Lecture 24: Processes

CSE 29: Systems Programming and Software Tools

Aaron Schulman (Shalev)

UCSDCSE

Computer Science and Engineering

Process: A running program on an OS

- When you run a program on an OS (e.g., start in shell), it starts a “process”
- A process is an abstraction provided by the OS of a single isolated running program
- A process includes all of the **context** of the program
  - Memory: Stack/Heap and Code
  - Registers: Current state of execution (e.g.. what instruction in the code is running)
  - Peripherals: Even disk access!
- Processes are completely isolated from each other:
  - You can not overwrite another processes’ stack/heap or code!

Process State: What does an OS maintain?

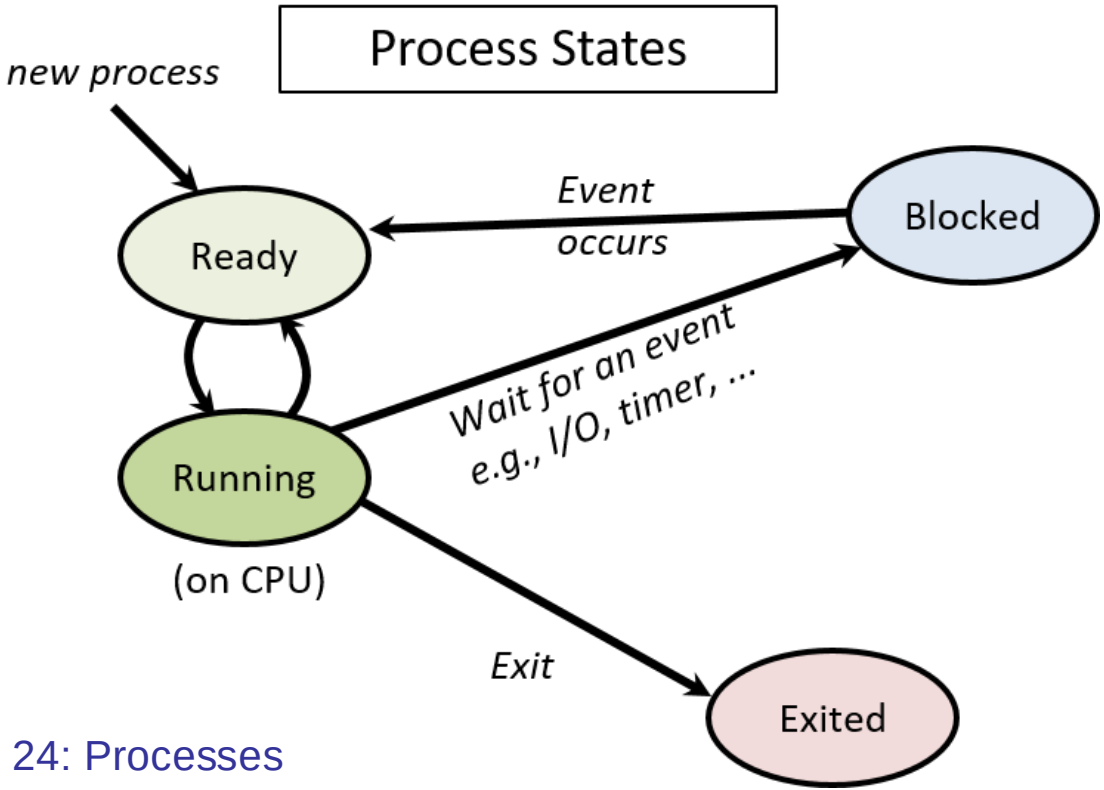
- Process ID: PID – A unique number for a process
- The address space (range) in RAM for the process
- The execution state of the process (i.e., Registers)
- The hardware resources in use (e.g., open files on disk)

Context Switching: Changing processes

- The Operating System decides when to switch between processes
  - This is known as "**Context Switching**"
  - The this is one of the big benefits of the process abstraction

- When an OS decides that a process had its time to execute it performs the context switch
  1. The OS saves all context of the running process by copying all Registers to RAM
  2. The OS then copies the saved context of another process from RAM to the Registers
  3. The new context switched process begins!

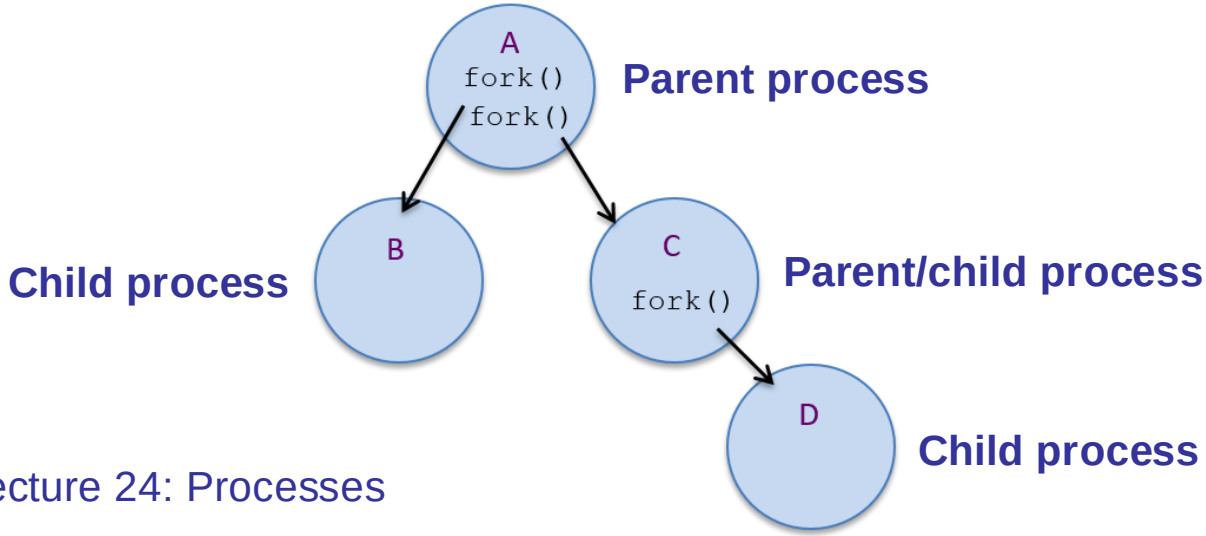
State diagram of a process in the OS



Creating and destroying processes

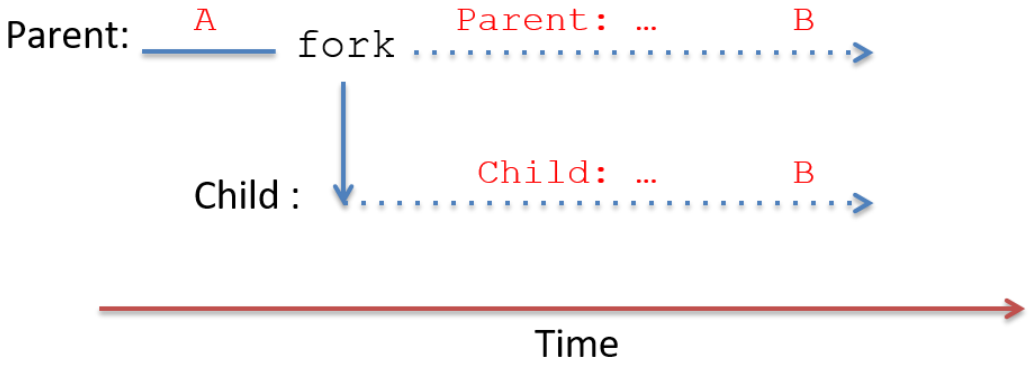
int fork(); Create a new process running the current code

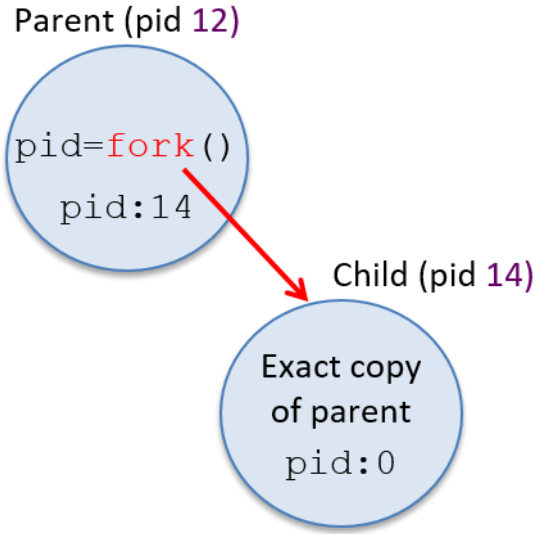
- Creates a new process (new state for address space, PID, registers, hardware)
- Starts with a copy of the same memory of the running process
  - » Process starts with a *copy* of the stack, heap, and code



Fork example

```
pid_t pid;
pid = fork(); /* create a new child process
print("pid = %d\n", pid); /* both parent and child execute this */
```





Waiting for a process to finish

int wait(pid_t pid); Wait for the child process **pid** to end

- This is called only by the parent process
- NULL will wait for all children to finish

Exec – Load new code in a process

```
int execvp(char *filename, char *argv[]);
```

- Loads new code from a binary file (filename) into the running process
- Starts running the main() function in that code
- This function never returns to the calling processes' codebase