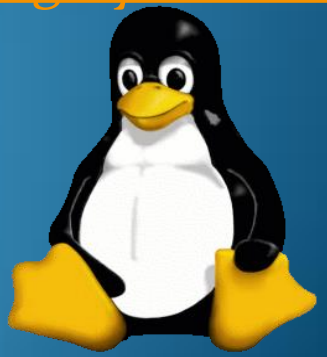
Operating System
lab5

Lab5: system calls
Instructor: Eng\ Samar Shaaban
E-mail: ssa10@fayoum.edu.eg
Email subject: **fci-os-2021**
Github Repo: https://github.com/SamarShabanCS/Operating-Sytem-2021
Slack workspace: https://fu-fci-os-2021.slack.com



OS management

- OS provides the process abstraction
  - Process: a running program
  - OS creates and manages processes
- Each process has the illusion of having the complete CPU, i.e., OS virtualizes CPU
- Timeshares CPU between processes
- OS manages the memory of the process: code, data, stack, heap
- Each process thinks it has dedicated memory space for itself.
- OS abstracts out the details of the actual placement in memory, translates from virtual addresses to actual physical addresses

What constitutes a process?

- A unique identifier (PID)
- Memory image
  - Code & data (static)– Stack and heap (dynamic)
- CPU context: registers
  - Program counter
  - Current operands
  - Stack pointer
- File descriptors
  - Pointers to open files and devices stdin, stdout ,stderr

- OS maintains a data structure (e.g., list) of all active processes.
- Information about each process is stored in a process control block (PCB)
  - Process identifier
  - Process state
  - Pointers to other related processes (parent)
  - CPU context of the process (saved when the process is suspended)
  - Pointers to memory locations
  - Pointers to open files

- API = Application Programming Interface
  = functions available to write user programs.
- API provided by OS is a set of “system calls”
  - System call is a function call into OS code that runs at a higher privilege level of the CPU
  - Sensitive operations (e.g., access to hardware) are allowed only at a higher privilege level
  - Some “blocking” system calls cause the process to be blocked and descheduled (e.g., read from disk)

- POSIX API: a standard set of system calls that an OS must implement.

- Program language libraries hide the details of invoking system calls
  - The printf function in the C library calls the write system call to write to screen
  - User programs usually do not need to worry about invoking system calls.
  - C language→ c lib→ sys calls

Process related system calls

- fork()creates a new child process
  - – All processes are created by forking from a parent
  - – The init process is ancestor of all processes
- exec()makes a process execute a given executable
- exit()terminates a process
- wait()causes a parent to block until child terminates
- Many variants exist of the above system calls with different arguments

I/O SYSTEM CALLS

- creat() Create a file for reading or writing.
- open() Open a file for reading or writing.
- close() Close a file after reading or writing.
- unlink() Delete a file.
- write() Write bytes to file.
- read() Read bytes from file.
- Seek()

Example usages of fork:

- Your shell uses fork to run the programs you invoke from the command line.
- Web servers like apache use fork to create multiple server processes, each of which handles requests in its own address space. If one dies or leaks memory, others are unaffected, so it functions as a mechanism for fault tolerance.
- Google Chrome uses fork to handle each page within a separate process. This will prevent client-side code on one page from bringing your whole browser down.
- fork is used to spawn processes in some parallel programs (like those written using MPI). Note this is different from using threads, which don't have their own address space and exist *within* a process.