Operating Systems

Processes-Part2

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Spring 2023

CPU Switch From Process to Process

process P_0	operating system	process P_1

interrupt or system call

executing

save state into PCB_0

idle

reload state from PCB_1
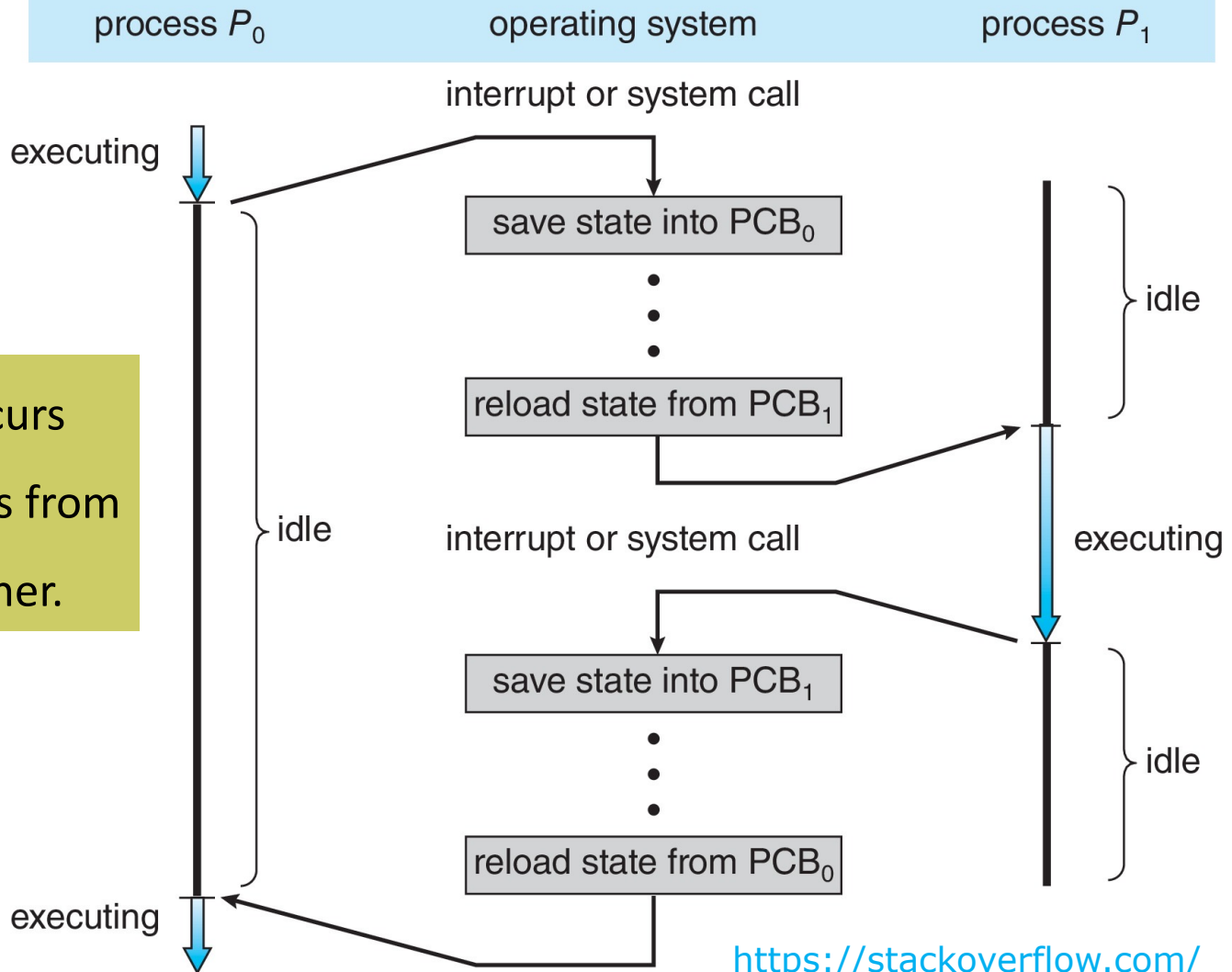
A **context switch** occurs when the CPU switches from one process to another.

idle

interrupt or system call

executing

save state into PCB_1

idle

reload state from PCB_0

executing

Context Switch

- The system must *save the state* of the old process and load the *saved state* for the new process via a *context switch*.

- *Context* of a process represented in the *PCB*.

- Context-switch time is *pure overhead*

  - The system does no useful work while switching.

| The more complex the OS and the PCB | ⮕ | the longer the context switch |

Context Switch (cont.)

- Time dependent on hardware support

<table>
<tr>
<td>Some hardware provides multiple sets of registers per CPU (e.g., the Sun UltraSPARC processor)</td>
<td>→</td>
<td>*multiple* contexts loaded at once</td>
</tr>
</table>

Operations on Processes

- System must provide mechanisms for:

  - Process creation

  - Process termination

Process Creation

fork()
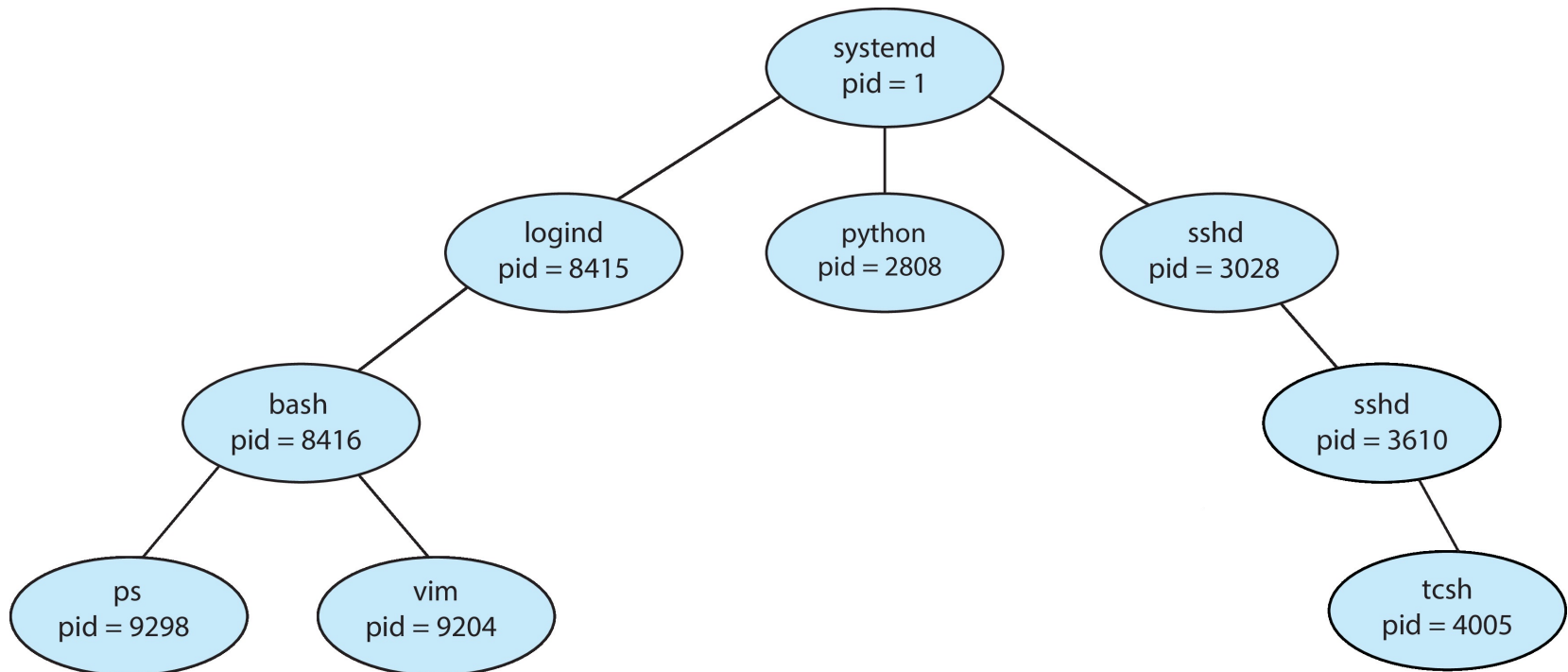
Process Termination

https://dextutor.com/operations-on-process-in-os/

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a *tree* of processes.

- Process identified and managed via a ***process identifier*** (pid).

Let's See It in Practice

- https://www.simplified.guide/linux/process-view-tree

- http://manpages.ubuntu.com/manpages/bionic/man1/pstree.1.html

```
ahmad@ubuntu20:~$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─acpid
        ├─anacron
        ├─avahi-daemon───avahi-daemon
        ├─colord───2*[{colord}]
        ├─cron
        ├─cups-browsed───2*[{cups-browsed}]
        ├─cupsd───dbus
        ├─dbus-daemon
        ├─dnsmasq───dnsmasq
        ├─gdm3─┬─gdm-session-wor─┬─gdm-x-session─┬─Xorg───9*[{Xorg}]
        │      │                 │               ├─gnome-session-b─┬─ssh-agent
        │      │                 │               │                 └─2*[{gnome-session-b}]
        │      │                 │               └─2*[{gdm-x-session}]
        │      │                 └─2*[{gdm-session-wor}]
        │      └─2*[{gdm3}]
```

Amirkabir University of Technology
(Tehran Polytechnic)

Process Creation-Resource Sharing Options

- Parent and children share all resources

Parent process	**Parent's resources** **(CPU time, memory, files, I/O devices)**	Child process

- Children share subset of parent's resources

Parent process	**Parent's resources** **(CPU time, memory, files, I/O devices)**	Child process

- Parent and child share no resources

Parent process	**Parent's resources** **(CPU time, memory, files, I/O devices)**	Child process
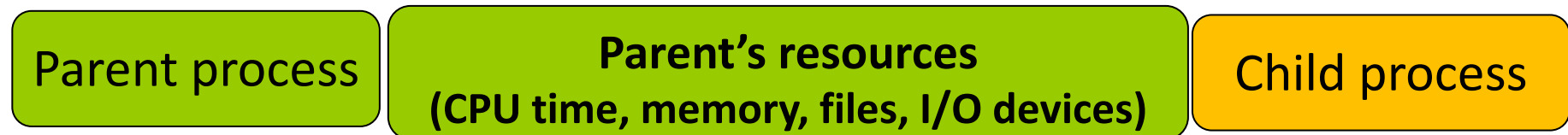
Process Creation-Execution Options

- Parent and children execute *concurrently*
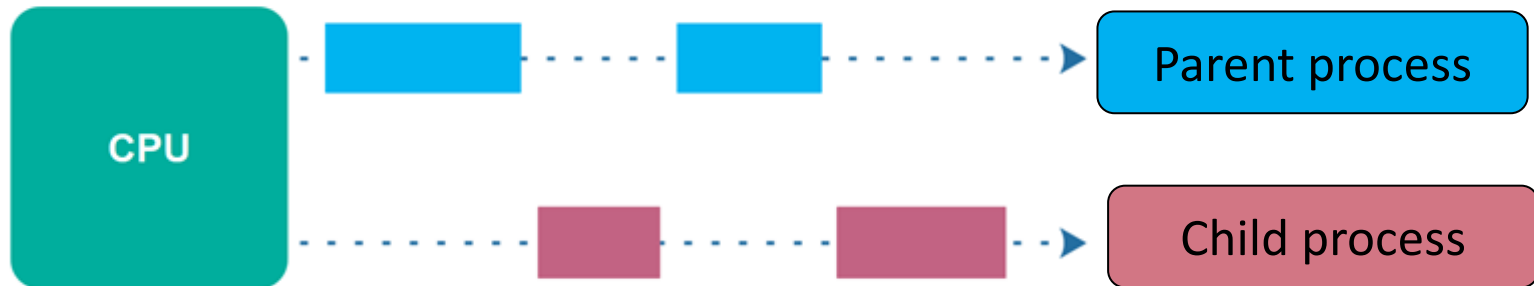
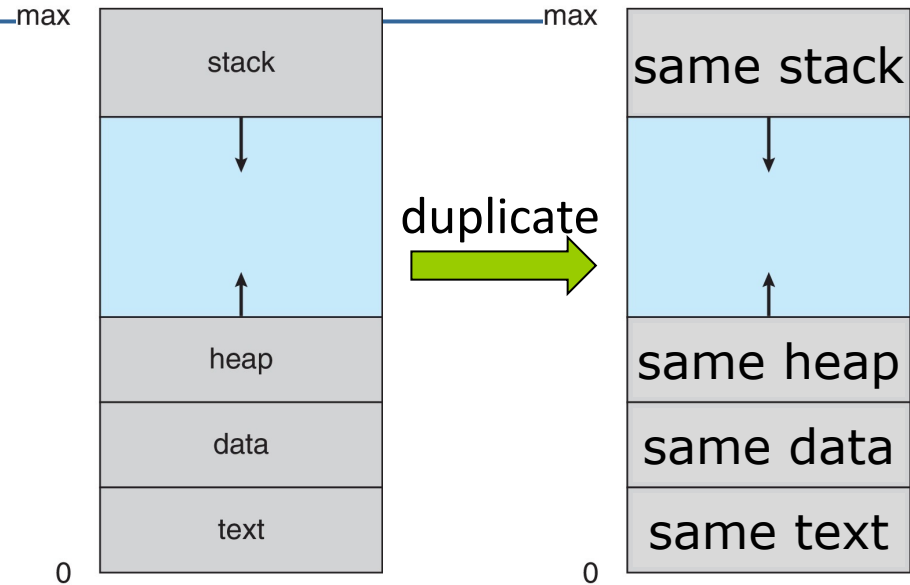

- Parent *waits* until children terminate
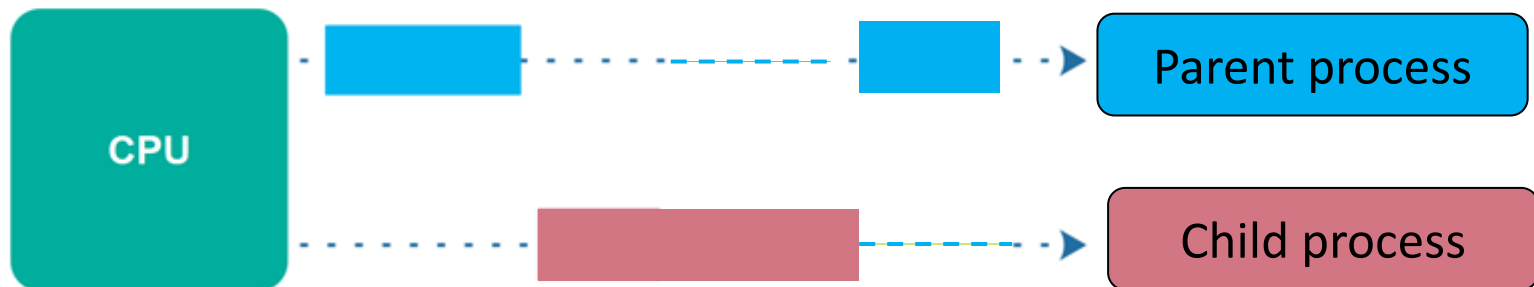
Process Creation-Address Space
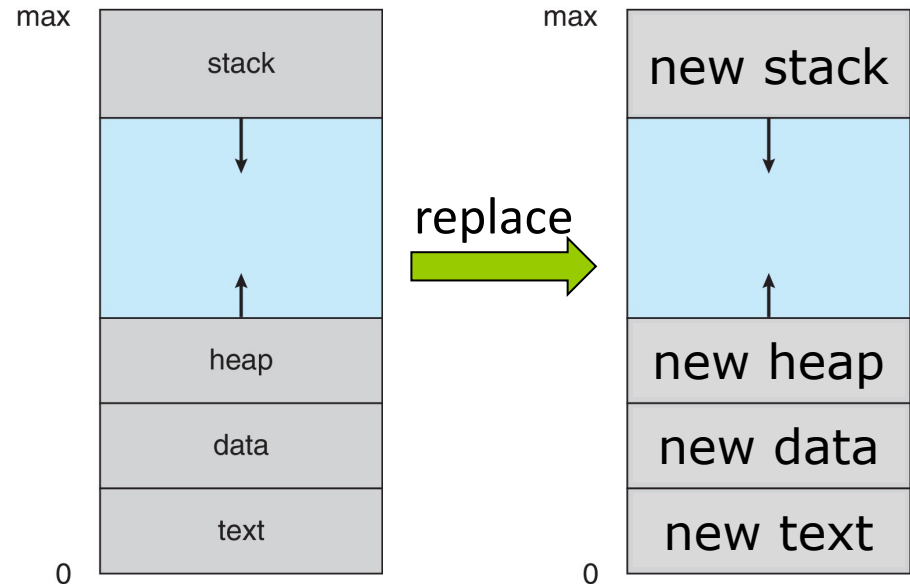
- Child **duplicate** of parent

- Child has a program loaded into it

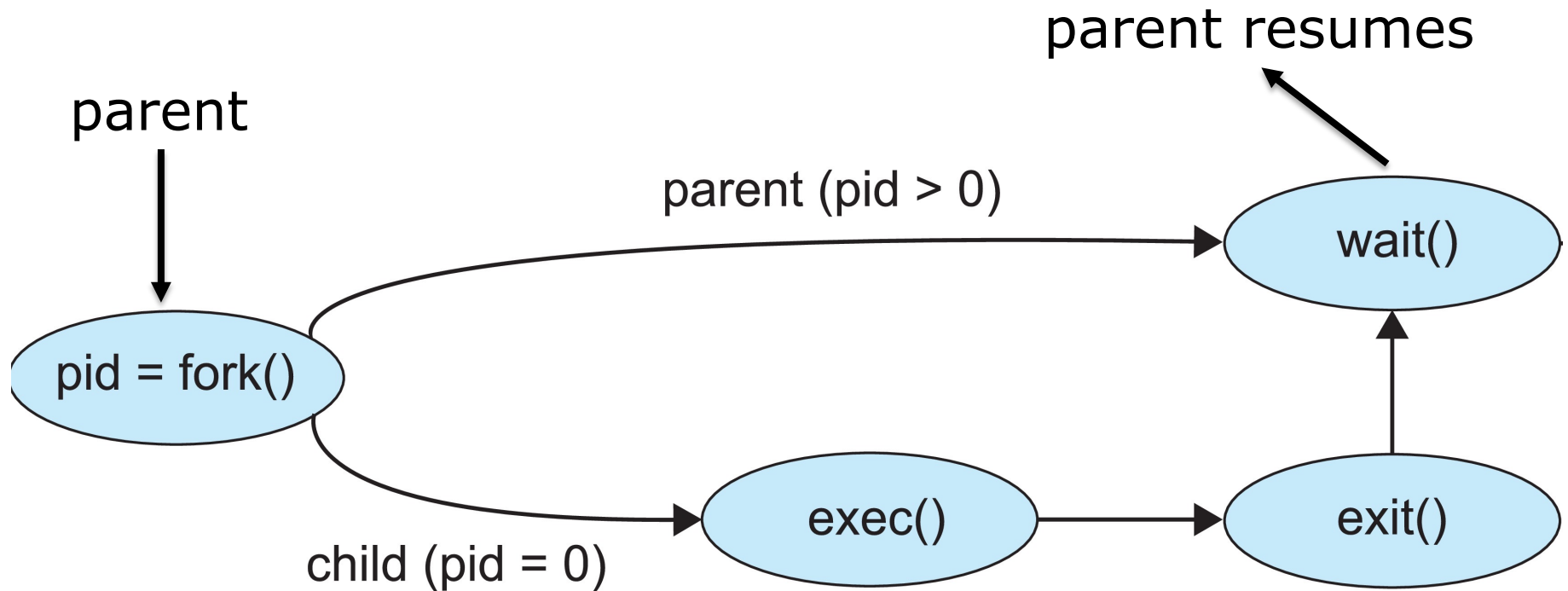Amirkabir University of Technology
(Tehran Polytechnic)

Process Creation-UNIX examples

- **fork()** system call creates new process.

- **exec()** system call used after a **fork()** to *replace* the process' memory space with a new program.

- Parent process calls **wait()** waiting for the child to terminate.

Process Creation (Cont.)

C Program Forking Separate Process

#include <sys/types.h> <stdio.h> <***unistd***.h>

```c
int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```