# CS 333: Operating Systems Lab (Autumn 2018)
# Lab Quiz #4 (30 marks), $5^{th}$ November 2018

## task master

This lab is xv6-based and aims to build a parallel execution framework based on a master-worker setup. A master process issues work to be performed by worker processes.

The setup is as follows,

- A parent process declares and defines a set of tasks (functions). Creates a set of child processes/workers and then issues tasks to be executed in parallel.
- By default each child process is waiting for a task, if no task is available the child sleeps (is blocked) on a condition variable. The condition for wakeup being arrival/availability of a task.
- Note that the task issue action by master process, waiting for tasks, wake up of child processes etc. need to operate by issuing system calls and/or appropriate handling in the kernel.
- The task itself executes in the user space (in the context of some child process). A task issued from the parent process updates task/work state in the kernel, the kernel then wakes up child processes (waiting on task arrival), one of them gets the task and returns in its user space to execute the task. On task completion, a special system call returns control to the kernel, which resets the state of the child process for it to get back to its original return address in the user space (where it was blocked). Once back again in user space, it updates the statistics regarding task execution and re-enters the kernel to check if more work is required to be done, else waits/sleep on a condition variable (in the kernel).
- The master process polls/waits (sleep on condition variable inside the kernel) for the completion of a task and the corresponding result. Note that similar to task issue by parent (where task issued by parent, status of tasks maintained by kernel, and processed based on this kernel state) the result of the task execution is returned by the child process to the kernel and delivered to the parent process by the kernel via the wait for task completion call.
- Appropriate system calls, sleep-wakeup setups in the kernel, moving back and forth from the kernel to user space during task handling, status of tasks etc. are required to be implemented.

  Note that all user processes (parent or child) checking for a condition or waiting for completion (some other condition) are blocked inside the xv6 kernel and should use the `sleep` kernel function.
- Study carefully the sample programs to understand the setup and system calls executed by the parent/master and child/worker processes.
  Additionally, skeleton xv6 code is provided that mentions important variables and functions that need to be implemented. These additions are annotated as `LAB QUIZ 4` in the source code files.

Implement the following system calls, (skeleton code is available in `sysproc.c` and `syscall.h`),

- `int init_taskmaster(void);`
  Initializes the `taskmaster_t` object in the kernel—resets locks and condition variables, initializes the task array.
- `int do_task(char* fptr, int task_id, int arg1, int arg2);`
  Master (parent process) issues a task to execute using this call. A task is defined by `fptr`, pointer to a function defining the task, its arguments, and an unique task identifier (`task_id`).

  This system call *adds* a task to the task array, and wakes up all sleeping workers. In case of an error—task array is full or `task_id` is already being used for work, the system call returns `-1`.
- `int wait_for_task_to_complete(int task_id, int* result);`
  Master process waits for a task, identified by `task_id`, to complete. `result` is the output variable used to store result of the task.

  If the task is already completed, the system call returns immediately. Otherwise, process sleeps until one of the worker process completes the intended task and wakes up the master process. In case of error—task with id `task_id` does not exist, call should return `-1`.

- `int wait_for_task(void);`
  Worker (child) process issues this system call to enter the kernel and check for pending tasks.

  If a task is pending, the worker process updates necessary kernel state and proceeds to execute the task in user space. Note that the task has to be executed in the user space of the worker process. Also note that several worker processes maybe contending for this task. On task completion, control returns back to kernel from user space via the `task_ret` system call.

  If no tasks are pending, the worker process sleeps until the master process issues new work. Note that on tasks available for execution, more than one worker process may want to execute the task. Obviously, only one of them should execute a task.

- `int task_ret(int task_id, int result);`
  A special system call to return control back to the kernel on completion of task execution in user space. `task_id` is the unique identifier of the task and `result` stores return value of the task function.

  The system call stores the result of the task, updates status of the task, and wakes the master process if it is waiting for a task to complete.

---

## Submission Guidelines

- Deadline: 5th November 2018, 5.00 p.m.
- Name your submission as: **<rollno-labquiz4>.tar.gz**
- Execute the following command in the xv6 directory to clean the source code.
  `make clean`
- Use the following command to create the submission archive.
  `tar -czvf rollno-labquiz4.tar.gz rollno-labquiz4`
- The tar should contain the following files in the following directory structure:

```
<rollno-labquiz4>/
|__xv6/
|____<the xv6 directory with all files>
|__sample-outputs/
|____<sample outputs>
```

- **Verify submission archive exists in the home directory,** `/home/cs333local`.
- Upload on moodle.