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

# Operating Systems

# Homework #02
# Due Date: March 30th 2016
# System Calls and Processes

In this homework, we will use our GTU-C312 CPU with our new GTU-OS for simple process management. Since we will keep more than one process in the memory, we will need a very simple virtual memory such as base and limit registers decribed in your book. We will update our register table as follows

| Memory Location | Register |
|---|---|
| 0 | Program Counter |
| 1 | Stack pointer |
| 2 | Base register |
| 3 | Limit register |
| 4 | System call result |
| 5-20 | Reserved for other uses and other homework |

Each process has its own base and limit registers. When a process wants to access the memory, then the accessed address is added to the base register. If the accessed location is beyond the limit, then a segmentation error occurs.

For example, if base register contains the value of 1000 and when you say
SET -20, 100
the memory location with address (100 + 1000) is assigned a value of -20.

Your simple OS will offer only three calls.

1. CALL PRN A
   This system call prints the contents of memory address A to the console.
2. CALL FORK
   It works like the fork system call of UNIX systems. It returns the result in register 4. On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created.
3. CALL EXEC A
   It works very similar to UNIX execl systems call without the arguments. It loads the program specified in the file name and lets the scheduler execute the program. The file name is the null terminated string at address A.

In order to simulate the blocking of the process during this system calls, your OS takes a random numner of ticks (between 5 and 10) before completing this system call for this process. Of course,

the other processes can be run during this time if they are ready.

Since you have more than one process in the memory, you need to keep a process table, the process table will hold at least the following properties for each process

- The name of the process (filename)
- The PID
- The parent PID
- Starting time of the process (the tick number of the CPU)
- How many ticks the process has used so far
- The state of the process (ready, blocked, running)
- The physical address of the memory location of the process

Any other data structure can be added to the process table as needed. Note that we do not need the registers in the process table because our memory (addess 0-20) keeps them.

The following program prints numbers from 10 to 1 to the console and then it loads another program named "PRG2". Then two processes starts runing, each would print the value returned by the fork call to the console.

```
#Program sample
Begin Data Section
0 0               #program counter
1 0               # stack pointer
...
60 'P'
61 'R'
62 'G'
63 '2'
64 0
...
255 0
End Data Section
Begin Instruction Section
0 SET 10 50       # i = 10
1 ADD -1 50       # i = i - 1
2 CALL PRN 50     # Print i to the console
3 JIF 50 5        # Go to 6 if i <= 0
4 SET 2 0         # Go to 2 - remember address 0 is the program counter
5 CALL FORK       # System call fork
6 CALL PRN 4      # This line will print two numbers from each process
7 CALL EXEC 60    # load the program named PRG2 to the memory and run
8 HLT             # note that PRG2 will run two times
End Instruction Section
```

When your simulated computer starts running, your BIOS will first read a program file. One example program is above. BIOS will read both data segment and instruction segment.

Our process scheduler runs  the round robin algorithm with a quantum of 5 ticks. The process switch can happen when preemption occurs or when the process is blocked.

Your main CPU loop will be something like this

```
int i=0;
for (;!myCpu.isHalted();++i)
{
      myCpu.tick();
      // If system call handle it and block the process and reschedule
```

```
        if (i % 5 == 0)
        // The quantum is over, run the scheduler
        ....
}
```

The CPU function tick will take just one instruction at the Program Counter (PC), execute it, then increment the PC (if no jump). This loop ends when the CPU is halted.

For this homework, you will do the following

- Update your CPU class with the system calls and virtual memory addresses.
- Write a round robin scheduler that keeps the process table updated and does the context swithcing
- Your new system should automatically chose memory location for the processes loaded
- 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. In debug mode 1, the contents of the memory will be printed to the screen after each CPU tick (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 displayed after the CPU halts.
  - In Debug mode 2, every time a process switch happens, the information is printed to the screen. The information printed will be
    - the process names (both processes)
    - the CPU tick number
  - In Debug mode 3, information about each process is printed on the screen, this information will be very similar to "ps -ef" command in UNIX systems. It will include the process name and all of its process table entries
- You should test your new system with at least 4 different processes running at the same time. Try at least these processes : one process prints numbers from 100 to 1, the other makes a search in an array and prints the result, the other process add numbers from 1 to 10 and prints the results and final process prints the numbers from 800 to 100.
- Do not forget to include some examples with system errors like the fork error and segmentation fault.

We will provide the submissin instructions in a separate document. You should strictly follow these instructions otherwise your submission may not get graded.