OS Lab Tutorial 2

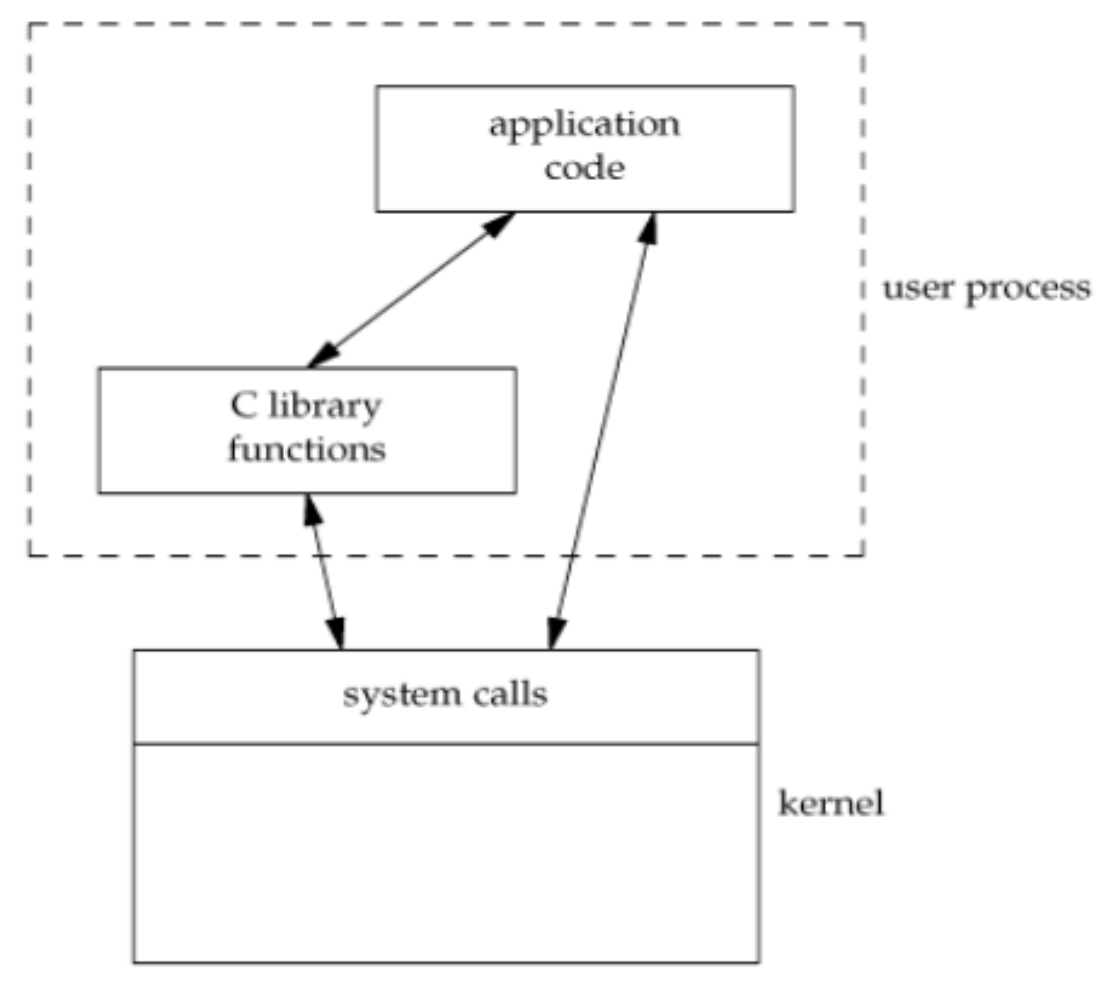
System Calls

System Calls

- Services provided by the kernel
- Documented in Section 2 of the Manual Page
- Each system call has a function of the same name in the standard C library
- You can use system calls just like the library functions
  - Includes the necessary header files

System Calls

- Difference between system calls and library functions



- Reference:
  - *Advanced Programming in the UNIX Environment*

First 3 Sections in Manual Page

- Section 1: user commands
  - $ man 1 ls
  - $ man 1 printf
- Section 2: system calls
  - $ man 2 fork
- Section 3: general-purpose functions to programmers
  - $ man 3 printf
  - $ man 3 exec() family of functions

System Calls - File System

- **mkdir()**: create a new directory
- **rmdir()**: remove a directory
- **getcwd()**: get the current directory
- **chdir()**: change directory
- **getenv()**: return the value of a PATH variable
  - getenv("HOME")
- Use man page to view the details
  - $ man 2 chdir
  - $ man 2 getcwd

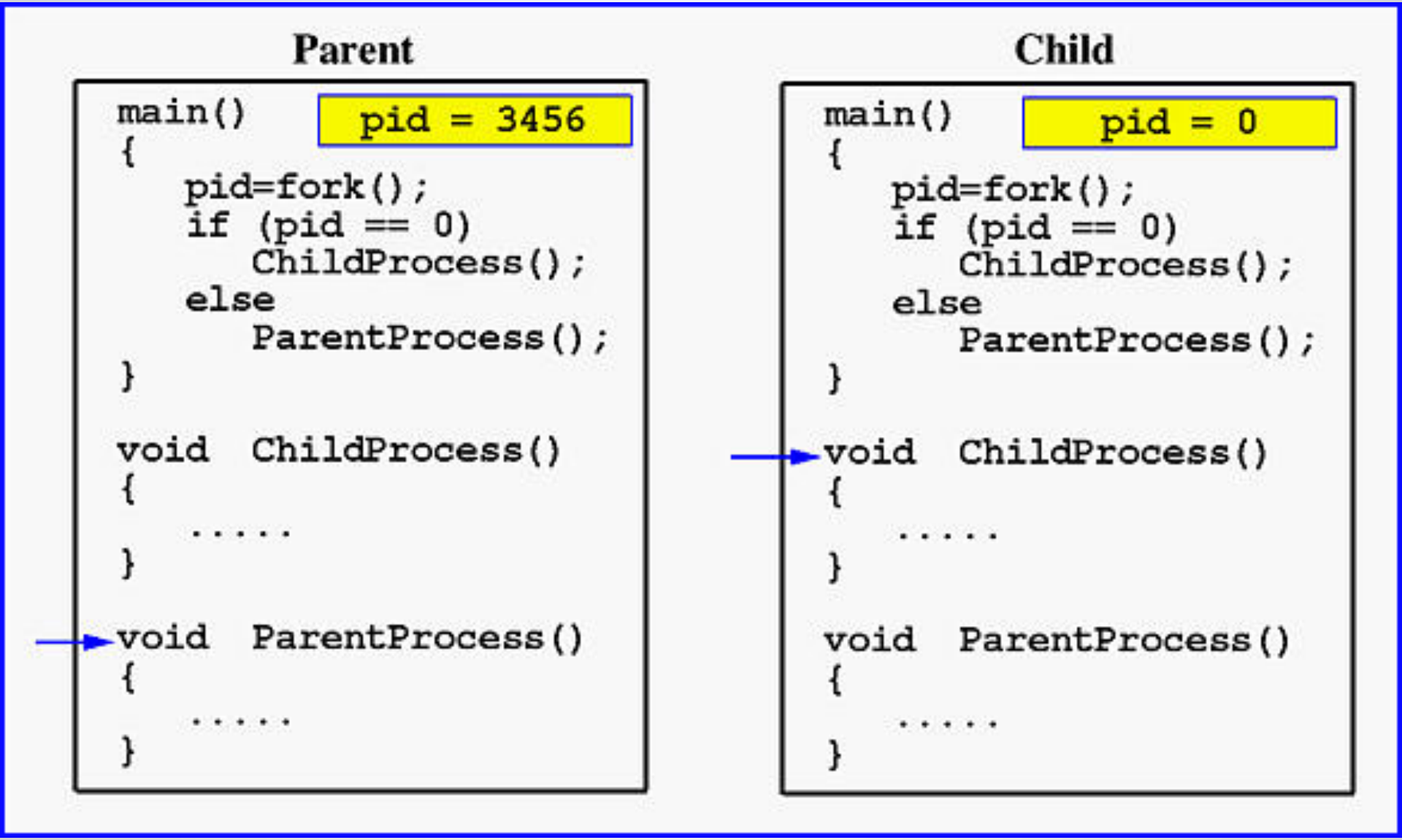
System Calls - Process Management

- **fork()** - create new process
- **getpid()** - return the process id of the calling process
- **getppid()** - return the parent of the calling process
- **exec()** - family of functions: execute programs
- **kill()** - send signals to process
- **waitpid()** - wait for process to change state
- Use man page to view the details
  - exec() functions (except execve()) are actually in the library section. But here we still regard them as system calls.

System Calls - fork()

- #include <unistd.h>
- After a successful call of fork():
  - A new process is created (child process)
  - Memory space of parent process is copied to child process (So they have the same code and share some of the data)
  - child process starts from the call of fork()
  - Other details, $ man 2 fork
- One Call, Two Returns
  - Parent gets child's pid (>0), Child gets "0"
  - If the return value is negative, the call fails
  - We can write different code for child and parent process

System Calls - fork()



System Calls - fork()

- A block of code in parent process

```
pid_t retValue = fork();                // Call of fork() in parent process

if(retValue == 0)          // Child gets '0' as the return value of fork()
 {
     printf("I'm child process.\n"); // Code for child's behavior
}
else if (retValue > 0)    // Parent gets the pid of child
{
     printf("I'm parent process.\n"); // Code for parent's behavior
}
else                              // Otherwise, it fails to create new proces
{
     printf("Creating Fails.");          // Check errors
}
```

- Which header file do we need for the type "pid_t" ?

System Calls - exec() functions

- We don't have exec(). Instead, we have six functions ($ man exec)

- The family members of exec() are used to load and run a new program so as to replace the current calling process

- exec() functions are often used with fork() so that child process can execute a new program in the child's memory space

An example of fork() and exec()

- We create a new process using fork and execute a Linux command by one of exec()
- Prototype of execv()
  - int execv(const char *path, char *const argv[]);

```
# include <stdio.h>
# include <unistd.h>
int main(void)
{
    char *const parmList[] = {"ls", "-l", NULL};
    if (fork() == 0)
        if (execv("/bin/ls", parmList) < 0)
            perror("Error on execv.");
    return 0;
}
```

- **NOTE**: Always check the return value.

System Calls - waitpid()

- The calling process will wait for a state change in its child process
- Prototype:
  - header files: "sys/types.h", "sys/wait.h"
  - pid_t waitpid(pid_t pid, int * status, int options)
  - e.g., pid_t retVal = waitpid(child_pid, NULL, WNOHANG);
- Options
  - If 0, then the parent waits until the child returns
  - WNOHANG - Parent process can return immediately if no child has existed
- Exercises
  - Use man page to learn how the three parameters control the behavior of waitpid() (E.g., (1) Different pids lead to different actions; (2) What's the meaning of its return values?; (3) What are the other options?)

System Calls - kill()

- Send signals to process specified by pid
- prototype
  - header files: "sys/types.h", "signal.h"
  - int kill(pid_t pid, int signal);
  - E.g., int retVal = kill(child_pid, SIGTERM);
- Exercises
  - Read man page for kill() and learn how to set pid parameter
    - $ man 2 kill
  - For more signals, you can read "signal.h" or the man page for signals
    - $ man 7 signals
  - What' the difference between SIGKILL and SIGTERM ?

Background Execution and Process Management

- 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
  - **jobs**: List background process and their job ID
  - **&**: 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
- A nice online tutorial here

popen, pclose - Communication between Process

- FILE *popen(const char *command, const char *type);
- int pclose(FILE *stream);
- The popen() function opens a process by creating a pipe, forking, and invoking the shell.
- A pipe is by definition unidirectional, the type argument may specify either reading or writing, not both
  - the resulting stream is correspondingly read-only or write-only
  - Create two pipes for a pair of processes, one for "read", one for "writing"
- Samples popen_r.c (read-only pipe) and popen_w.c (write-only pipe)

Other Useful Library Functions

- **gettimeofday(), time(), localtime()** : get the current date time
- **getenv()**: get the environment variable
  - getenv("PATH");
- **getlogin()**: get the current username
- **gethostname()**: get the name of this machine
- Use man page to view the details
  - $ man localtime
  - $ man getlogin

System Calls Tool - strace

- Used to trace system calls in your program.

```
/* test.c */
# include <stdio.h>
int main(void)
{
    printf("Hello World.\n");
}
```

```
$ gcc -Wall test.c -o test
$ strace ./test
```

- An interesting and helpful article by Bill Zimmerly
  - http://www.ibm.com/developerworks/aix/library/au-unix-strace.html?S_TACT=105AGX52&S_CMP=cn-a-aix

Post Questions in Discussion