# Lab 10 : Shell Lab

# What is shell?

- A program **running applications** on behalf of user and managing them
- Common example: **bash** (Linux default)
- Most applications in linux(command line) are run through shell

# What is your assignment?

- Write a simple **unix shell** (called ‘tsh’)
- Only need to write **seven specified functions**
- Helper function already provided in source file

# Tsh basic functionality

1. Run a command or application on shell
   - E.g. tsh> ls –l –h

     -> Shell forks child process

     -> executes “ls” with arguments “-l” and “-h”

* Tsh manages running application as child processes

# Tsh basic functionality

2. Foreground job management
   - Runs application in foreground and waits for its ending
   - E.g., tsh> ls –l –h

     -> shell executes “ls” with “-l –h”

     -> Wait for it to finish before other application runs

* Every application run is foreground by default

# Tsh basic functionality

3. Background job management
   - Runs application in background
   - Many simultaneous background jobs possible
   - “&” added to end of command/application name
   - E.g. tsh> ./myprogram &

* Tsh can run many jobs in the background

# Tsh basic functionality

- 4. Background/foreground management
  - Change job status (bg to fg/ fg to bg)
  - E.g. tsh> fg <job_id>

    -> Makes a background job run as foreground

*Tsh can move jobs between foreground and background

# Work in this assignment

- Many helper function coded for you
  - Parseline
  - Addjob, deletejob, clearjob, …
- Implement interesting functions
  - Eval (evaluate a command)
  - Do_bgfg (Do background/foreground switching)
  - Signal handlers, etc.

* Reuse already coded functions in tsh.c to write specified functions

# Work in this assignment

- eval: Main routine that **parses** and **interprets** the command line.
- builtin cmd: **Recognizes** and **interprets** the built-in commands: quit, fg, bg, and jobs.
- do bgfg: **Implements the bg and fg** built-in commands.
- waitfg: **Waits** for a foreground job to complete.
- sigchld handler: Catches **SIGCHILD** signals.
- sigint handler: Catches **SIGINT** (ctrl-c) signals.
- sigtstp handler: Catches **SIGTSTP** (ctrl-z) signals.

# How to evaluate your code

- Use the provided ‘reference tsh’ binary & 16 traces
- Test each tsh feature with a trace & provided “sdriver.pl”
- Your tsh output must match ‘rseference tsh’ output
- E.g. make test01

1. Unix> ./sdriver.pl –t trace01.txt –s ./tsh –a “-p”
2. Unix> ./sdriver.pl –t trace01.txt –s ./tshref –a “-p”
3. If the result of (1) and (2) match, then your code is correct.

# Assignment Submission

- Type “make handin”
- Submit your renamed tsh code to LMS

# Hints

- Re-read lecture slides about shells, signals and exceptional handling
  - ✓Best summarized reference

- Take a look at the textbook chapter 8.
- Start early

Any Questions?