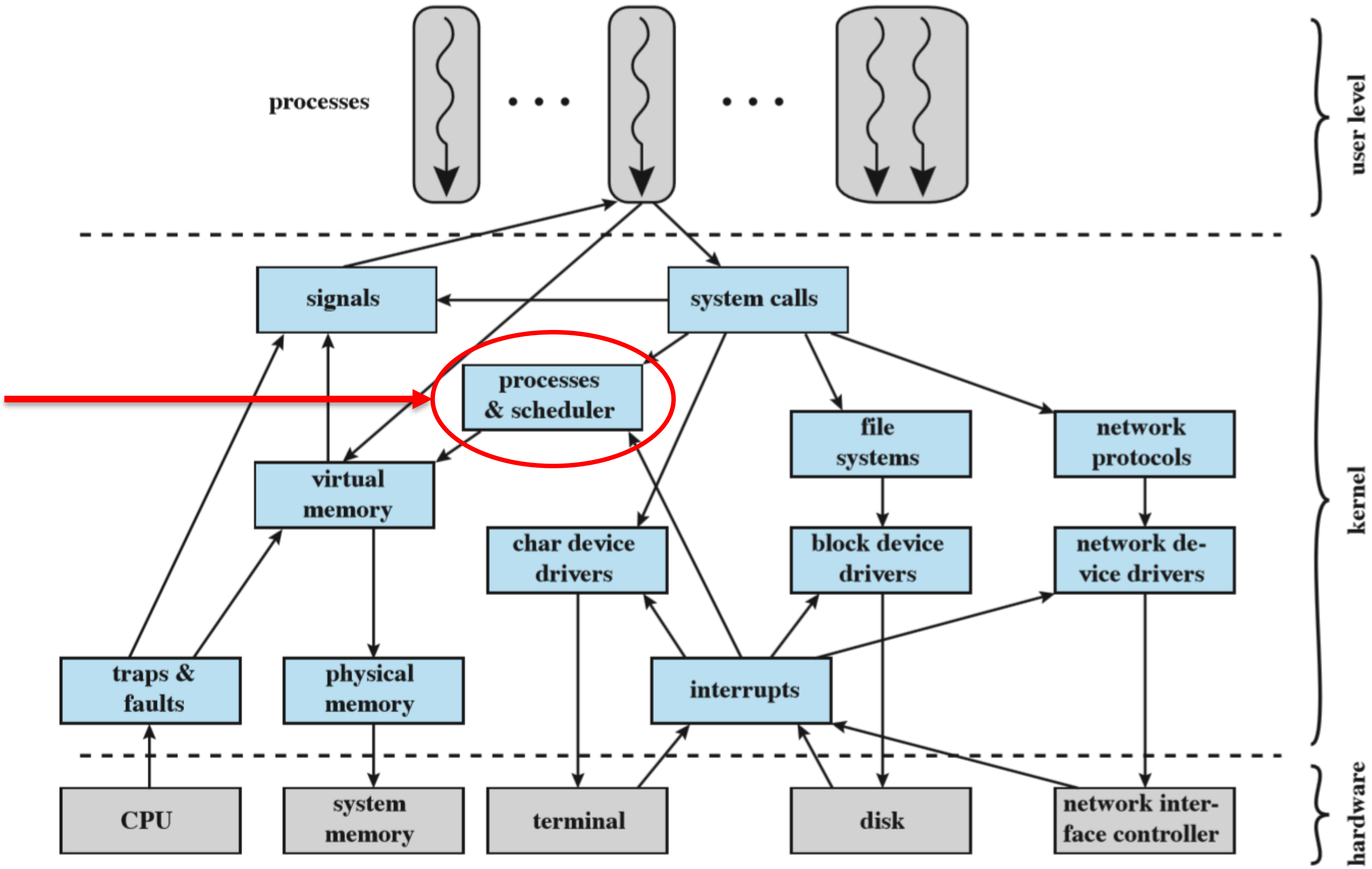
CS3841 – Design of Operating Systems Processes

- Objectives
  - Explain the contents of the text section, data section, heap, and stack of a program
  - Draw a graphical representation of a process in memory
  - Explain the concept of process state
  - Draw a state transition diagram for process states
  - List the contents of a process control block
  - Explain what the process scheduler is responsible for doing within the operating system.
  - Be able to obtain information about the processes which are running under Linux.
  - Explain the relationship between process ids, groups, and the general process hierarchy in Unix

Dual Mode Operation

- Modern Operating Systems use at least two modes of operation
  - User mode
    - A restricted mode of operation which only allows certain instructions to be executed by the program
    - Prevents errant processes from crashing the system
  - Kernel Mode
    - Also referred to as supervisor mode, system mode, or privileged mode
    - Allows the system full access to the microprocessor
    - Intended to be used only by the operating system

Linux Kernel



Operating Systems: Internals and Design Principles, 9th Edition William Stallings

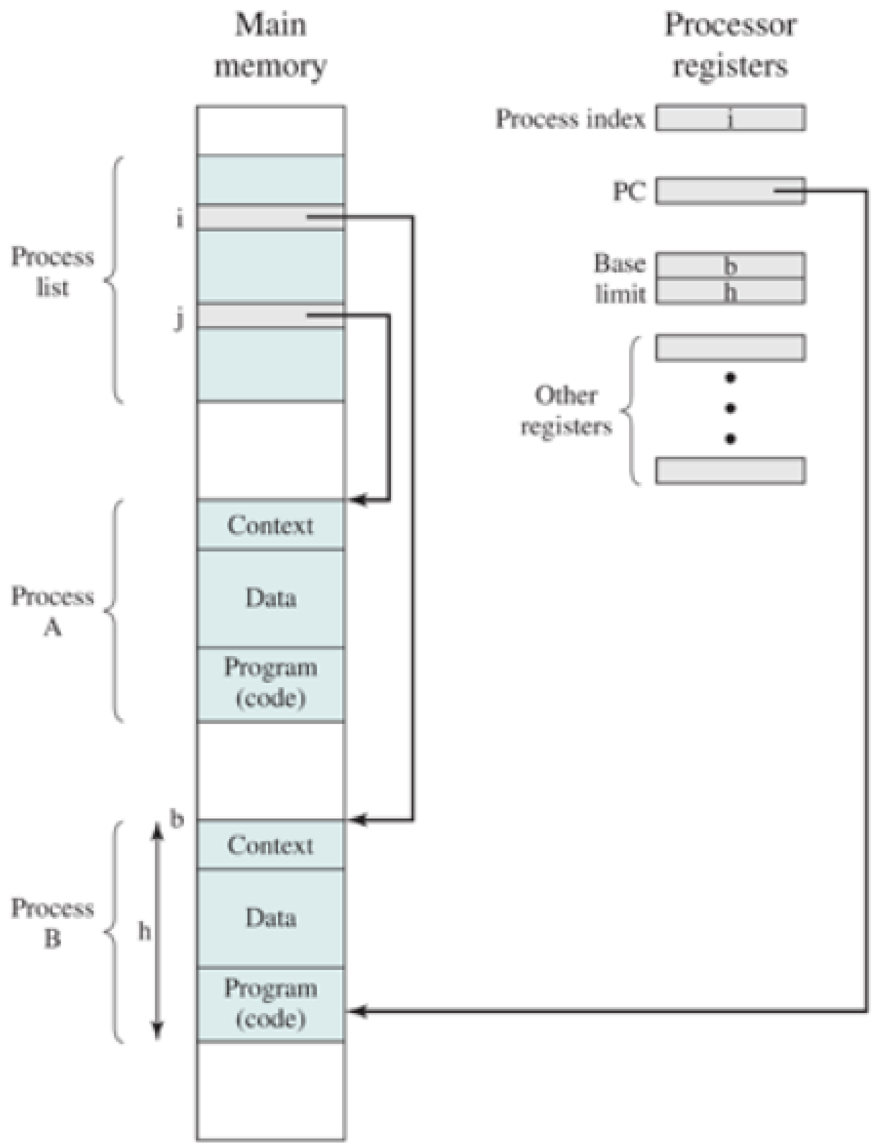


Figure 2.8 Typical Process Implementation

Program vs Process

- Program
  - Static representation of operations and data
  - Compiled code
- Process
  - Instance of active exeuction

Why do we need processes?

- Concurrent Processing
- Real concurrency achieved by hardware
  - I/O devices operate at same time as processor
  - Multiple processors/cores each operate at the same time
- Apparent concurrency achieved with multitasking (multiprogramming)
  - Multiple programs appear to operate simultaneously
  - Operating system provides the illusion
- Isolation and Protection
  - Can't let one process affect another without permission

Program Structure

- A program has multiple pieces – Here are some examples
  - Text
    - The instructions to execute
  - Data sections
    - Static data (numbers, strings, etc.)
  - Linking information
    - What software libraries does this program use? (math library, crypto library, etc.)
  - Symbol Table
    - Information about the symbols (variable names) this program uses

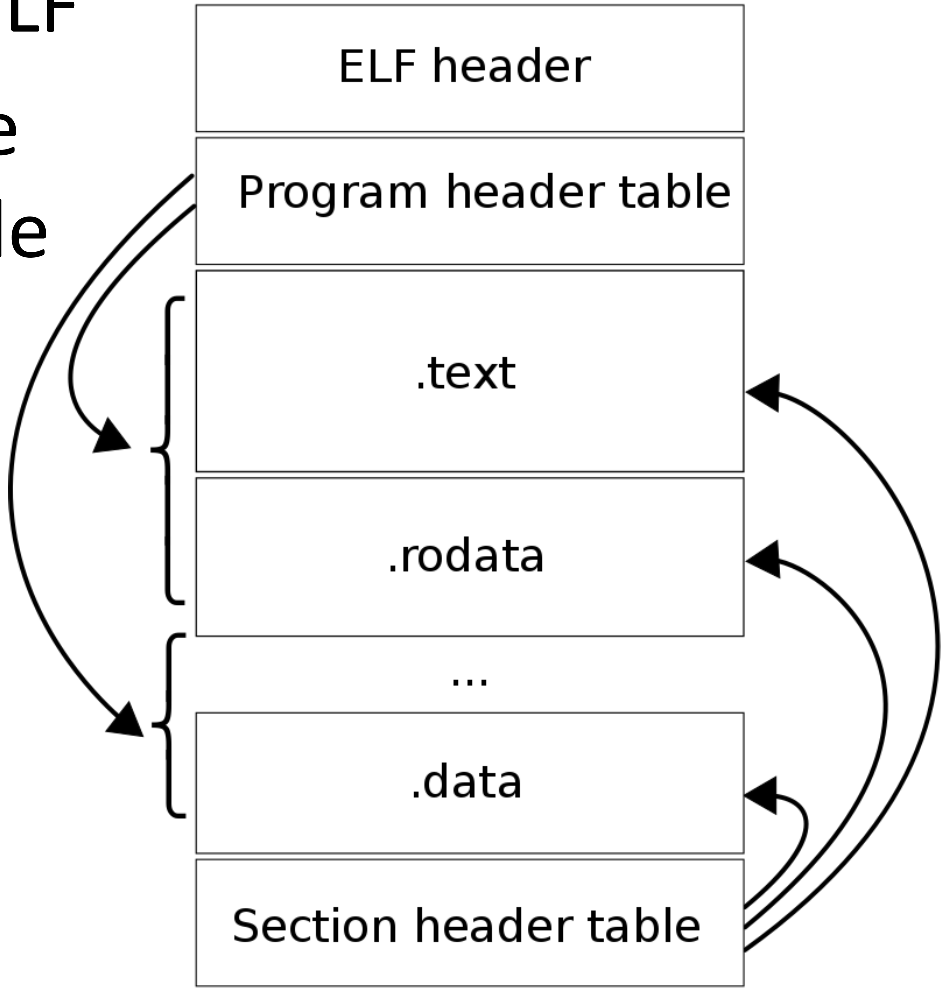
Process Structure

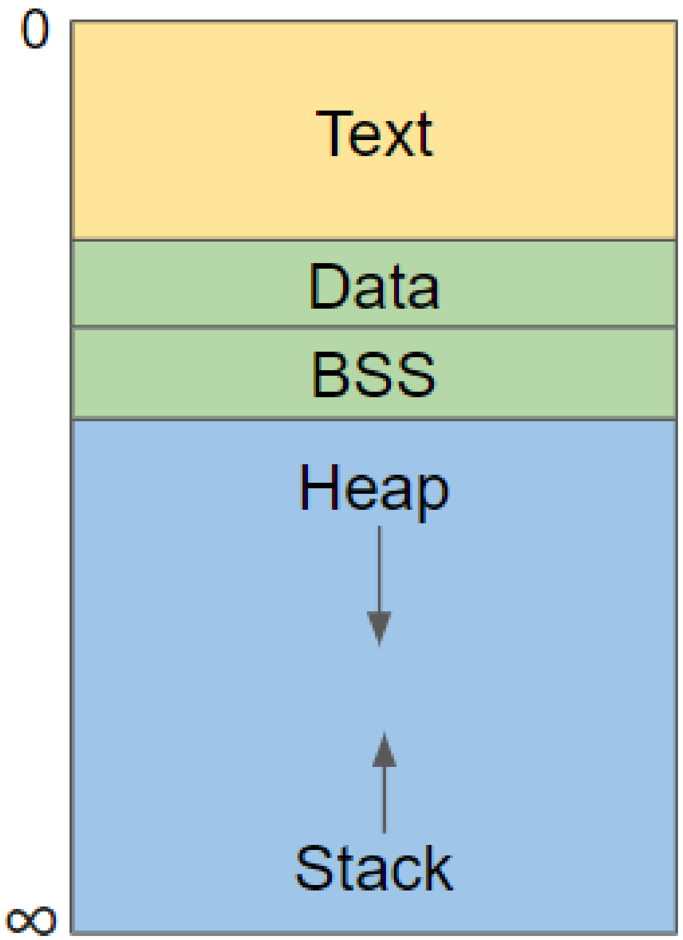
- A process has multiple pieces
  - Text section
    - The executable code that is running
  - Data section
    - The global variables of the program
    - BSS (Block Started by Symbol) – Uninitialized global variables
  - Heap
    - Dynamically allocated memory when a process executes (i.e. new)
  - Stack
    - Temporary data for the process
      - Function parameters, return addresses, local variables, etc.

Program vs Process

Program – ELF

Executable and Linkable Format





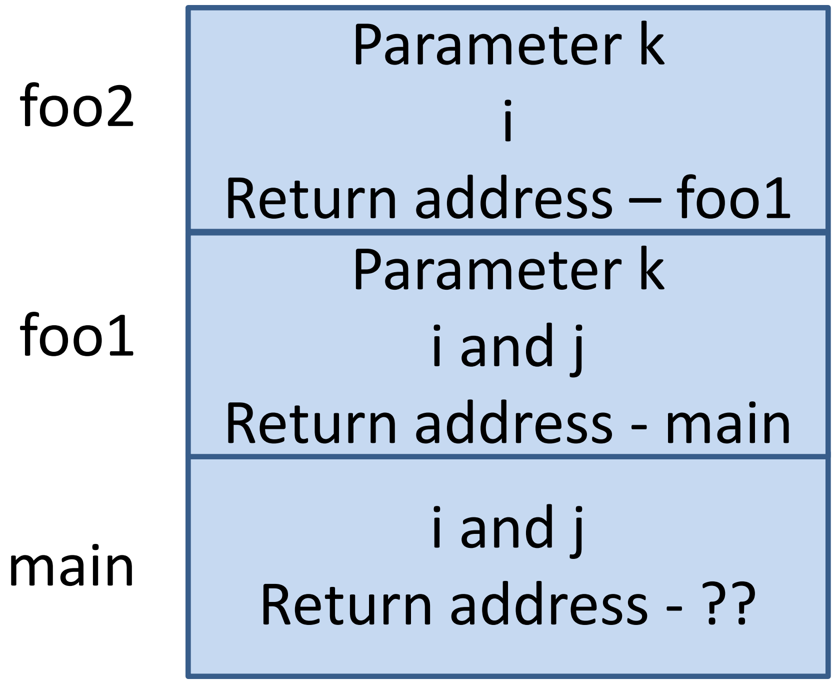
Process

Stack

```
int foo2(int k) {
        int i = 5
        return i + k;
}

int foo1(int k) {
        int i = 5;
        int j = k + i + foo2(k);
        return j;
}

int main() {
        int i = foo1(20);
        int j = i + 10;
        return j;
}
```

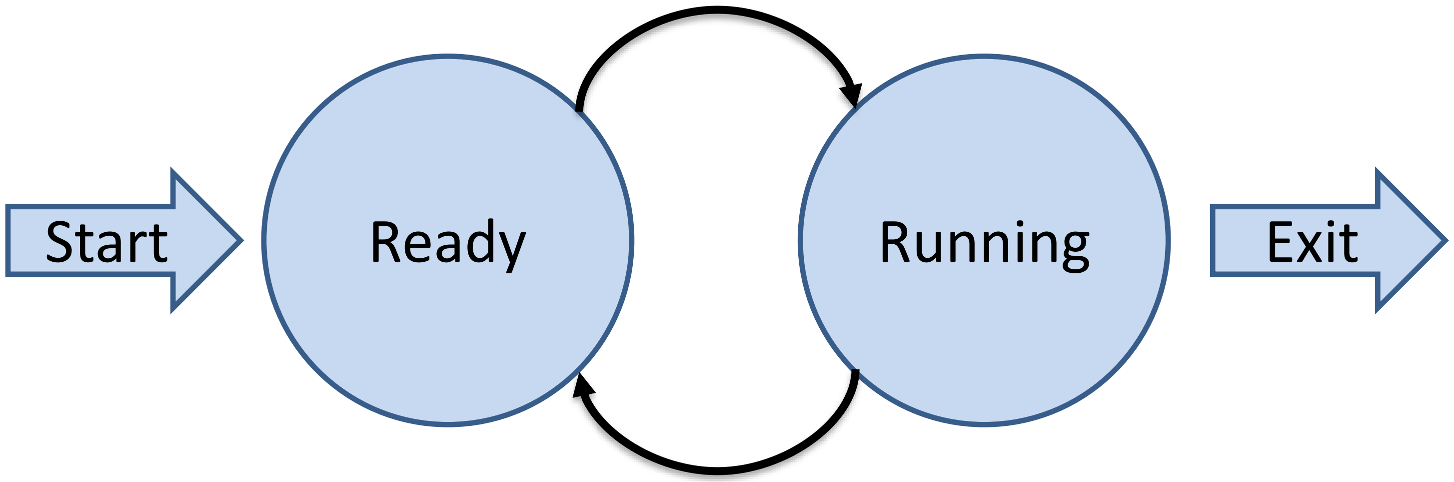


Processes

- OS abstraction
- Created by OS system call
- Managed entirely by OS; unknown to hardware
- Operates “concurrently” with other processes
- Processes have “state”

Process State

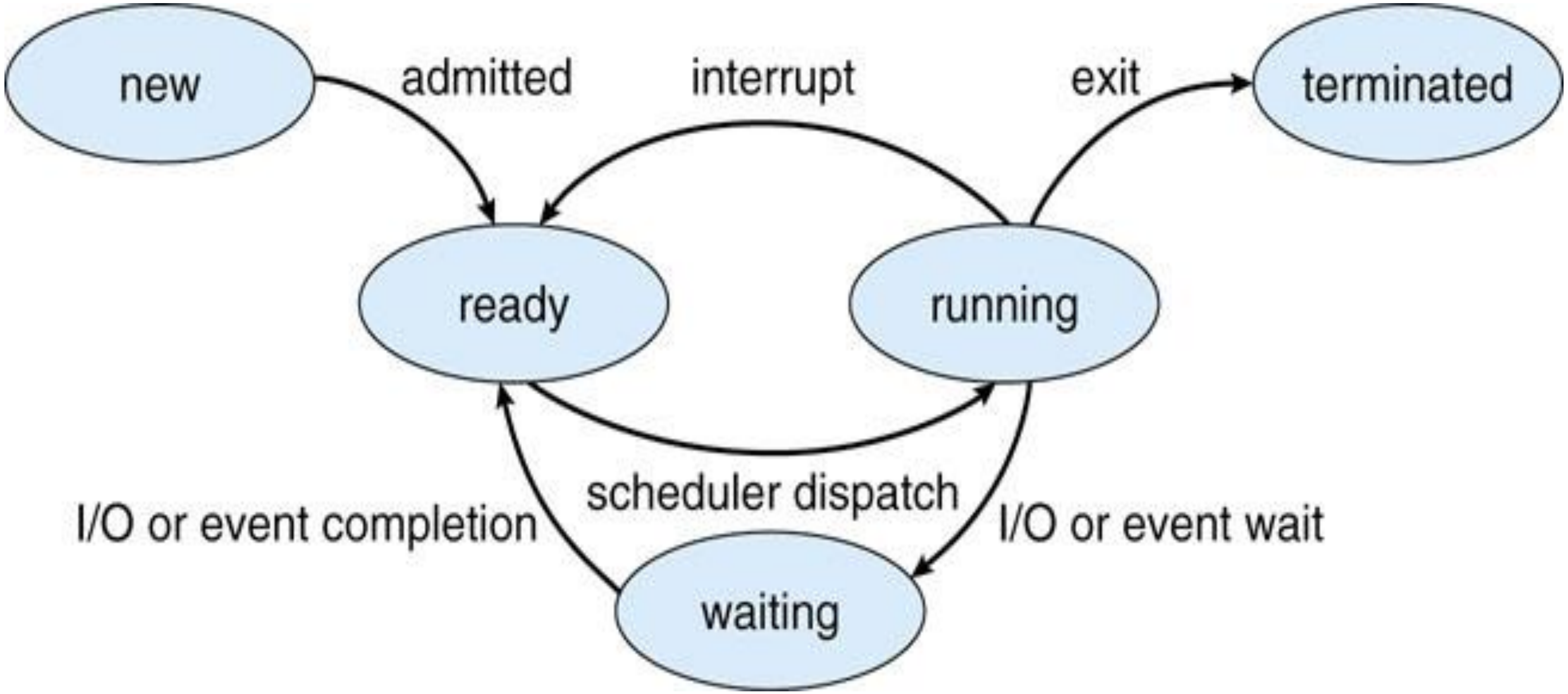
- Two state model
  - Running and Ready
- Is this all we need?
- What about I/O?
- How do we decide state transitions?
- Round robin scheduling:
  - Each process in the queue is given a certain amount of time to execute and then returned to the queue, unless it completes
  - Period is known as a quantum
- Efficiency - Can we do better?



Process State – 5 State Model

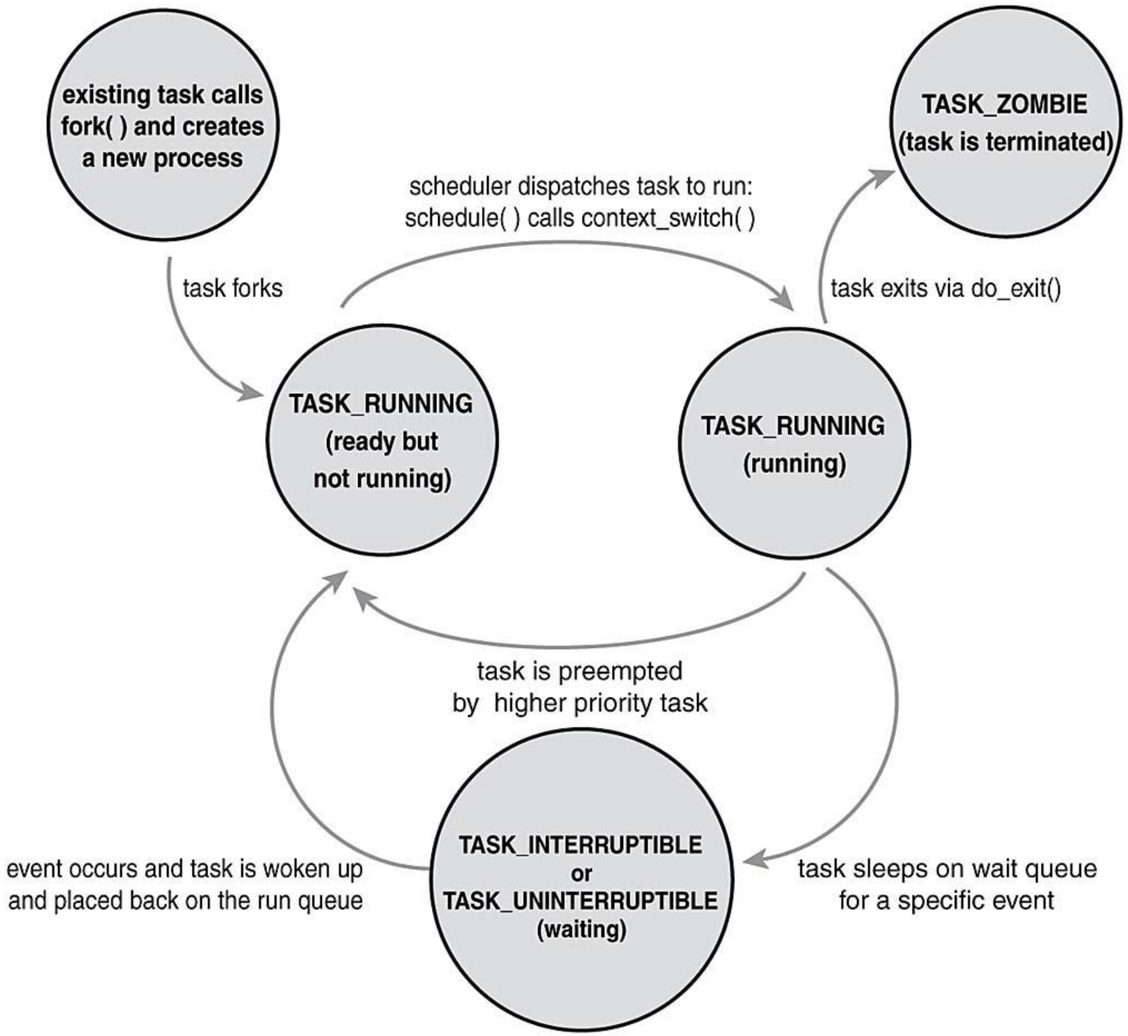
- New
  - The process has just been created but has not yet executed
- Ready
  - The process is waiting to be assigned to a CPU
- Waiting (Blocked)
  - The process is waiting for some event to occur
- Running
  - The process is executing on the CPU
- Terminated (Exit)
  - The process has finished execution

Process State – 5 State Model



- Is this all we need?
- Efficiency - Can we do better?

Process State – Linux Model



Process – More things to think about

- How many processes do we want to allow?
- What if we run out of memory?
- What about process priority?
- How do we handle run-away processes?
- How do we schedule processes? Fairness?

Process Scheduler

- Objective of multiprogramming
  - The CPU must always be doing something
- Process scheduler
  - Enforces scheduling policy
  - Selects an available process which is ready and determines that it will be the next process to execute
  - Send the process to the dispatcher
- Process dispatchers
  - Responsible for causing the CPU to start executing the desired process

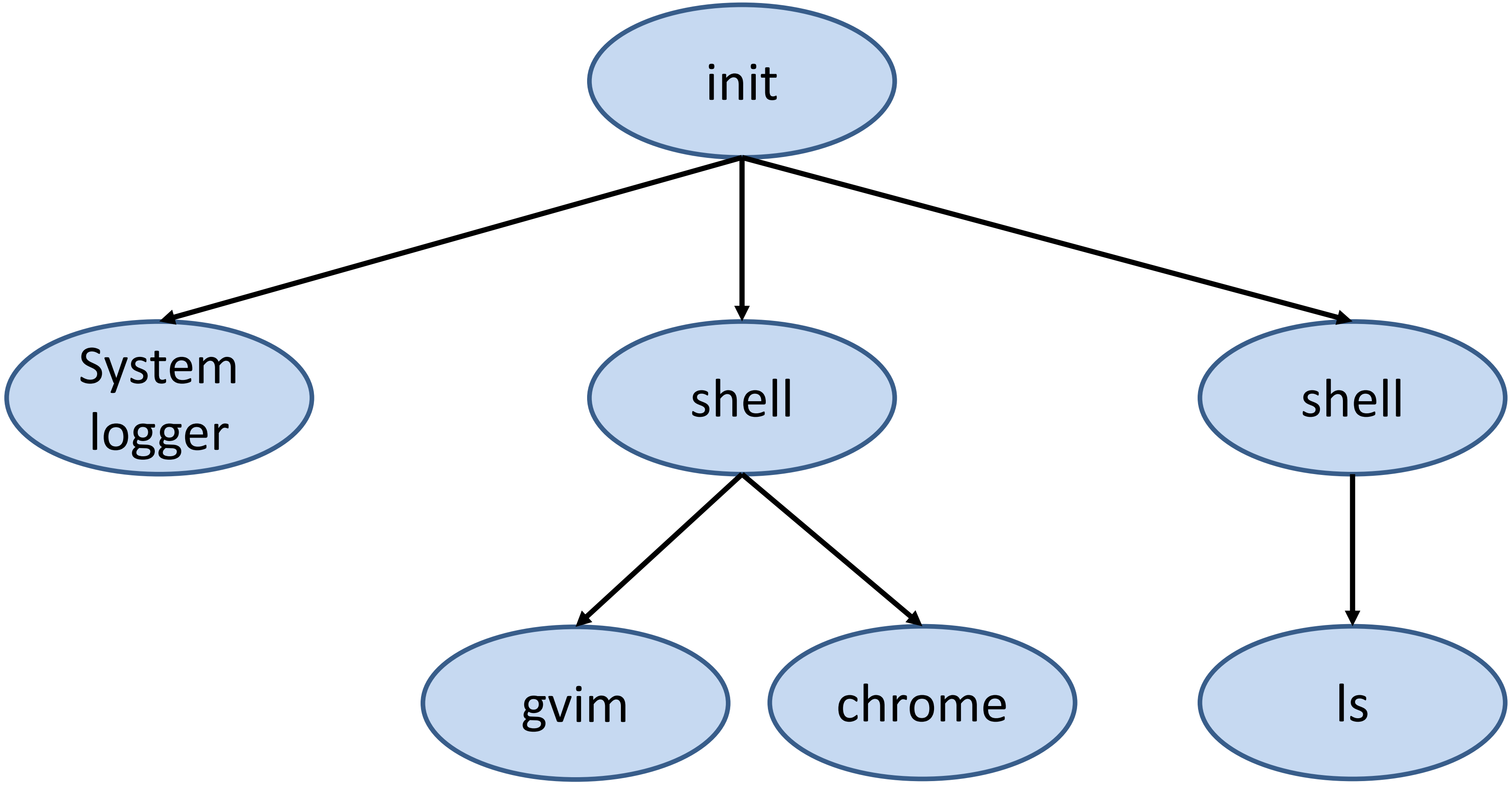
Process Control Block (PCB)

- What does the OS need to keep track of?
  - Process state
  - Process identifier
  - Owning user
  - Contents of registers
  - Program counter
  - Memory references
  - Others?

Process Table

- How do we track multiple processes?
- OS keeps a table of all PCBs for all processes
  - Indexed by process identifier

Process Hierarchy



Creating a Process

- fork()
  - System call that “splits” a processes into two
  - New process begins executing at the return from fork
  - Parent keeps executing after calling fork
  - Programmer can tell the difference based on fork return value
    - Return value in parent process – child process process identifier (pid)
    - Return value in the child process – 0
  - Questions:
    - How can the child figure out its pid?
    - How can the child process figure out the parent’s pid?
    - Is there a use to having multiple processes in a single program?

About fork

- "man fork" for all the details
- Parent and child processes
  - Execute the same source code
  - Do NOT share memory locations
  - Do share file descriptors
- Can we communicate easily between parent and child?
  - File system: named files, FIFOs, pipes
  - Shared memory
- Is there a better way? Threads