P1: Intro to Processes (ABC 12.1)

Ion Mandoiu
Laurent Michel
Revised by M. Khan and J. Shi

Process Basics

- A **process** is an instance of a program being executed
  - Core operating system (OS) concept
- In a **multiprocessing** OS
  - Multiple programs can be executed at the same time
  - Multiple instances of a program can be executed at the same time
- Executing multiple programs
  - Single-core: time-sharing
  - Multi-core: true parallelism + time-sharing

Process Management: OS View

- OS maintains a process table
  - Each process has a table entry, called process control block (PCB)
  - Typical PCB info

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

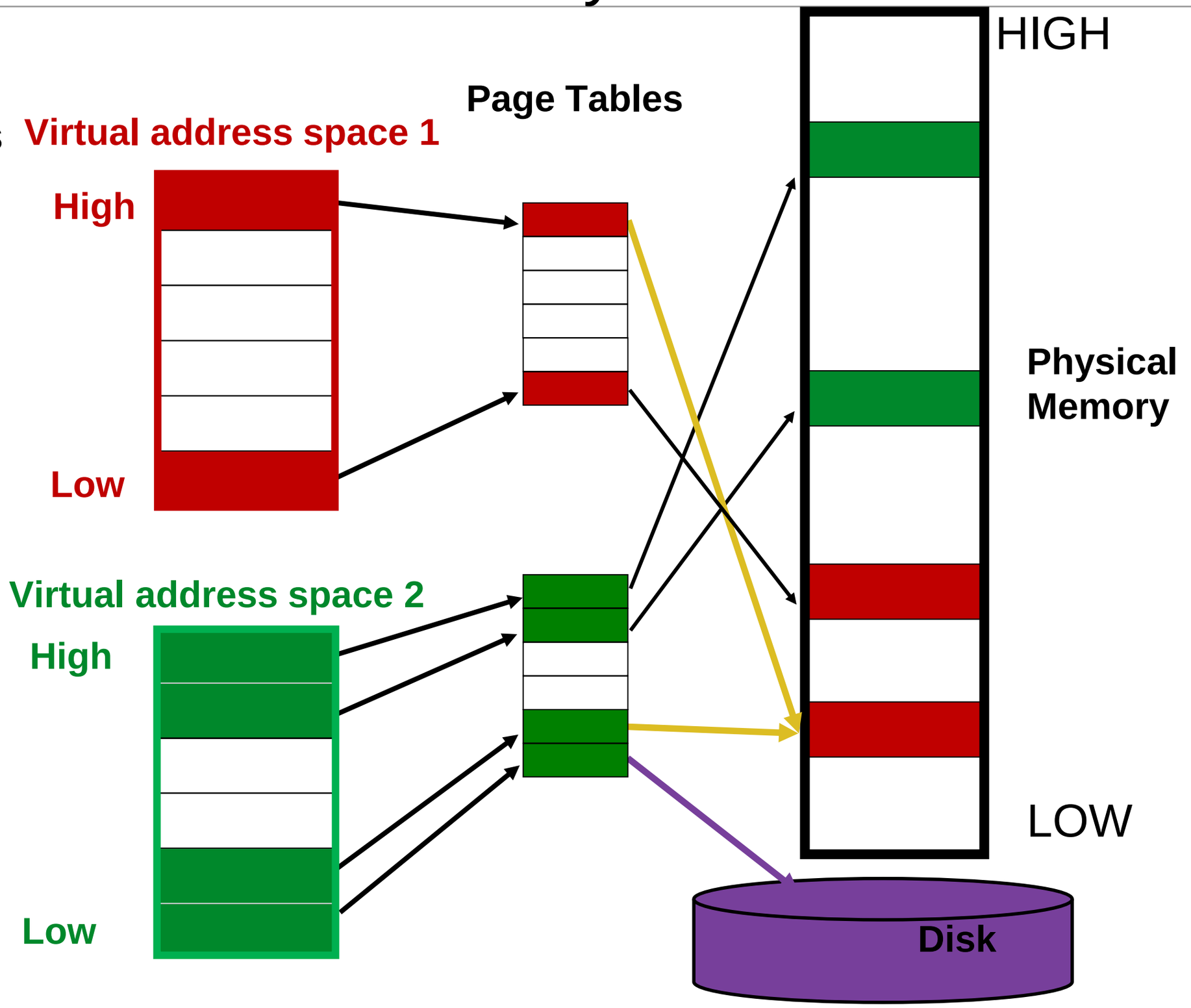
- OS **scheduler** picks processes to be executed at any given time
  - When a process is suspended, its state is saved in PCB
  - What about the process memory?

Paged Virtual Memory: How Processes Share Memory

Physical memory is shared by all processes

Page table maps virtual address to physical address

Multiple virtual pages can be mapped to the same physical pages



Process Management: User's View

- Events which cause process creation
  - System initialization
  - User request to create a new process (e.g., **shell command**)
  - Executing a **shell script**, which may create many processes
- Events which cause process termination
  - Normal program exit
  - Error exit
  - Fatal error, e.g., segmentation fault
  - Killed by user command or signal (Ctrl-C)

Useful Commands

- ps
  - List running processes
- pstree
  - Display the **tree** of processes
- top
  - Dynamic view of memory & CPU usage + processes that use most resources (to exit top, press q)
- kill
  - Kill a process given its **process ID**
  - Try -9 option if simple kill does not work

Additional functions/options in man page of each command

Process Management: Programmer's View

- Process birth
  - Processes are created by other processes!
  - A process always starts as a **clone** of its parent process
  - Then the process may **upgrade itself** to run a different executable
    - Child process **retains access** to the files open in the parent
- Process life
  - Child process can create its own children processes
- Process death
  - Eventually calls **exit** or **abort** to commit "suicide"
  - Or gets killed

Birth via Cloning

- The function to create a new process in your code

```
#include <unistd.h>
pid_t  fork(void);
```

- Child is an exact copy of the parent
  - Both return from fork()
- **Only difference is the returned value**
  - In the **parent** process:
    - fork() returns the process identifier of the child (> 0)
    - If a failure occurred, it returns -1 (and sets errno)
  - In the **child** process: fork() returns 0 (zero)

Concurrency

- Parent and child processes return from fork() concurrently
  - They may return at the same time (on a multicore machine) or one after the other
  - Cannot assume that they return at the same time or which one "returns first" (even on a uni-core)
  - Order is chosen by OS scheduler

Cloning effect

- On memory
  - The parent and child memory 100% identical
  - But are viewed as distinct by OS (“copy-on-write”)
  - Any memory change (stack/heap) affects only that copy
  - Thus the parent and child can quickly diverge
- On files
  - All files open in the parent are accessible in the child!
  - I/O operations in either one move the file position indicator

In particular

- stdin, stdout, and stderr of the parent are accessible in the child

What can the parent do ?

- Depends on application!
  - It could wait until the child is done (dies!)
    - Typical of a shell like bash/ksh/zsh/csh/….
  - It could run concurrently and check back on the child later
  - It could run concurrently and ignore the child
    - If child dies it enters a **zombie** state

Waiting on a child

```
#include <sys/wait.h>

pid_t   wait(int * status);
pid_t   waitpid(pid_t pid, int * status, int options);
```

- Purpose
  - Block the calling process until a child is terminated
    - Or other state changes specified by options
  - Report status in *status (which is ignored if NULL is passed)
    - The cause of death
    - The exit status of the child (what he returned from main)
  - Return value identifies the child process (or -1 on error)
- Run “**man -S2 wait”** for full details

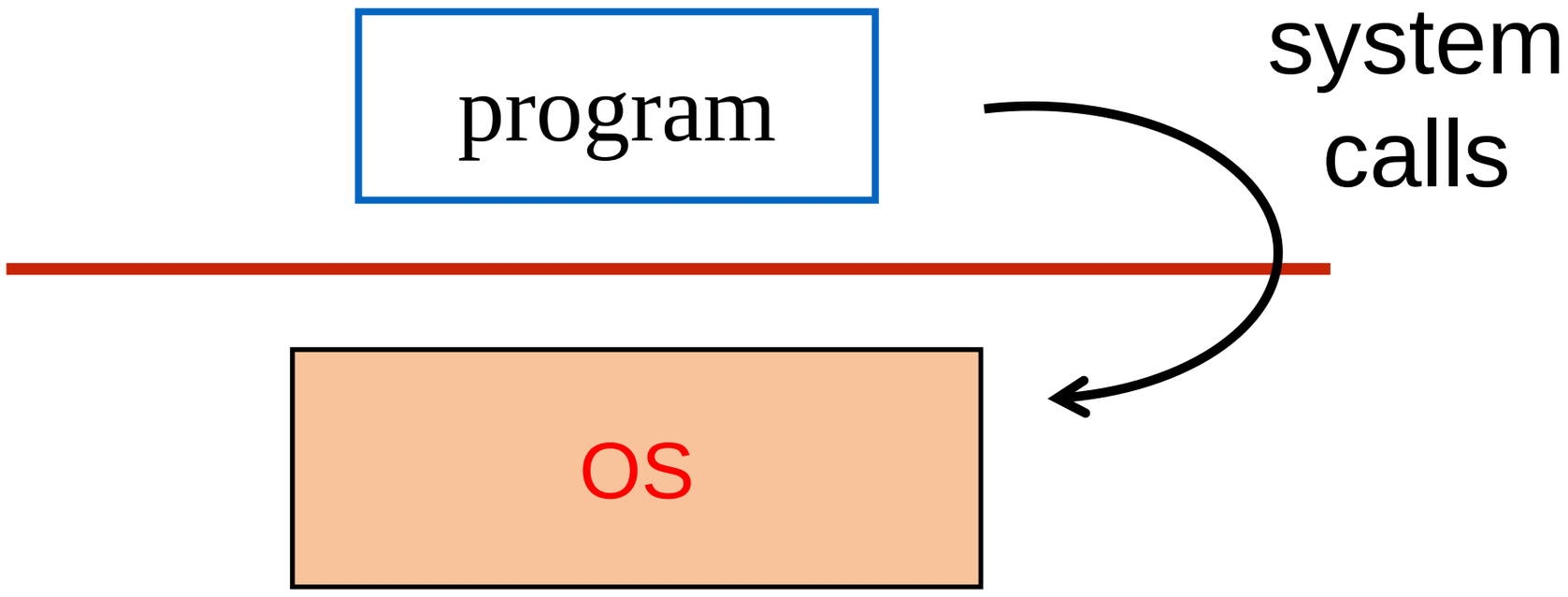
Zombies

- A dead process, waiting to be 'reaped' (checked by its parent)
  - You cannot kill it, because it is already dead
  - Most resources released, but still uses an entry in the process table
- Parents should check their kids
  - On some systems, parents can say they do not want to check
- When a parent dies, ‘init’ becomes the new parent
  - Then the zombie child is reaped

System calls

- APIs used to request services from the OS kernel
  - Example: fork()
  - System calls are more expensive than normal function calls
  - Manuals for system calls are in section 2

```
man -S2 intro ;  man -S2 syscalls
```



Summary

- Clone a process with fork()
  - The child is exactly the same as the parent
  - Check the return value
- Parents wait for child processes
  - Reap the zombies!

demo/fork

demo/forkfib