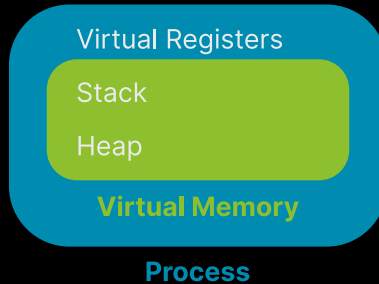


# Process Creation

2024 Fall ECE 344: Operating Systems  
Jon Eyolfson

Lecture 4  
2.0.1

## Recall: A Process is an Instance of a Running Program



## We Can Add More to a Process

Virtual Registers

Stack

Heap

Global Variables

**Virtual Memory**

Open File Descriptors

**Process**

## A Process Control Block (PCB) Contains All Information

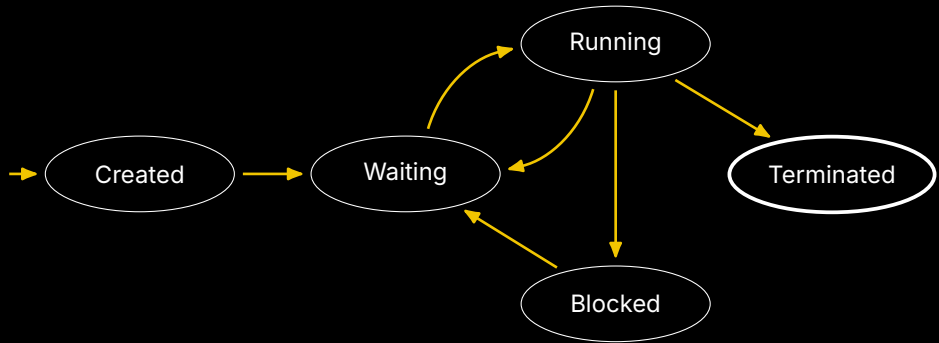
Specifically, in Linux, this is the `task_struct` you can browse on [GitHub](#)

It contains:

- Process state
- CPU registers
- Scheduling information
- Memory management information
- I/O status information
- Any other type of accounting information

Each process gets a unique process ID (pid) to keep track of it

## Process State Diagram (You Could Rename Waiting to Ready)



## You Can Read Process State Using the “proc” Filesystem

There's a standard /proc directory (on Linux) that represents the kernel's state

These aren't real files, they just look like it!

Every directory that's a number (process ID) in /proc represents a process

There's a file called status that contains the state (used for Lab 1)

## **We Could Create Processes from Scratch**

We load the program into memory and create the process control block  
(this is what Windows does)

Unix decomposes process creation into more flexible abstractions

## **Instead of Creating a New Process, We Could Clone It**

Pause the currently running process, and copy it's PCB into a new one

This will reuse all of the information from the process, including variables!

Distinguish between the two processes with a parent and child relationship

They could both execute different parts of the program together

We could then allow either process to load a new program and setup a new PCB



## fork Creates a New Process, A Copy of the Current One

`int` `fork(void)` as the following API:

- Returns the process ID of the newly created child process
  - 1: on failure
  - 0: in the child process
  - >0: in the parent process

There are now 2 processes running

Note: they can access the same variables, but they're separate  
Operating system does "copy on write" to maximize sharing

## On POSIX Systems, You Can Find Documentation Using `man`

We'll be using the following APIs:

- `fork`
- `execve`
- `wait` (next lecture)

You can use `man <function>` to look up documentation,  
or `man <number> <function>`

2: System calls

3: Library calls

## fork-example.c Has One Process Execute Each Branch

```
int main(int argc, char *argv[]) {
    pid_t returned_pid = fork();
    if (returned_pid == -1) {
        int err = errno;
        perror("fork failed");
        return err;
    }
    if (returned_pid == 0) {
        printf("Child returned pid: %d\n", returned_pid);
        printf("Child pid: %d\n", getpid());
        printf("Child parent pid: %d\n", getppid());
    }
    else {
        printf("Parent returned pid: %d\n", returned_pid);
        printf("Parent pid: %d\n", getpid());
        printf("Parent parent pid: %d\n", getppid());
    }
    return 0;
}
```

## **execve Replaces the Process with Another Program, and Resets**

execve has the following API:

- `pathname`: Full path of the program to load
- `argv`: Array of strings (array of characters), terminated by a null pointer  
Represents arguments to the process
- `envp`: Same as `argv`  
Represents the environment of the process
- Returns an error on failure, does not return if successful

## execve-example.c Turns the Process into ls

```
int main(int argc, char *argv[]) {
    printf("I'm going to become another process\n");
    char *exec_argv[] = {"ls", NULL};
    char *exec_envp[] = {NULL};
    int exec_return = execve("/usr/bin/ls", exec_argv, exec_envp);
    if (exec_return == -1) {
        exec_return = errno;
        perror("execve failed");
        return exec_return;
    }
    printf("If execve worked, this will never print\n");
    return 0;
}
```

## **The Operating System Creates Processes**

The operating system has to:

- Maintain process control blocks, including state
- Create new processes
- Load a program, and re-initialize a process with context