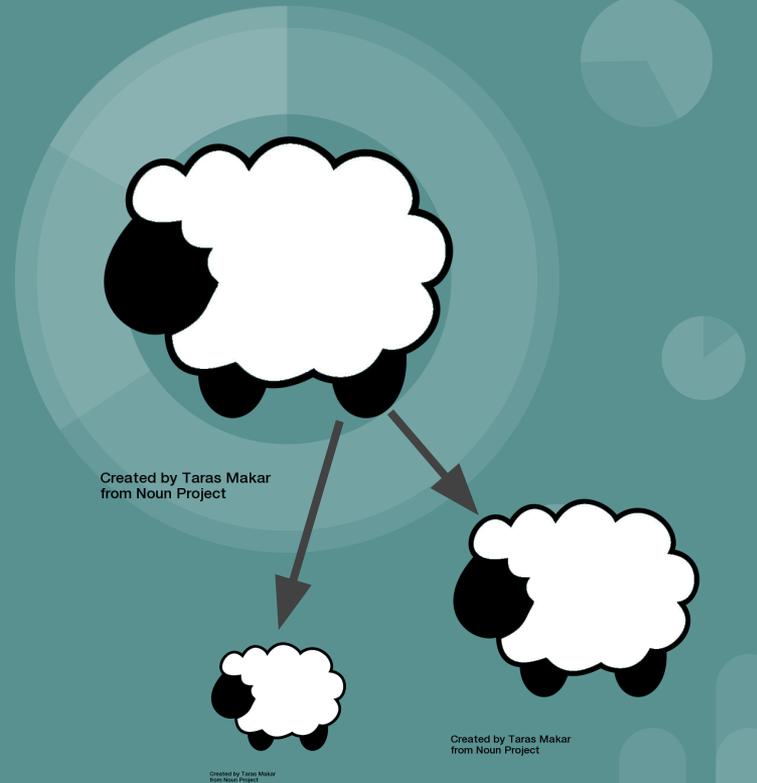


# Hello Dolly

Forks, zombies, orphans, pipes, exec...

Hannes Rabo, Katariina Martikainen, Charlotta Spik  
With credit to Isabel Ghourchian who's slide text from last year  
was used for this presentation  
2018-11-08  
KTH Kista, Stockholm



Read about the (not strictly related) sheep  
Dolly here:  
[https://en.wikipedia.org/wiki/Dolly\\_\(sheep\)](https://en.wikipedia.org/wiki/Dolly_(sheep))



# Intro - processes



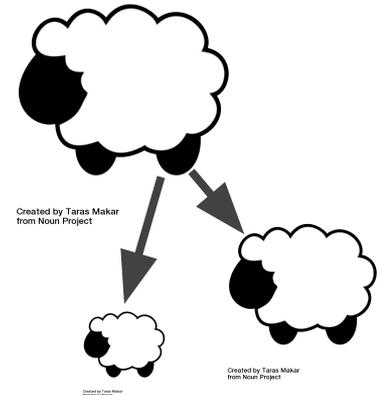
Created by Tessa Maker  
from Noun Project

- A process is a running program, an execution of instructions and a set of resources
- Can be represented by a unique number called process id (pid)



# fork()

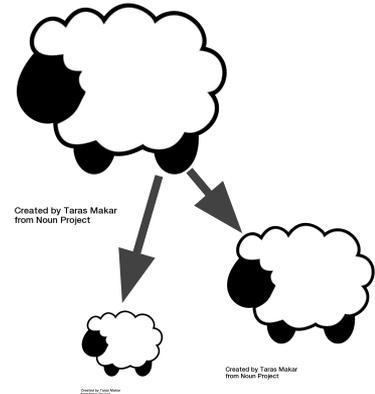
- Function that creates a new process
- Creates a copy (child process) of the current process (parent process)
- Returns twice
  - child's pid to the parent
  - 0 to the child
- Child process has its own address space





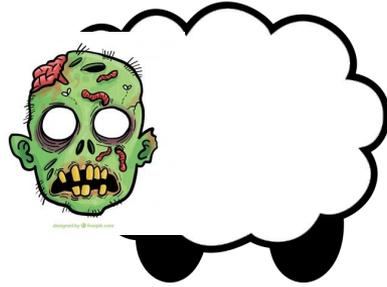
## Example – fork()

```
int pid = fork();  
if (pid == 0) {  
    printf("I'm the child");  
}  
else {  
    printf("I'm the parent ");  
}
```





# Zombie



Created by Taras Makar  
from Noun Project

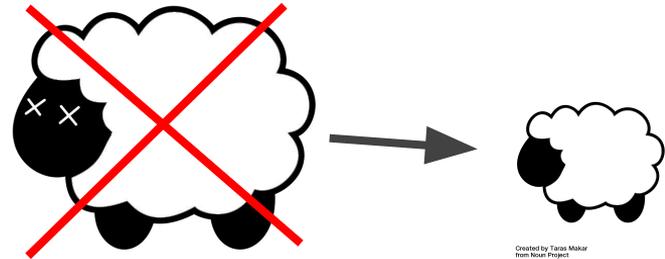
- Child process that has finished executing but still exists in the process table
- Happens if parent does not call wait()

```
else {                                     //parent process
    printf("I'm the parent");
    wait(NULL);                             //wait for child process to terminate, prevent zombie
}
```



# Orphans

- Child process that is still running but the parent process has finished executing or terminated
- Will be "adopted" by another process, get a new parent process
- In UNIX the new parent process is the init or systemd process
  - parent of all processes
  - process id 1





# Daemon

- Process that runs automatically in the background instead of under direct control of a user
- Performs operations at predefined times or in response to events
- Runs most of the tasks in a system

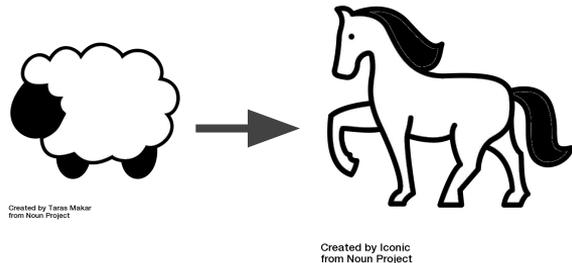


Created by Taras Makar  
from Noun Project



# exec()

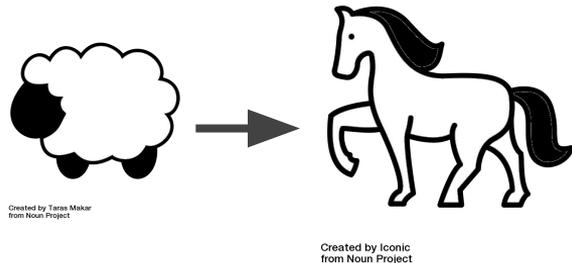
- Runs an executable file in the context of an existing process, replacing the previous execution context
- `execlp("ls", "ls", NULL);`
  - check if 'ls' exists
  - run it with the rest of the arguments
- exec functions do not return when successful





# exec() family of functions

- `execl`, `execlp`, `execle`, `execv`, `execvp`, `execvpe`
  - e: pass an array of pointers to environment variables
  - l: arguments are passed individually to the function
  - v: arguments are passed as an array of strings
  - p: uses PATH environment variable to find the file that is to be executed





# Pipes

- Sends the output of one program to another programs input
- Denoted by symbol '|'
- Piping in the shell: combine several commands
  - Ex: `cat countries.txt | grep a | sort`
  - Displays all countries that start with an 'a' and sorts them in alphabetical order



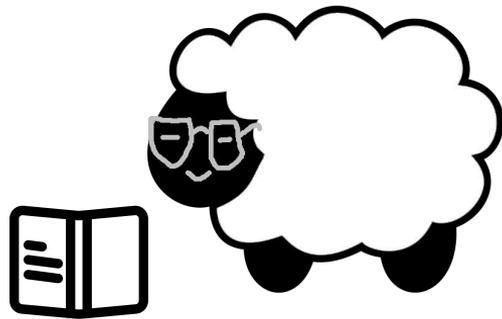


# Exam question

## 1.3 Arghhh! [2 points]

Assume that we have a program `boba` that writes “Don’t get in my way” to `stdout`. What will the result be if we run the program below and why is this the result? (the procedure `dprintf()` takes a file descriptor as argument)

```
int main() {  
  
    int fd = open("quotes.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);  
  
    int pid = fork();  
  
    if (pid == 0) {  
        dup2(fd, 1);  
        close(fd);  
        execl("boba", "boba", NULL);  
    } else {  
        dprintf(fd, "Arghhh!");  
        close(fd);  
    }  
    return 0;  
}
```



Created by Gary  
from Noun Project

Created by Taras Makar  
from Noun Project



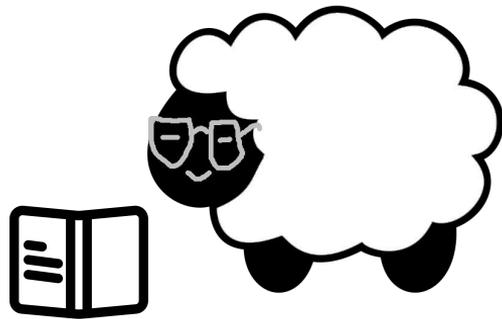
# Exam question

## 1.3 Arghhh! [2 points]

Assume that we have a program `boba` that writes “Don’t get in my way” to `stdout`. What will the result be if we run the program below and why is this the result? (the procedure `dprintf()` takes a file descriptor as argument)

```
int main() {  
  
    int fd = open("quotes.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);  
  
    int pid = fork();  
  
    if (pid == 0) {  
        dup2(fd, 1);  
        close(fd);  
        execl("boba", "boba", NULL);  
    } else {  
        dprintf(fd, "Arghhh!");  
        close(fd);  
    }  
    return 0;  
}
```

**Answer:** In `dup2(fd,1)` we redirect `stdout` to the opened file. Boba will write its line to the file `quotes.txt`. At the same time the mother process will write “Arghhh!” to the same file. The two processes will share the file current position and combine the write operations. The result is a mixture of the two texts in the file `quotes.txt` i.e. the texts will not overwrite each other.



Created by Gato  
from Noun Project

Created by Taras Makar  
from Noun Project



## Exam question: What is the value of count after?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

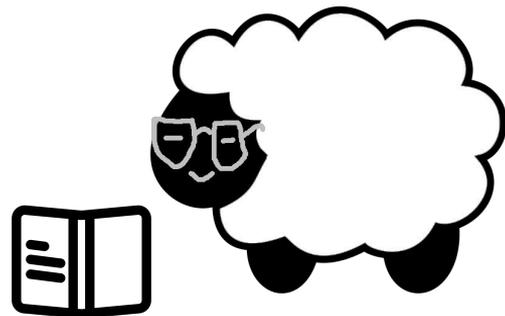
int count = 0;

int main() {

    int *status;

    int pid = fork();

    if( pid == 0) {
        for (int i = 0; i < 10; i++) {
            count += 1;
        }
        return 0;
    } else {
        for (int i = 0; i < 10; i++) {
            count += 1;
        }
        wait(status);
    }
    printf("count = %d\n", count);
    return 0;
}
```



Created by Guru  
from Noun Project

Created by Taras Makar  
from Noun Project



## Exam question: What is the value of count after?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int count = 0;

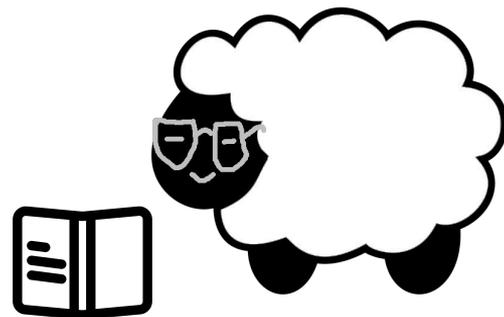
int main() {

    int *status;

    int pid = fork();

    if( pid == 0) {
        for (int i = 0; i < 10; i++) {
            count += 1;
        }
        return 0;
    } else {
        for (int i = 0; i < 10; i++) {
            count += 1;
        }
        wait(status);
    }
    printf("count = %d\n", count);
    return 0;
}
```

As both processes have their own virtual memory which means that the value of count will be **10** for both of them.



Credited by Guru  
from Noun Project

Created by Taras Makar  
from Noun Project



# Exam question: How does pipes work?

## 2.2 pipes [2 points]

If we have two processes, one producer and one consumer, that are communicating through a so called *pipe*. How can we then prevent that the producer sends more information than the consumer is ready to receive and thereby crash the system.



# Exam question: How does pipes work?

## 2.2 pipes [2 points]

If we have two processes, one producer and one consumer, that are communicating through a so called *pipe*. How can we then prevent that the producer sends more information than the consumer is ready to receive and thereby crash the system.

**Answer:** Pipes have built-in flow control. If the consumer does not read from the pipe the producer will be suspended when it tries to write the filled pipe.