ECS 150 - System Calls

Assignment Project Exam Help

Joël Porquet

https://powcoder.com

UC Davis - Spring Quarter 2017

Add WeChat powcoder

Readings: *OSPP Chap 3*

Copyright © 2017 Joël Porquet - CC BY-NC-SA 4.0 International Licence

System calls

Example

```c
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd, nread;
    char buf[1024];

    fd = open("/path/to/myfile", 0);
    nread = read(fd, buf, 1024);

    ...

    close(fd);

    return 0;
}
```

Assignment Project Exam Help

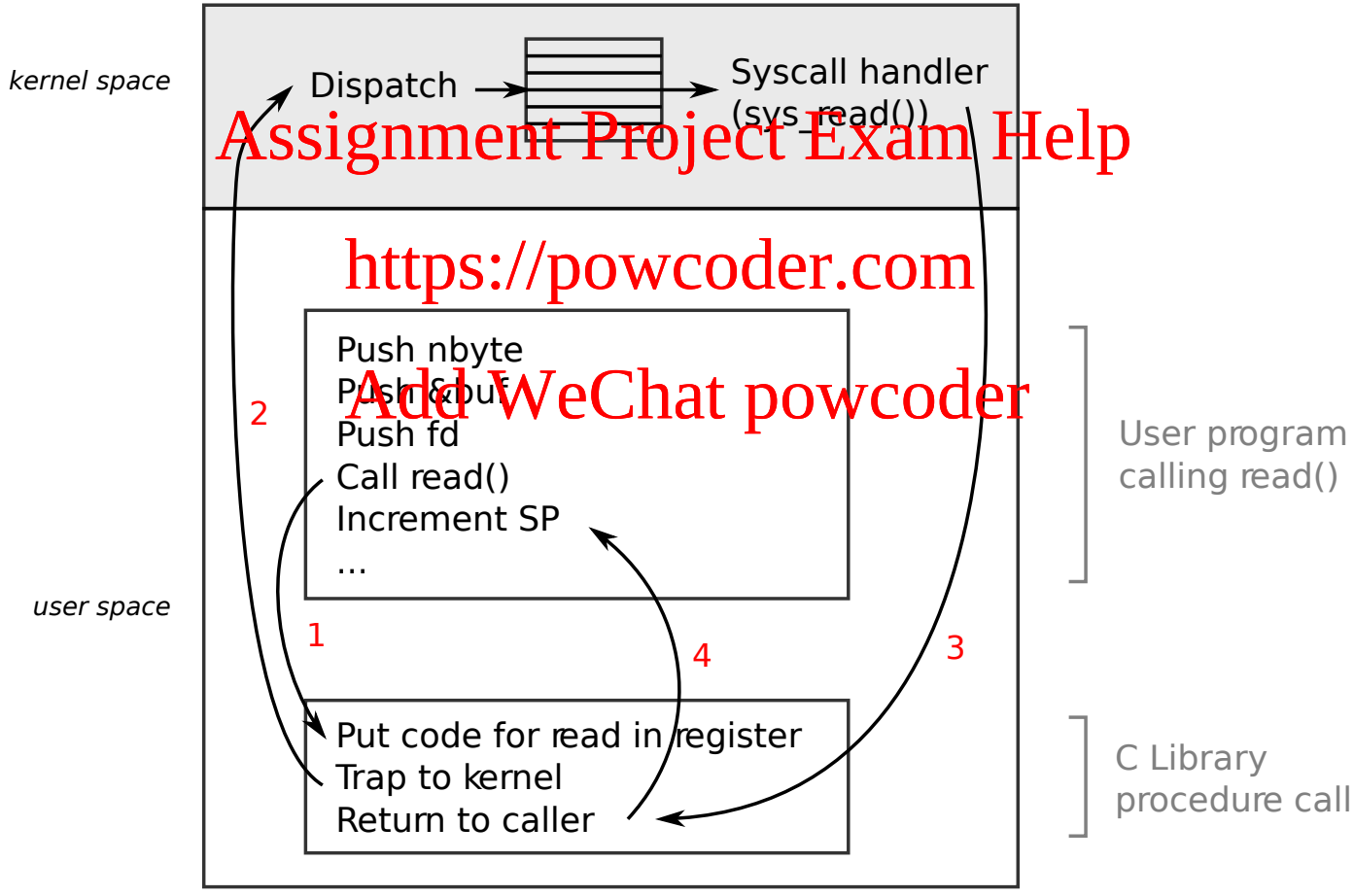
https://powcoder.com

Add WeChat powcoder

System calls

Overview

- API to the kernel
- Hide kernel and hardware complexity from applications



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

System calls

Inter-communication

How the system calls communicate back to applications?

Return value

- Usually -1 on error, >= 0 on success
- C library functions set global variable `errno` to encode the error
  - E2BIG, EACCESS, EAGAIN, EBADF, ENOMEM, ... (`man errno`)
  - Use `perror()` to display a string

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
void *ptr = malloc(1);
if (!ptr) {
    int errno_save = errno;
    perror("malloc");
    fprintf(sderr, "malloc: %s\n", strerror(errno_save));
    exit(EXIT_FAILURE);
}
```

Buffers

- Part of system calls arguments
- Values need to be copied between user and kernel space

System calls

- Process
- Files
- Pipe
- Signals
- Memory

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Process

Definition

- *A process is a program in execution*
- Each process has its own memory space and process control block
- **Process control block**
  - Kernel structure
  - Stores all information associated with a process
  - Register values for context switching, open files, user ID, group ID, etc.
- Processes are indexed by the process ID (**PID**)

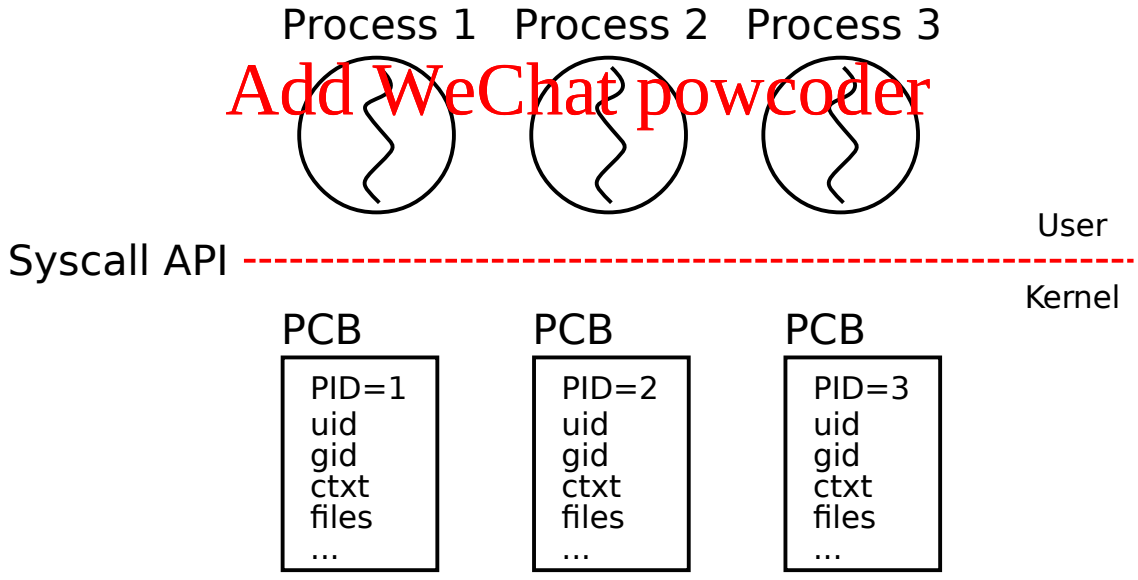
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Process

Definition

- *A process is a program in execution*
- Each process has its own memory space and process control block
- **Process control block**
  - Kernel structure
  - Stores all information associated with a process
  - Register values for context switching, open files, user ID, group ID, etc.
- Processes are indexed by the process ID (**PID**)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Process

Related system calls

- `fork()`: Create a new process
- `exec()`: Change executed program in process
- `exit()`: End process
- `wait()/waitpid()`: Wait for a child process and collect exit code
- `getpid()`: Get process PID
- `getpgrp()`: Get process GID

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Process

fork()

- Syntax: `pid = fork()`
- The child gets almost identical copy of the parent
- File descriptors, arguments, memory, stack, etc. are all copied
- Even the program counter
- Only one thing differs... Which one?

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Process

Forgetful forking

```
int main(int argc, char **argv)
{
    fork();
    prinft("I am the process!\n");
    return 0;
}
```

What gets printed?

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Process

Forgetful forking

```
int main(int argc, char **argv)
{
    fork();
    prinft("I am the process!\n");
    return 0;
}
```

What gets printed?

Assignment Project Exam Help

```
$ ./simple-fork
I am the process!
I am the process!
```

https://powcoder.com

Add WeChat powcoder

Process

Forgetful forking

```
int main(int argc, char **argv)
{
    fork();
    prinft("I am the process!\n");
    return 0;
}
```

What gets printed?

Assignment Project Exam Help

```
$ ./simple-fork
I am the process!
I am the process!
```

https://powcoder.com

Using the return value

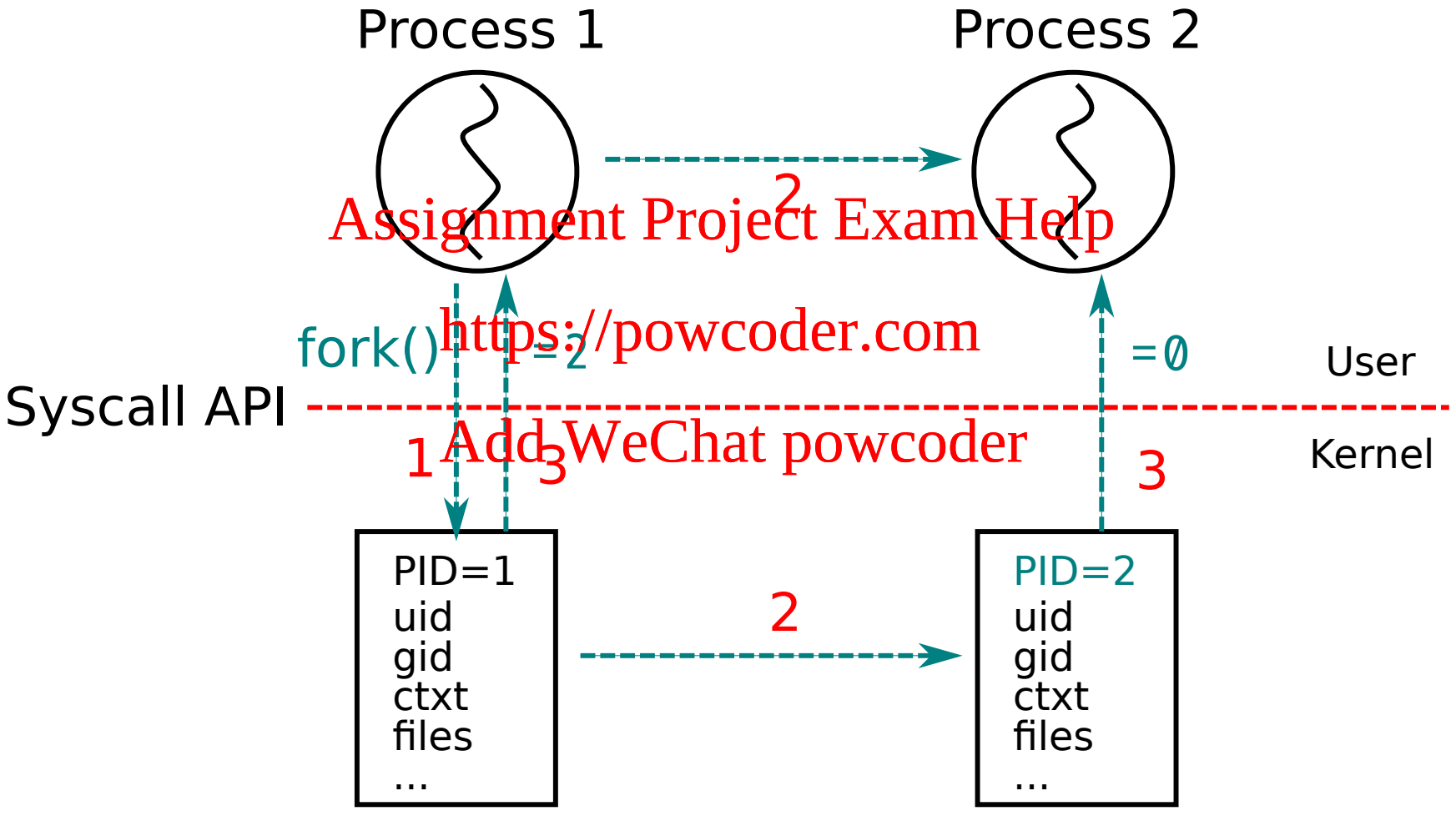
Add WeChat powcoder

`fork()` returns:

- **zero** for the child
- **PID of the child** for the parent
- (`-1` in case of error)

Process

Fork illustrated



Process

Fork example

```
int main(int argc, char **argv)
{
    pid_t pid;
    pid = fork();
    if (pid > 0)
        prinft("I am the parent!\n");
    else if (pid == 0)
        prinft("I am the child!\n");
    else
        printf("I am the initial process! But something went wrong...");

    printf("I am here now!");
    return 0;
}
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

What gets printed? (assuming everything goes fine)

Process

Fork example

```
int main(int argc, char **argv)
{
    pid_t pid;
    pid = fork();
    if (pid > 0)
        prinft("I am the parent!\n");
    else if (pid == 0)
        prinft("I am the child!\n");
    else
        printf("I am the initial process! But something went wrong...");

    printf("I am here now!");
    return 0;
}
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

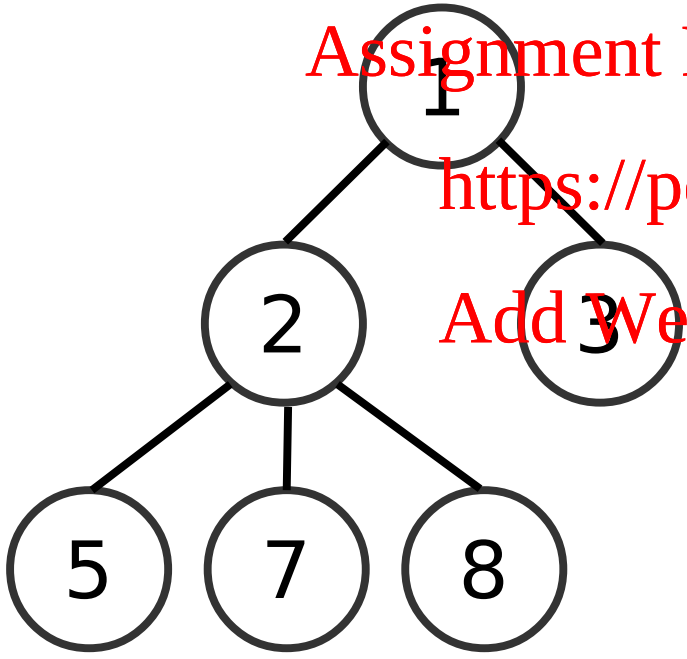
What gets printed? (assuming everything goes fine)

```
$ ./complete-fork
I am the parent!
I am here now!
I am the child!
I am here now!
```

Process

Process hierarchy

- Notion of a hierarchy (tree) of processes
- Each process has a single parent
- In Unix, all user processes have *init* as their ultimate ancestor



- Additional ways to group processes:
  - Process groups (signalling)
  - Sessions (job control)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Process

exec()

- Change executed program in current process
- Several different forms with slightly different syntax:
  - `exec[lv]p?e?()` (see man page for details)

```
void exec_ls(void)
{
    char *cmd = "/bin/ls";
    char *args[] = { cmd, ".", NULL };
    char *env[] = { NULL };

    status = execve("/bin/ls", args, env);

    printf("Did everything go well?\n");
    printf("status=%d\n", status);
}
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Will the `printf()`'s be executed?

Process

wait()/waitpid()

- When a process is done, it can either explicitly call `exit(status)` or just return and the exit call will be done implicitly

```
int retval = main(argc, argv);
exit(retval);
```

- The status is what $? reflects in the shell
- A parent can wait for its children (and by default blocks until they are done)

```
int statloc;
waitpid(pid, &statloc, options);
wait(&statloc);
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Process

wait() example

```
pid = fork();
if (pid == 0) {
    /* child */
    prinft("I'm the child and I will die soon!\n");
    exit(42);
} else {
    /* parent */
    int status;
    wait(&status); /* could also be waitpid(pid, &status, 0) */
    printf("Child exited with return code %d\n", WEXITSTATUS(status));
}
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

What gets printed?

Process

wait() example

```
pid = fork();
if (pid == 0) {
    /* child */
    prinft("I'm the child and I will die soon!\n");
    exit(42);
} else {
    /* parent */
    int status;
    wait(&status); /* could also be waitpid(pid, &status, 0) */
    printf("Child exited with return code %d\n", WEXITSTATUS(status));
}
```

Assignment Project Exam Help
https://powcoder.com

What gets printed?

Add WeChat powcoder

```
$ ./fork-wait
I am the child and I will die soon!
Child exited with return code 42
```

Process

Putting it together: fork()+exec()+wait()

```c
int main(int argc, char **argv)
{
    pid_t pid;
    int status;
    char *cmd[3] = { "ls", ".", NULL, };

    pid = fork();
    if (pid == 0) {
        /* Child */
        execvp(cmd[0], cmd);
        perror("execvp");
        exit(1);
    } else if (pid > 0) {
        /* Parent */
        waitpid(-1, &status, 0);
        printf("Child exited with status: %d\n", WEXITSTATUS(status));
    } else {
        perror("fork");
        exit(1);
    }
    return 0;
}
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Process

Full example: the shell

- A **Shell** is a programs that typically make heavy use of process system calls
- Basic cycle:
    1. Display prompt
    2. Read line from input
    3. Parse line
    4. Fork
        1. Child executes the command
        2. Parent waits
- Handle background jobs (`&`)
- Handle redirections (`<` and `>`) and pipes (`|`) for connecting `stdin` or `stdout` of the child to files or other processes

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Process

Shell skeleton

```
while(1) {                              /* Repeat forever */
    char **command;

    display_prompt();                   /* Display prompt in terminal */
    read_command(&command);             /* Read input from terminal */

    if (fork() != 0) {                  /* fork off child process */
        /* Parent */
        waitpid(-1, &status, 0);        /* wait for child to exit */
    } else {
        execv(command[0], commands);    /* execute command */
        perror("execv");                /* coming back here is an error */
        exit(1);
    }
}
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder