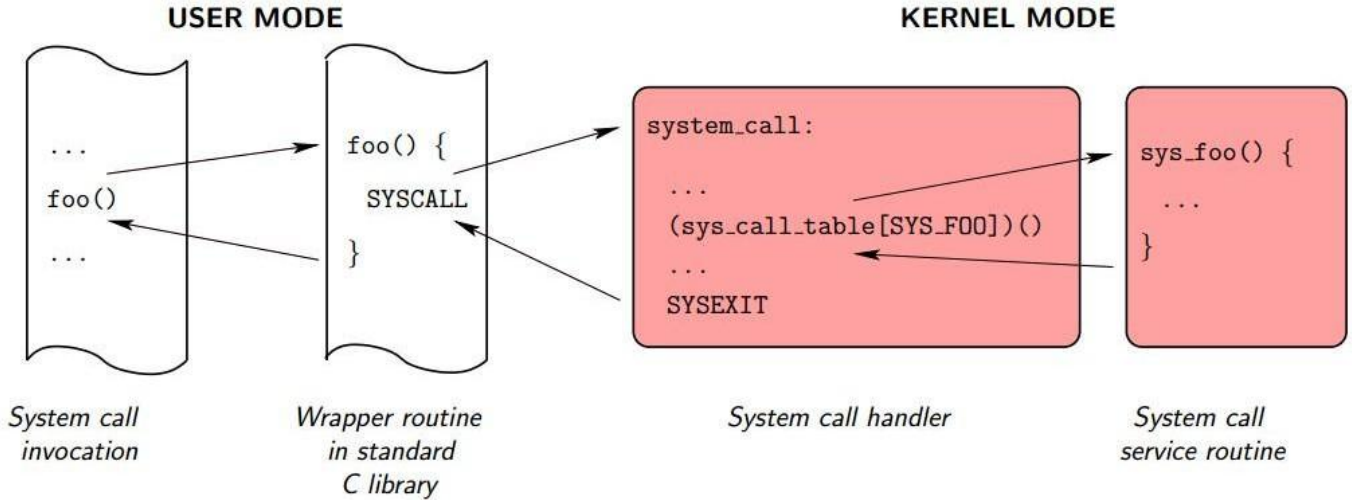
OS Tutorial 2: System Calls

Huan Wang
huanwang@uvic.ca

System Calls (1)

* kernel interfaces: services provided by the OS kernel
* Use system calls like library functions, including necessary header files (e.g., #include <unistd.h>)



System Calls (2) – Process Mgmt.

* **fork():** create a new (child) process
* **exec*():** a family of functions: execute other programs from a process
* **kill():** send signals to process
* **wait()/waitpid():** wait for a (child) process to change state
* Use man to check the details

fork()

* By calling fork(), a process can spawn a child process.
* The child duplicates the parent process, so it is almost identical to the parent (except PID, etc.)
* After fork(), both the parent and child execute the same program
* Prototype: pid_t pid = fork(); (**pid_t** is an alias to an integer type)
* Header files: #include <unistd.h> & #include<sys/types.h>
* Return values:
  * pid > 0: succeed, in parent process
  * pid == 0: succeed, in child process
  * pid < 0: fail.
* For more details: **$man 2 fork**

exec*() (1)

- A family of six functions:
  - int execl(char *path, char *arg, ...);
  - int execv(char *path, char *argv[]);
  - int execle(char *path, char *arg, ..., char *envp[]);
  - int execve(char *path, char *argv[], char *envp[]);
  - int execlp(char *file, char *arg, ...);
  - int execvp(char *file, char *argv[]);
- What ***l***, ***v***, ***e***, and ***p*** mean:
  - ***l*** means an argument list,
  - ***v*** means an argument vector,
  - ***e*** means an environment vector, and
  - ***p*** means a environment path.
- For more details: **$man 3 exec**

exec*() (2)

* load and run a new program so as to replace the current process
* Upon success, **exec()** never returns to the caller unless the call failed.
  * Failed reasons: non-existent file (bad path) or bad permissions

kill()

* Send signals to a process specified by PID
* Prototype:
  * int kill(pid_t pid, int signal);
  * E.g., int retVal = kill(child_pid, SIGTERM);
* Header files: #include <sys/types.h> & #include <signal.h>
* Return values:
  * On success: retVal = 0
  * On error: retVal = -1
* Linux signals:
  * **SIGSTOP**, **SIGCONT**, **SIGTERM**, **SIGKILL**, etc.
  * **$ man 7 signal**

wait

* Forces the parent to suspend execution and wait for its children or a specific child to change states/state.

  Two forms:
  * wait(): parent waits for any child process
* Prototype:
  * pid_t wait(int *status);

* Header file: #include <sys/types.h> & #include <sys/wait.h>
* Return values:
  * On success: PID of the exited process
  * On error: -1

waitpid()

* Parent waits for a state change of a child with given PID
* pid_t waitpid(pid_t pid, int *wstatus, int options);
* Options:

* If 0, then waits until the specified child return
* WNOHANG - return immediately if no child has exited
* WUNTRACED - also returns if a child has stopped
* WCONTINUED - also returns if a stopped child has been resumed by SIGCONT

* return value:

* return the PID of the child whose state has changed
* return 0 if WNOHANG was specified but child have not yet changed state
* return -1 on error

Note: the parent can only wait its **direct child** (**NOT** grandchild) * For more details: **$man 2 wait**

exit()

* Gracefully terminates process execution: clean up and release resources; puts the process into zombie state.
* Prototype: void exit(int status);
* Header file: #include <stdlib.h>
* **exit()** specifies a return value from the process (i.e., status of the dead process), which a parent process may need to examine.
* **_exit()** call is another possibility of quick death without cleanup.
* For more details: **$ man 3 exit** & **$ man 2 _exit**

Background Execution

* Switch a program from the foreground to the background or vice-versa.
* Linux commands:
  * **Ctrl + C**: Terminate the foreground process and return to Shell
  * **Ctrl + Z**: Suspend the foreground process, send it to background and return to Shell
  * **&**: Let the program run at background
  * **fg [num]**: Move the process with job ID=num to foreground
  * **bg [num]**: Move the process with job ID=num to background
* Online tutorial: http://www.thegeekstuff.com/2010/05/unix-background-job/

Outline

* **System call (Questions?)**

Contributors:

* Cheng Chen, Guoming Tang, Yongjun Xu
* Huan Wang, Changli Zhang