
    - A preview of cooperative multitasking that will be needed on succeeding features.



- File directory manager

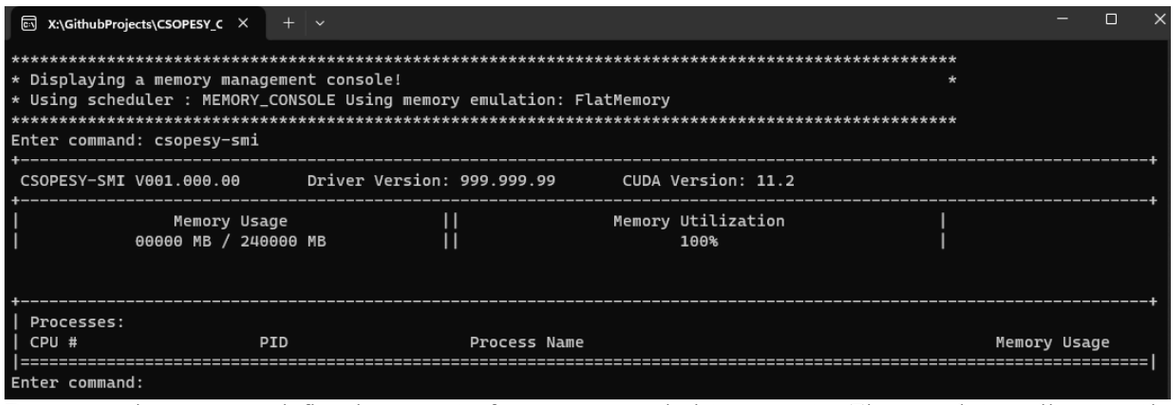


-
  - Akin to file directory managers of most OS.
  - Supports standard file directory commands, like **mkdir**, **cd**, **createdir**, **search**, **dirlist**, etc.
  - All files are represented as a **string of bytes**, all files are bundled together into a single **.csopesy file**.
  - A file directory system reads the .csopesy file, parsing and interpreting the multi-level directory structure.
  - Certain files may be required by certain processes → depending on configuration.
- Scheduling

```
 _____ ____   ___  ____  _____ ______   __
/ ___/ ___| / _ \|  _ \| ____/ ___\ \ / /
| |   \___ \| | | | |_) |  _| \___ \\ V /
| |___ ___) | |_| |  __/| |___ ___) || |
 \____|____/ \___/|_|   |_____|____/ |_|
-----------------------------------------
Welcome to CSOPESY Emulator!

Developers:
Del Gallego, Neil Patrick

Last updated: 01-18-2024

root:\> screen -ls
CPU utilization: 100%
Cores used: 4
Cores available: 0

--------------------------------------
Running processes:
process05    (01/18/2024 09:15:22AM)    Core: 0     1235 / 5876
process06    (01/18/2024 09:17:22AM)    Core: 1     3 / 5876
process07    (01/18/2024 09:17:45AM)    Core: 2     9 / 1000
process08    (01/18/2024 09:18:58AM)    Core: 3     12 / 80

Finished processes:
process01    (01/18/2024 09:00:21AM)    Finished     5876 / 5876
process02    (01/18/2024 09:00:22AM)    Finished     5876 / 5876
process03    (01/18/2024 09:00:42AM)    Finished     1000 / 1000
process04    (01/18/2024 09:00:53AM)    Finished     80 / 80
--------------------------------------

root:\> report-util
root:\> Report generated at C:/csopesy-log.txt!
```

-
  - A scheduling simulation → simulates various CPU scheduling algorithms, given N processes and X CPU cores. Note that CPU cores are simulated/declared during compile-time.
  - An arbitrary number of processes are scheduled. Each process contains Z commands → commands could either be a simple print or an I/O operation such as searching for a file or a request for a driver/hardware.
  - Can display a summary of the order of processes finished/still executing.

- Memory management
  - Simulation that combines memory management + scheduling → using existing CPU schedulers implemented, processes only execute if sufficient memory is available.
  - Has a command that displays memory utilization and running processes. E.g. nvidia-smi



- Processes have a pre-defined amount of memory needed to execute. The emulator allocates the needed memory, if available.
- Allows user input-driven creation of processes on the fly → while having a predefined list of running processes, the user can add new ones. All processes must finished eventually → CPU and memory utilization should show 0%.
- Allows real-time viewing of CPU and memory utilization, while still typing commands.

Sample assessments for each feature

- This is to help set expectations on how your machine project would be graded.
- Typical approach: Online quiz system → submit .mp4 or .png files as proof of output, using your running OS emulator.
- Online quiz system is in time-pressure format (e.g. 2 hours or 1 hour) → this is to avoid drastic changes to your code, hacking, band-aid fixing, and frequent recompiling just to meet the expected output → promotes pro-active QA/testing of your app.
- The test cases could be given earlier or when the quiz is started.

Marquee console and command-line interface

Test	Expected output
Typing "the quick brown fox jumped over the lazy dog" with pre-defined animation speed for the marquee	The marquee should behave/move in accordance with the set animation speed while keeping the keyboard polling active as much as possible.
Ability to recognize placeholder commands for future use: scheduling, filesystem, memory	Commands are recognized properly, while the marquee is still animating.

Needed programming concepts: I/O and display interface, parsing commands, threads.

Scheduling

- Each process is attached to a screen and the overall application behaves like a process multiplexer.
- Please refer to a general Linux/Windows powershell/Windows command line. This serves as a strong reference for the design of your command-line interface.
- For the process multiplexer, refer to the Linux "screen" command on its behavior: https://www.geeksforgeeks.org/screen-command-in-linux-with-examples/

Test	Expected output
The teacher will provide the screen commands	All screen commands and processes associated with it must execute/finished execution using the intended scheduling algorithm.

Needed programming concepts: reading/writing files (for parsing the test case input), CPU scheduling, threads.

Memory management

- There must be a mechanism to visualize and debug memory. The user can use either "process-smi," which provides a high-level overview of available/used memory, or "vmstat," which provides fine-grained memory details.

- The "process-smi" is similar to the nvidia-smi command that prints a summarized view of the memory allocation and utilization of the processor (CPU for your program / GPU for nvidia-smi).

Test	Expected output
Given a batch of processes, provide periodical memory snapshots at fixed intervals	The memory snapshots should report the overall memory usage, free memory, fragmented memory, etc.

Simulate a low memory environment	The memory report should show high thrashing activity and high page swaps.

Needed programming concepts: I/O and display interface, parsing commands, threads, CPU scheduling, cooperative multitasking, deadlock handling, memory management.

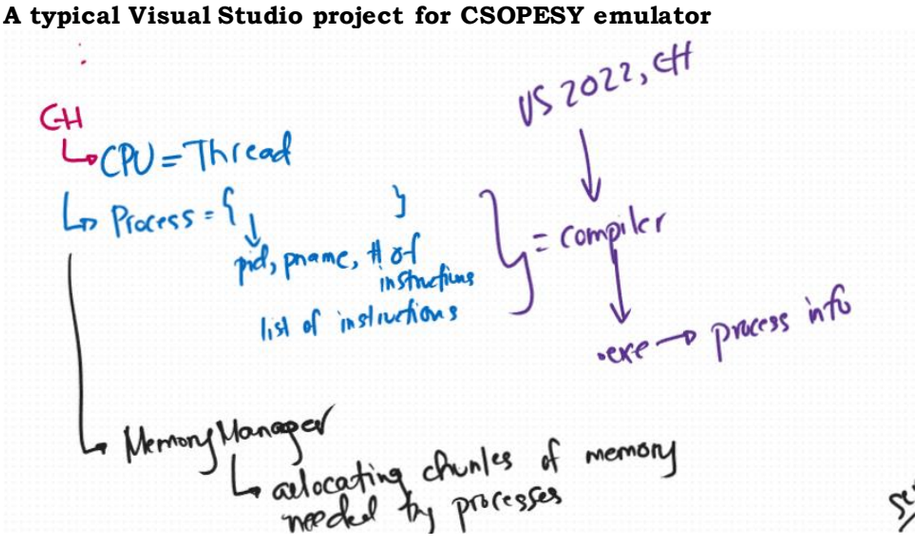
File directory manager

- The teacher sends a sample .csopesy file → the student performs the following tests

Test	**Expected output**
Searching for a specific file	Outputs "file not found" or "file found"
Searching for a keyword	Outputs lists of files that met the search keyword
Creation of directories and files	Able to output the newly created directory path, as well as the file path.

Needed programming concepts: file systems, reading/writing files, parsing commands.

A typical Visual Studio project for CSOPESY emulator



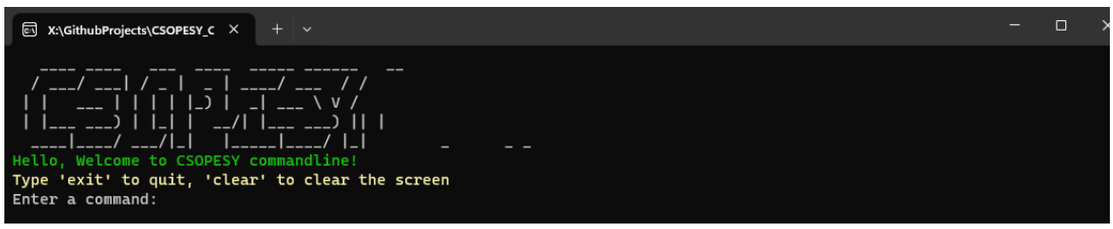
- A CPU could be represented by an inherited std::thread class.
- A process could be represented as another class containing important information such as: process id, name, number of instructions, and types of instructions.
- The memory allocator could be created as another class where various memory allocation algorithms will be implemented.

Things to discuss

Discuss hands-on activity: CSOPESY-Emulator tour.

Activities

#1: Setting up your own CSOPESY project



- Create your first-ever **main menu** console. Use only the C++ built-in libraries.
- Provide your ASCII text header "CSOPESY" or a name for your command line emulator.
- Set up template code that accepts the following commands: **marquee, screen, process-smi, nvidia-smi, clear, exit.** Simply print "X command recognized. Doing something.", where X is the command recognized.
- For the clear command, once recognized, the screen clears, reprinting the header texts.
- For the exit command, the application/CLI immediately closes.
- Be prepared to show your design implementation when called.

#2 Written: Mapping specific OS functions to the general steps
Identify the step in the bootstrapping process for each of the following 35 functions of a commercial OS:

1. Initializing the hardware abstraction layer (HAL)
2. Setting up the Global Descriptor Table (GDT)
3. Enabling the Floating Point Unit (FPU)
4. Loading the initial RAM disk (initrd)
5. Configuring the Interrupt Descriptor Table (IDT)
6. Initializing the Advanced Configuration and Power Interface (ACPI)
7. Enabling virtual memory and setting up page tables
8. Initializing the Programmable Interval Timer (PIT)
9. Setting up the boot loader environment
10. Starting the initial process scheduler
11. Configuring the Memory Management Unit (MMU)
12. Loading kernel modules and drivers
13. Initializing the System Management Mode (SMM)
14. Enabling symmetric multiprocessing (SMP) support
15. Initializing the Real-Time Clock (RTC)
16. Configuring the Unified Extensible Firmware Interface (UEFI)
17. Setting up kernel command line parameters
18. Configuring input/output ports and devices
19. Starting the initial RAM disk filesystem
20. Initializing the system logger
21. Setting up the Real Mode Interrupt Vector Table (IVT)
22. Setting up the initial file system
23. Starting the graphical user interface (GUI)
24. Configuring network interfaces and protocols
25. Initializing user authentication mechanisms
26. Loading and initializing system libraries
27. Setting up inter-process communication (IPC) mechanisms
28. Configuring power management features
29. Initializing system time and date settings
30. Starting system monitoring and performance analysis tool
31. Saving system configuration settings to persistent storage
32. Flushing system buffers and caches
33. Terminating user processes and services gracefully
34. Releasing allocated system resources and memory
35. Closing open file handles and network connections