Gebze Technical University
Department Of Computer Engineering
CSE 312 / CSE 504

Operating Systems

Semester Project
Due Date: June 1 2025

You will simulate a simple hypothetical CPU with a small instruction set and you will write a small operating system using the instructions of this CPU.

You will run your OS and a few threads on this CPU in parallel and show the memory contents during and after the program execution. After this project, you will have the following skills

- CPU and ISA Design: Understanding custom instruction sets, memory-mapped registers, and instruction execution logic.
- Operating System Implementation: Writing a simple cooperative OS in assembly-like language, including thread scheduling and system calls.
- Systems Simulation in C/C++/Python: Building a CPU simulator with debug modes, memory tracing, and thread execution control.
- Thread Management: Creating and managing thread tables, states, context switching, and cooperative multitasking via SYSCALL YIELD.
- Algorithmic Problem Solving: Implementing low-level sorting, searching, and custom logic using the GTU-C312 instruction set.

The CPU

Your CPU (named GTU-C312) has a very small and non-typical instruction set.

Intruction	Explanation
`SET B A`	Direct Set : Set the Ath memory location with number B. Example: SET -20, 100 writes the value of -20 to memory location 100.
`CPY A1 A2`	Direct Copy: Copy the content of memory location A1 to memory A2. Example: CPY 100, 120 copies the memory value of address 100 to the memory address 120
`CPYI A1 A2`	Indirect Copy: Copy the memory address indexed by A1 to memory address A2. Example: CPYI 100, 120: if memory address 100 contains 200, then this instruction copies the contents of memory address 200 to memory location 120.
`CPYI2 A1 A2 (optional)`	Indirect Copy 2: Copy the memory address indexed by A1 to memory address indexed by A2. Example: CPYI2 100, 120: if memory address 100 contains 200, and 120 contains 300 then this instruction copies the contents of memory address 200 to memory location 300.
`ADD A B`	Add number B to memory location A

ADDI A1 A2	Indirect Add: Add the contents of memory address A2 to address A1.
SUBI A1 A2	Indirect Subtraction: Subtract the contents of memory address A2 from address A1, put the result in A2
JIF A C	Set the CPU program counter with C if memory location A content is less than or equal to 0
PUSH A	Push memory A onto the stack. Stack grows downwards.
POP A	Pop value from stack into memory A.
CALL C	Call subroutine at instruction C, push return address.
RET	Return from subroutine by popping return address.
HLT	Halts the CPU
USER A	Switch to user mode and jump to address contained at location Ard
SYSCALL PRN A	Calls the operting system service. This system call prints the contents of memory address A to the console followed by a new line character. This system call will block the calling thread for 100 instruction executions.
SYSCALL HLT **SYSCALL YIELD**	Calls the operting system service. Shuts down the thread Calls the operting system service. Yields the CPU so OS can schedule other threads.

The numbers in the above instruction sets are signed long intergers, which means that ADD 300 -1 can be used as an instruction to decrement the memory location 300 value by 1.
The parameters for the instructions can be memory locations (A, A1, A2), unsigned long numbers (B), or instruction numbers (C).
GTU-C312 does not have any registers, instead it uses special memory locations as registers shown below

Memory Location	Register
0	Program Counter
1	Stack pointer
2	System call result
3	Number of instructions executed so far
3-20	Reserved for other uses

Your whole OS + program file formats will include two sections: one section for the data, one section for the program instructions.

One example program without the OS is given below, it adds number from 1 to 10 and puts the result in address 51.

```
#Program sample
Begin Data Section
0 0                 #program counter
1 0                 # stack pointer
2 0
3 0
4 0
```

```
5 0
...
255 0
End Data Section
Begin Instruction Section
0 SET 10 50       # i = 10
1 SET 0 51        # sum = 0
2 ADDI 50 51      # sum = sum + i
3 ADD 50 -1       # i = i - 1
4 JIF 50 7        # Go to 7 if i <= 0
5 SYSCALL PRN 51  # print the sum so far, since this program does not have OS
                  # This will be ignored
6 SET 2 0         # Go to 2 - remember address 0 is the program counter
7 HLT             # end of program, the result is in memory address 51 (sum)
End Instruction Section
```

The data and the instruction section can be as large as needed. You can have comments after the # charater in any line.

When your simulated computer starts running, your BIOS will first read a file. One example is above. BIOS will read both data segment and instruction segment, then it will start executing starting from the PC which is stored in address zero. If the file is only a single program like above, it will be executed. If the file contains OS and other threads, then everything will be controlled by the OS. In other words, the program you are reading is your OS and your threads.

Then it will run a loop similar to

```
while (!myCpu.isHalted())
      myCpu.execute();
```

The CPU function execute will take just one instruction at the Program Counter (PC), execute it, then increment the PC (if no jump), increment register number 3 (number of instructions executed). This loop ends when the CPU is halted. Your CPU is in the kernel mode when it starts, before you start running threads, you will execute USER instruction. Of course, when a system call is made, the CPU will be automatically in the kernel mode. In KERNEL mode, your CPU can access anywhere in the memory, in USER mode, it can access only the address 1000 and above. If in the USER mode your instruction accesses in the first 1000 addresses, then that thread is shutdown.

The OS

Your OS will be written in GTU-C312 code. When it starts first it will setup a thread table for 10 threads. The threads will be functions at specific locations in memory (the first thread will be at address 1000, the second thread will be at address 2000, and the last thread will be at 3000, etc. If you like to keep a thread inactive just run SYSCALL HLT as the first instruction of the thread. The thread table will hold at least the following properties for each thread

- The thread ID
- Starting time of the thread (how many instructions were executed since the system bootup by CPU)
- How many instruction executions the thread has used so far
- The state of the thread (ready, blocked, running)
- The registers of the threads. The PC and the stack pointers should be set accordingly at the beginning.
- Any other data structure can be added to the process table as needed.

Note that OS is a thread too, so include a table entry for the OS too.

After setting up the thread table, it will start scheduling the threads. Each thread is supposed to run SYSCALL YIELD or some other system call so that our OS can do scheduling, which is a simple round robin scheduler. In other words, our OS is not a preemptive OS.

General Placement of the OS and the threads			
Address	**Data Segment**	**Instruction Segment**	**Explanation**
0-20	Registers	OS area	Of course, the registers are accessible to all
21-999	OS data only	OS instruction only	USER mode can't access this area
1000-1999	Thread #1 data	Thread #1 instructions	
2000-2999	Thread #2 data	Thread #2 instructions	
…	…	…	Space for threads 3 to 9
10000-10999	Thread #10 data	Thread #10 instructions	

The Project Tasks

For this project, you will do the following

- Write a CPU class that can execute the instruction set listed above in C/C++/Python.
- Write the OS in GTU-C312 code to do setting up the thread table, do scheduling and handle system calls. Note that only PRN system call will be written in C/C++/Python, the rest will be in GTU-C312.
- Write at least 3 threads with your OS.
  - One thread sorts N numbers in increasing order. The number N and the numbers are given in the data segment of the program. At the end, the sorted numbers are printed.
  - One thread makes a linear search out of N numbers. The number N, the key and the other numbers are given in the data segment of the program. At the end, the found position is printed.
  - One thread will contain a program of your design. It should contain at least a loop and PRN calls.
- Write a simulation program that runs your systems with some command line parameters. There should be parameters for the program name and debug flag.
  - Simulate filename -D 1 : will read the program from filename which will include the OS the threads. In debug mode 1, the contents of the memory will be sent to standard error stream after each CPU instruction execution (memory address and the content for each adress).
  - In Debug mode 0, the program will be run and the contents of the memory will be sent to the error stream after the CPU halts.
  - In Debug mode 2, after each CPU execution, the contents of the memory will be sent to the error stream. Your simulation will wait for a keypress from the keyboard and it will continue for the next tick.
  - In Debug mode 3, after each context switch or system call, you will print the contents of the thread table to the standard error stream.

The Rules

- Please use AI based tools to write both C/Python code and GTU-C312 code. You will submit your interaction files with AI tools such as ChatGPT chats.

- Write a detailed report about your OS structure and the threads you have implemented.
- Run a number of simulations to show how your system works. Submit your outputs.
- We will post more instructions about how to submit your project
- You will demo your project live and we will ask you questions about your project.