# Process
# (Chap7-Chap9)

Hongik University

Eunsung Jung

*Disclaimer: The slides are borrowed from many sources!*

# Contents

- Process States and Transitions
- Process table and u area
- Context of a process
  - User-level context
  - Register context
  - System-level context

# process

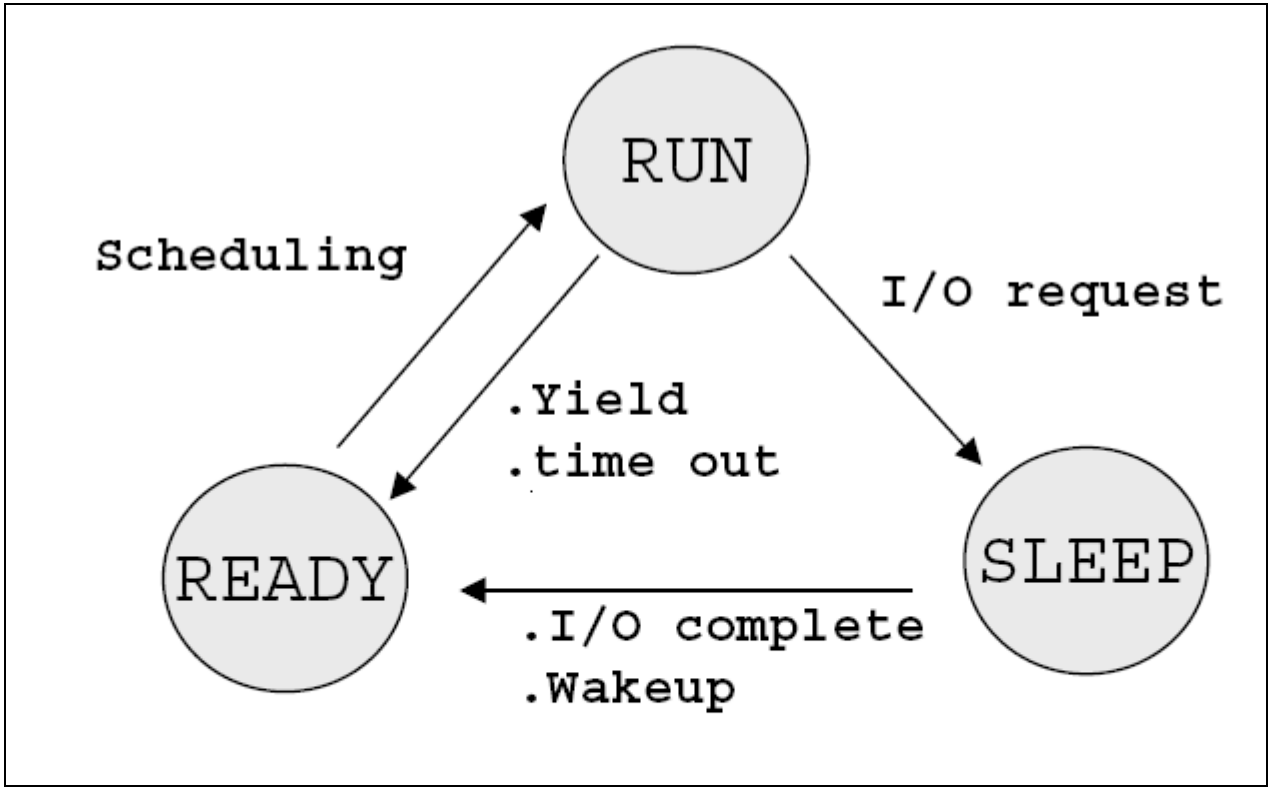
- an instance of running program

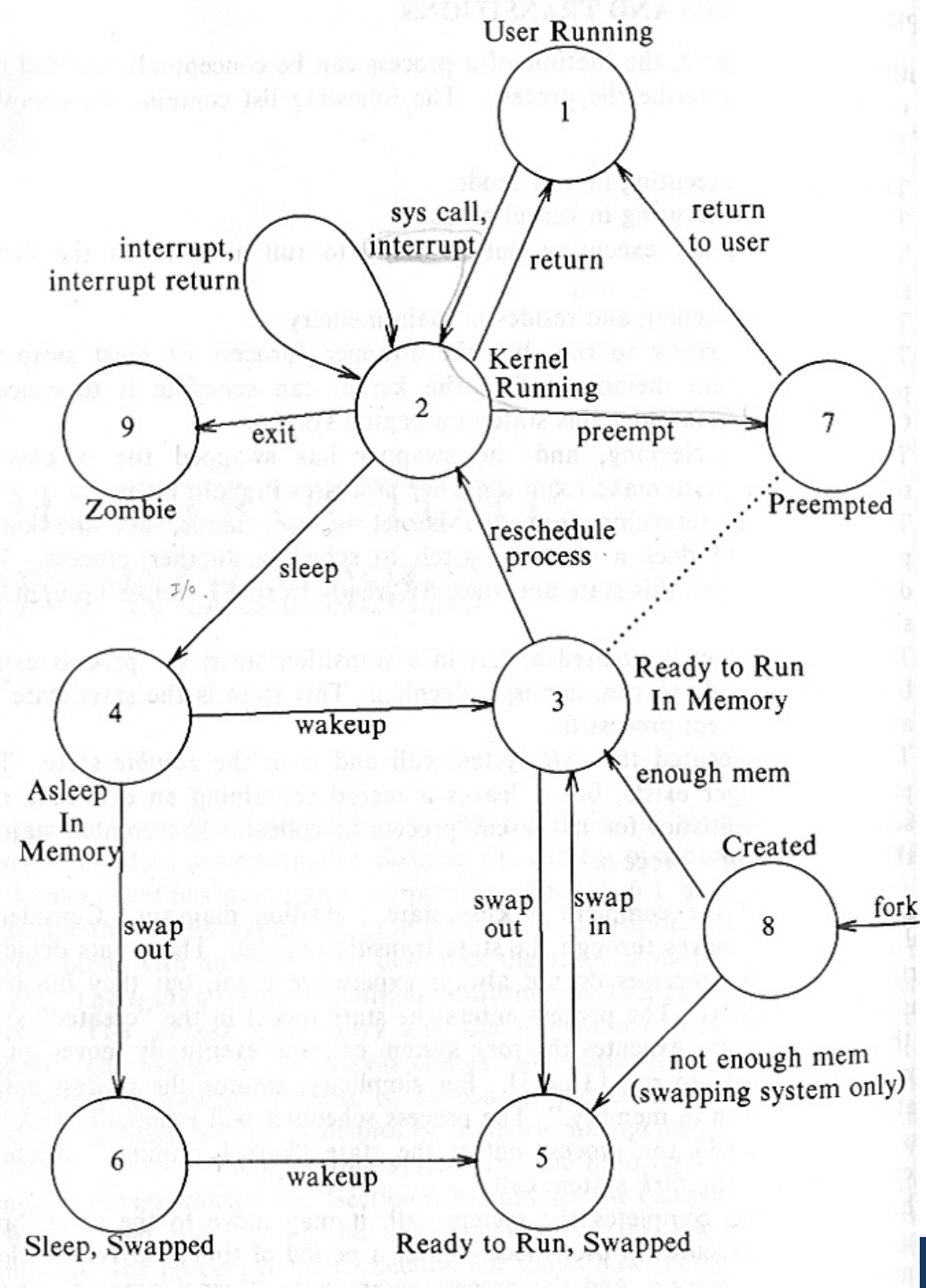
- Program vs process
  - Program : just a passive collection of instructions
  - Process : actual execution of the instructions
  - Several processes may be associated with one program

- In addition to program code, necessary resources (memory, CPU, etc) are allocated to process

# Process States and Transitions

- Lifetime of a process






# Process State Transition Diagram in UNIX

# CPU execution mode

- place restrictions on the operations that can be performed by the process currently running in the CPU
- **Kernel mode**
  - When the CPU is in kernel mode, it is assumed to be executing *trusted* software, and thus it can execute any instructions and reference any memory addresses (i.e., locations in memory).
  - The kernel (which is the core of the operating system and has complete control over everything that occurs in the system) is *trusted* software, but all other programs are considered *untrusted* software.
- **User mode**
  - It is a *non-privileged* mode in which each process (i.e., a running instance of a program) starts out. It is non-privileged in that it is forbidden for processes in this mode to access those portions of memory (i.e., RAM) that have been allocated to the kernel or to other programs.

# PCB (Process Control Block)

- Contain process-related information
  - (e.g. which resources allocated to a process
  - in what state.

- Largely three parts
  - proc table
  - u area
  - Text table
    - For code sharing (e.g. vi , shell , …)

# Process (proc) table and u area

- Kernel data structures
- Describes the state of a process
- Process table
  - Always accessible to the kernel.
- U area (user area)
  - Part of the process space
  - Mapped and visible only when the process is running.
  - Generally, much bigger than proc table

- Swappable vs non-swappable

# Fields in proc table

- Process ID
- Process state
- Pointers to process (code) and its u area
- User ID
- Scheduling parameters (priority, CPU utilization,…)
- Signal field
  - Signals not yet handled (pending delivery)
- Various timers
  - Execution time, resource utilization, scheduling priority

# Fields in u area

- Pointer to process table entry
- Real and effective user IDs
- Timers – time the process spent executing
- An array for the process to react to signals
- Control terminal – if one exists
- Error field, return value – system call
- I/O parameters
  - (file offsets for I/O, data amount to transfer, …)
- Current directory, current root
- User file descriptor table
- Limits – process, file size
- Permission – used on creating the process

# Layout of System Memory

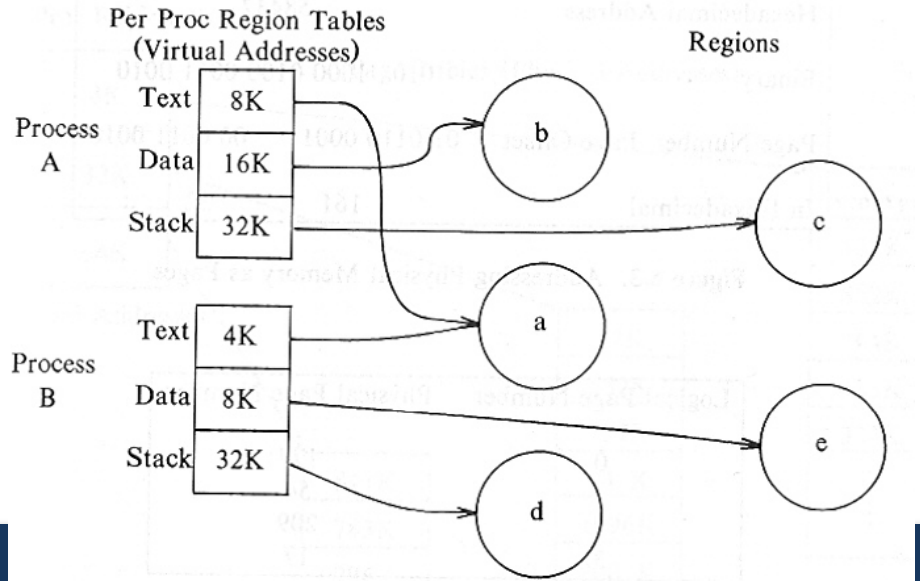
- Physical address space
  - impossible for two processes to execute concurrently if their set of generated addresses overlapped.
- Virtual address space
  - Allows many processes to share finite amount of physical memory
  - Each process uses the same virtual addresses but reference different physical addresses
  - Requires mechanism for translating  virtual address to physical address

# Regions

- Region (similar to segment)
  - Contiguous area of virtual address space of a process that can be treated as a distinct object to be shared or protected.
- Virtual address space of a process is divided into logical regions
  - Text : a set of instructions
  - Data : (initialized & uninitialized) data variables
  - Stack : data structures local to a subroutine

# Pregion (per process region table)

- Each pregion entry points to starting virtual address of region in the process.
- Can exist in proc table or u area
- Shared region may have different virtual address in each process



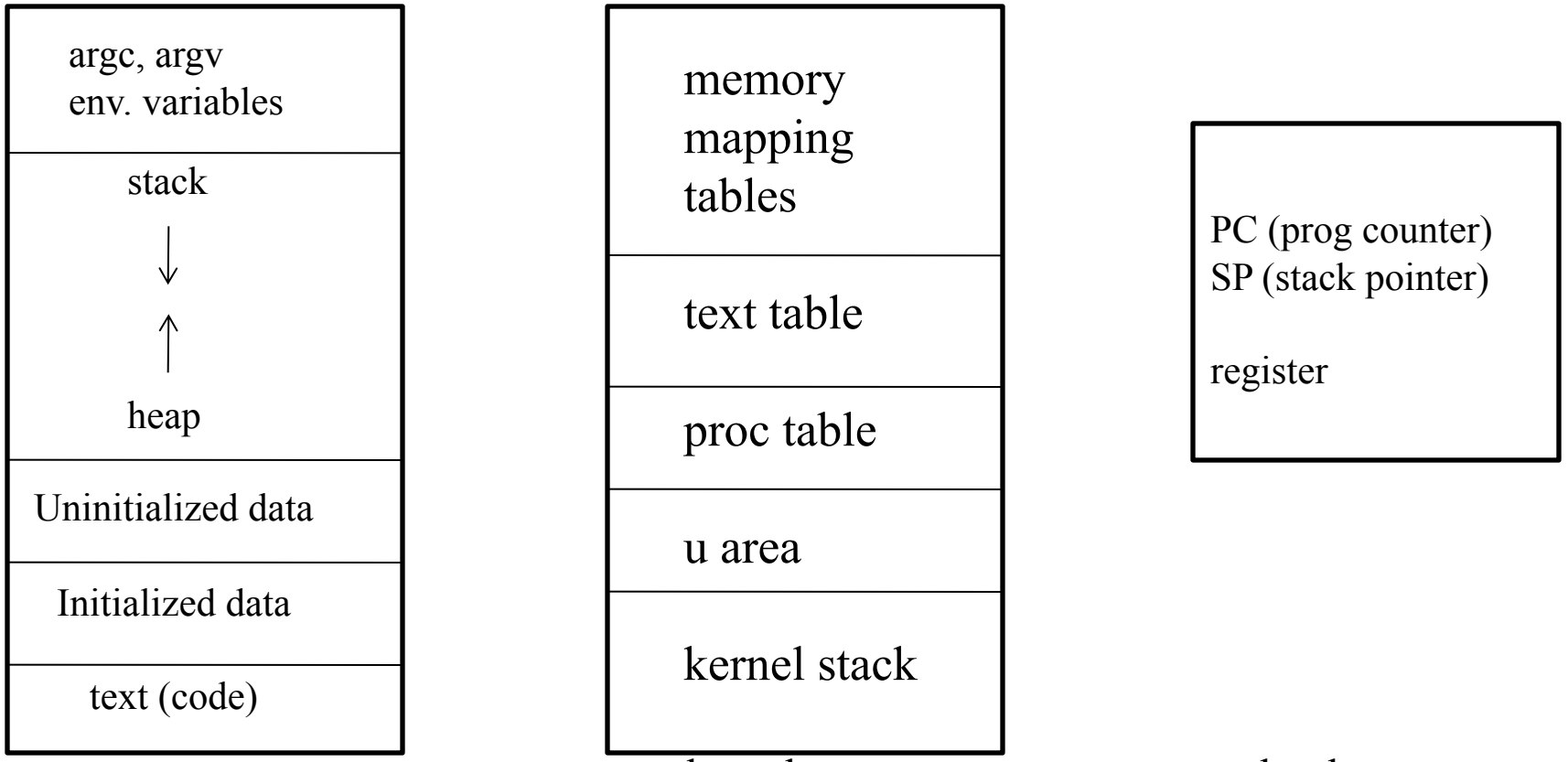
# Process context

- Each time a process is removed from access to CPU, sufficient information on its current **operating state** must be stored such that when it is again scheduled to run on the processor it can resume its operation from an identical position.
- This operational state data is known as its *context*
- Context switch
  - the act of replacing a process to another for execution

# Context of a process

- consists of its (user) address space, hardware registers and kernel data structures that relate to the process
  - User-level context
  - Register context
  - System-level context

# Process Context




<user space>
User level context

<kernel space>
System-level context

<hardware>
Register context

# User-level context

- Process text, data, user stack, and shared memory
- Parts of the virtual address space of a process periodically do not reside in main memory because of swapping.

# Register context

- Program counter (PC)
- process status (PS) register (e.g. overflowed?)
- stack pointer (SP)
- general-purpose registers

# System-level context

- PTE (proc table entry)
- u area
- Region table
- Kernel stack
- system-level context layer
  - Contains necessary information to recover the previous layer

# Components of the process context