Operating Systems – Processes

ECE 344

The Process Concept

- An OS executes a variety of programs
  - In batch systems, referred to as ***jobs***
  - In time shared systems, referred to as user programs or ***tasks***
- So far pretty informally referred to as *programs in execution, processes, jobs, tasks* …
- From now on we'll try to use the term **process** synonymously with the above terms and really mean …

Definitions of Processes

Exact definitions in textbooks differ:

- Program in execution
- An instance of a program running on a computer
- A **unit of execution** characterized by
  - A single, sequential thread of execution
  - A current state
  - An associated set of system resources (memory, devices, files)
- A **unit of resource** ownership
- …

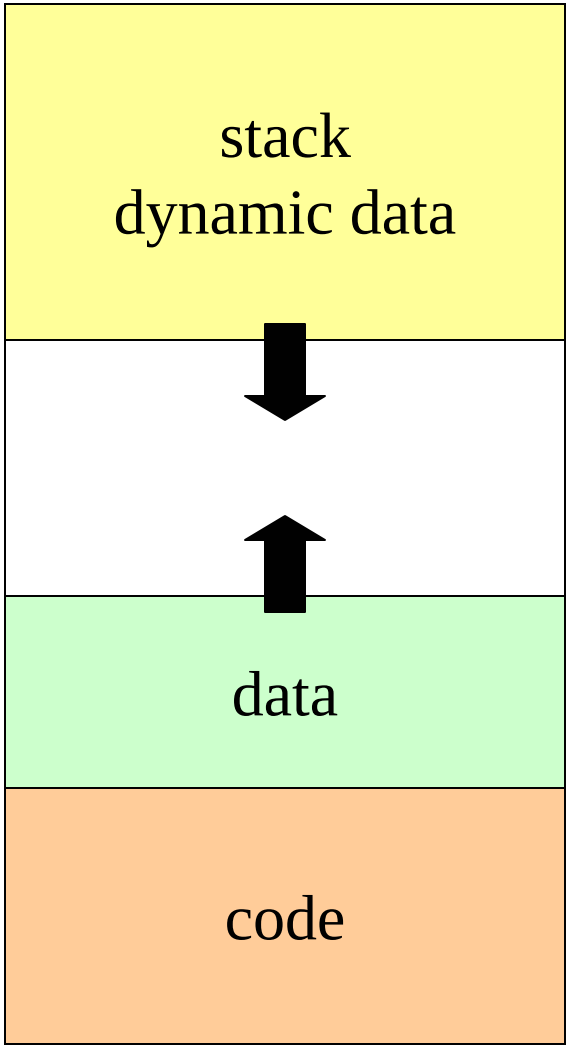
The OS has to …

- Load executable from hard disk to main memory
- Keep track of **the states** of each process currently executed
- Make sure
  - No process monopolizes the CPU
  - No process **starves** *to death*
  - Interactive processes are responsive
  - Processes are shielded from one another

Process Structure

A process consists of

1. An executable (i.e., code)
2. Associated data needed by the program (global data, dynamic data, shared data)
3. Execution context (or state) of the program, e.g.,
   - Contents of data registers
   - Program counter, stack pointer
   - Memory allocation
   - Open file (pointers)

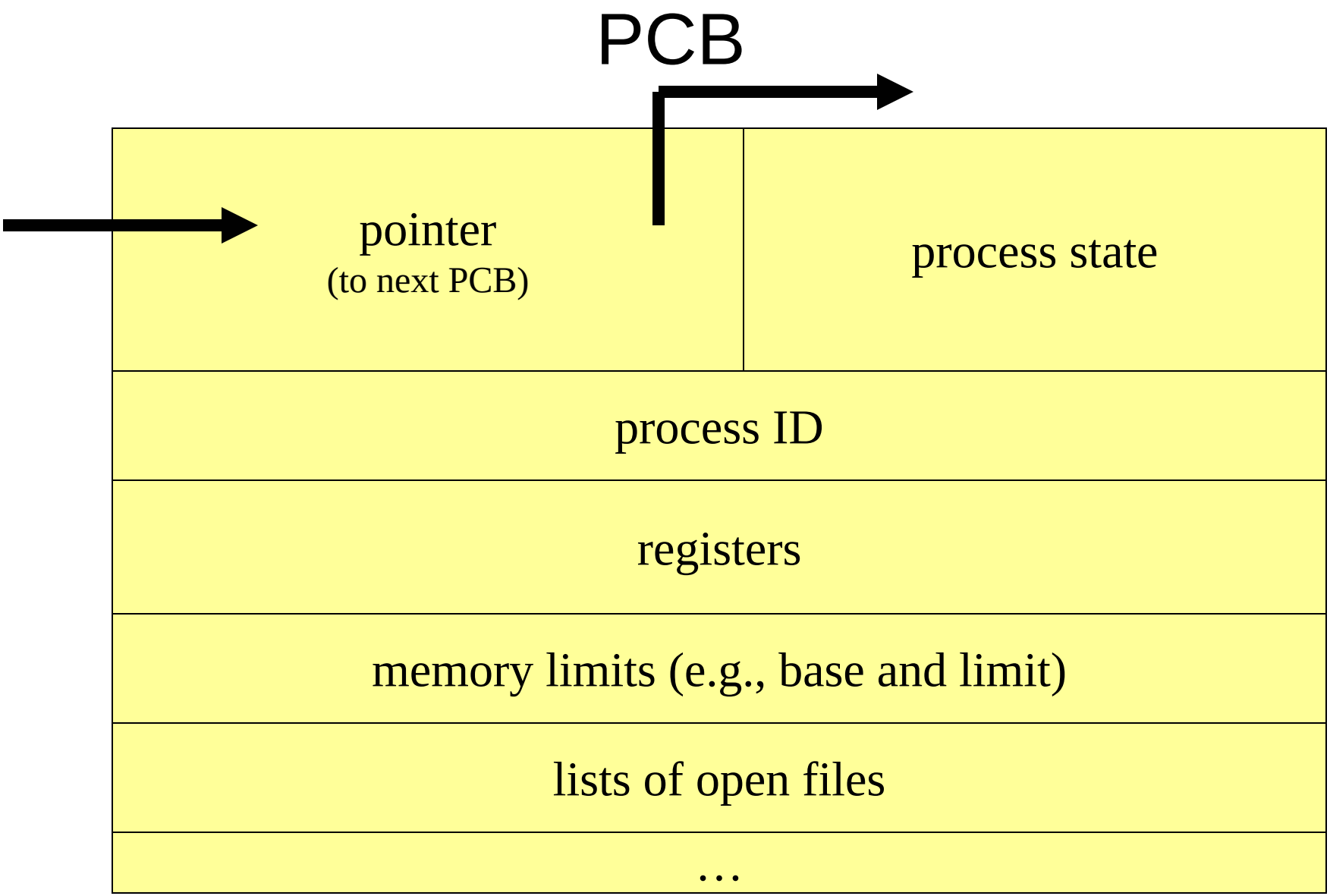


Processes

- 16 users may all be running an application (e.g., emacs), while there is only one image of “emacs” loaded in the system
- This image (i.e., binary) is **shared** among the different processes running on behalf of the 16 users
- I.e., code (and data) can be shared among processes
- Shared libraries, shared objects, .so, DDLs!

Process Control Block (PCB)

- Process state
- Program counter
- CPU registers
- CPU scheduling information (e.g., priority, scheduling queue information)
- Memory management information (e.g., base and limit registers)
- Accounting information (e.g., time)
- I/O status information


PCB
pointer
(to next PCB)
process state
process ID
registers
memory limits (e.g., base and limit)
lists of open files
…

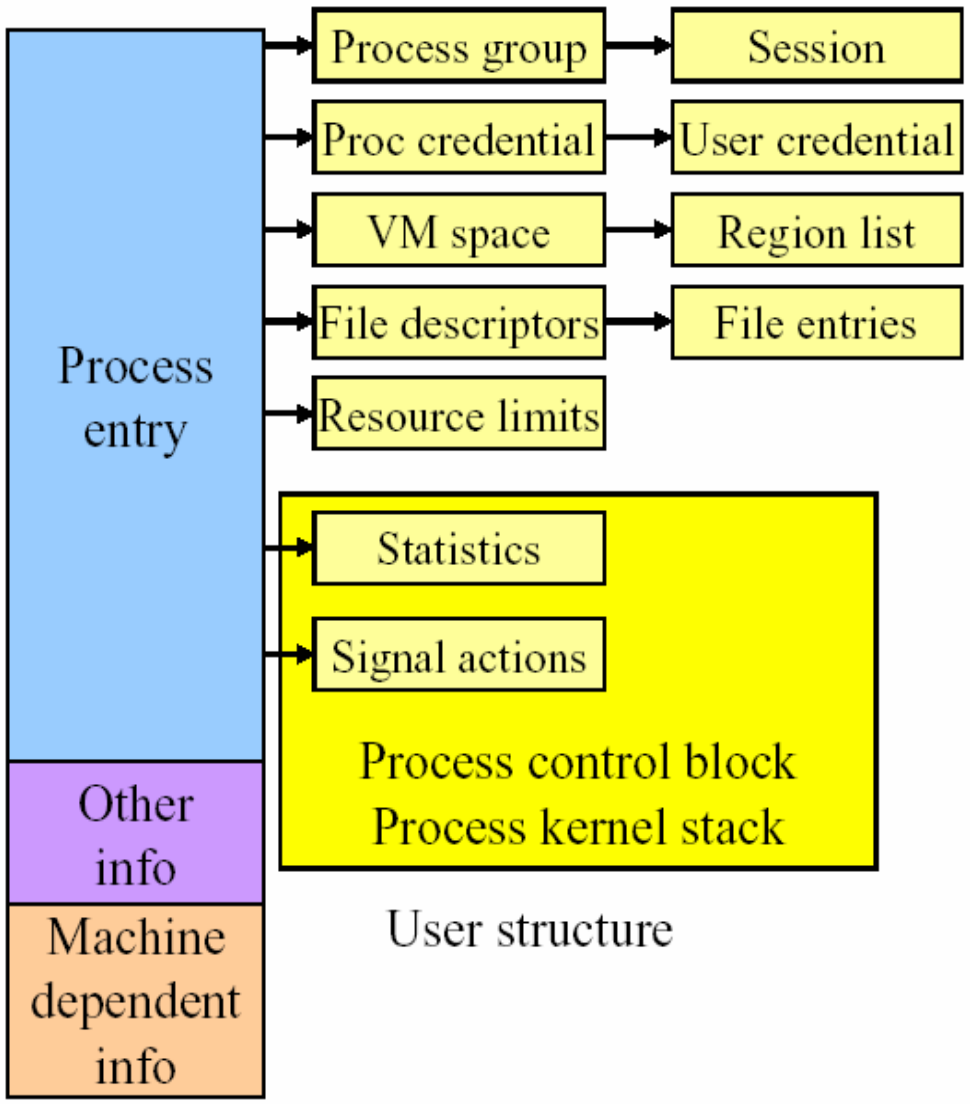
Process Organization in OS161
`thread.h`

Anything missing? **This is our notion of a process.**

```
struct thread {
        struct pcb t_pcb;
        char *t_name;
        const void *t_sleepaddr;
        char *t_stack;
        struct addrspace *t_vmspace;
        struct vnode *t_cwd;
};
```

```
$ more arch/mips/include/pcb.h
/*
 * Process Control Block: machine-dependent part of thread
 */
    struct pcb {
       // stack saved during context switch
       u_int32_t pcb_switchstack;
       // stack to load on entry to kernel
       u_int32_t pcb_kstack;
       // are we in an interrupt handler?
        u_int32_t pcb_ininterrupt
       // recovery for fatal kernel traps
        pcb_faultfunc pcb_badfaultfunc;
       // jump area used by copyin/out etc.
       jmp_buf pcb_copyjmp;                              };
```

Process Organization in BSD



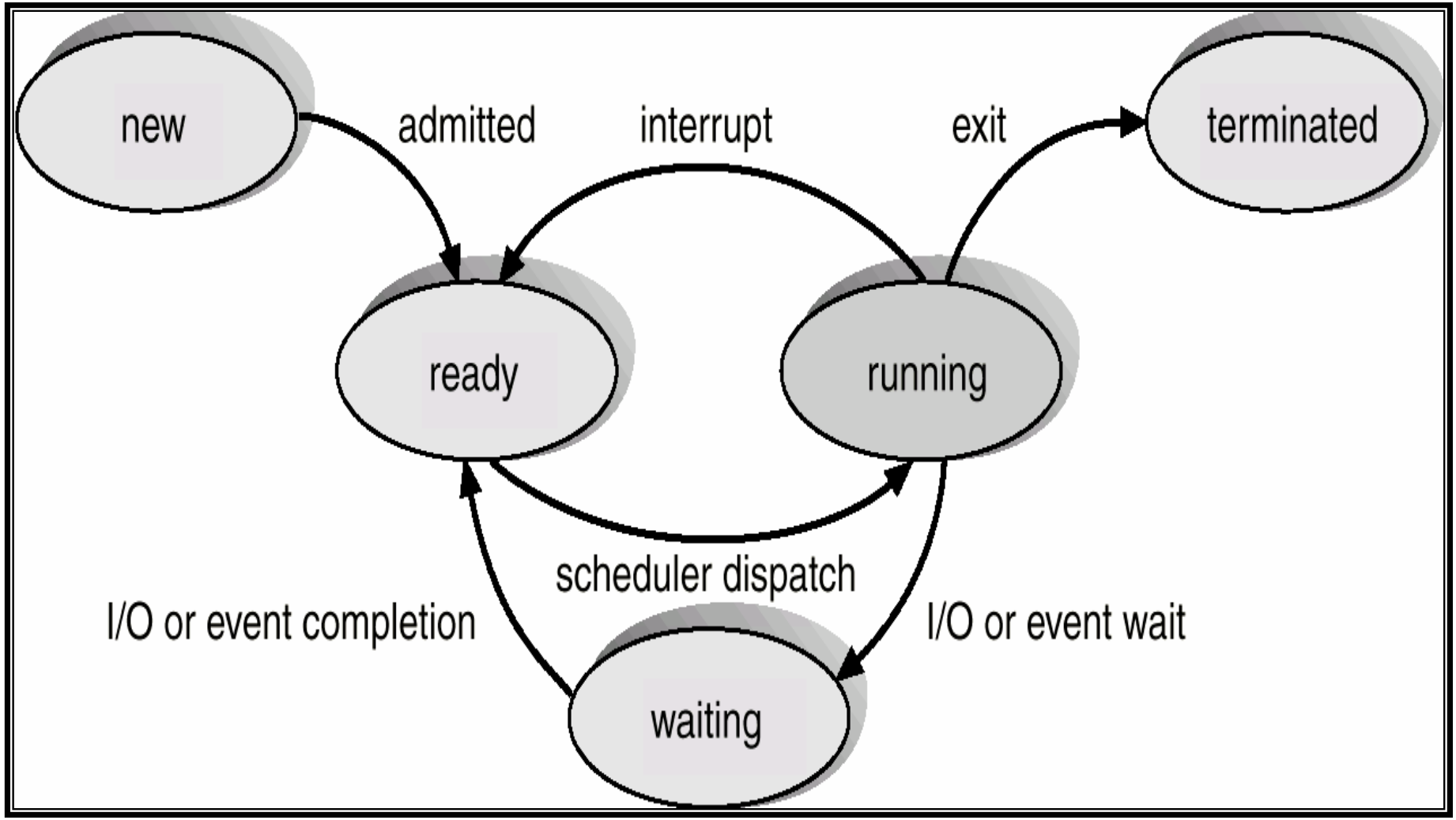
- Contents of process contrc block include
  - Process identifier
  - Scheduling info
  - Process state
    - Wait channel
  - Signal state
  - Tracing info
  - Machine state
  - Timers
- Other stuff is pointed to by process entry
  - Process group implements hierarchy of processes

Process State

As a process executes it changes state.

- **New**: the process is being created.
- **Running**: instructions are being executed.
- **Waiting**: the process is waiting for some event to happen.
- **Ready**: the process is waiting to be assigned to a processor.
- **Terminated**: the process has finished executing.

Diagram of Process States

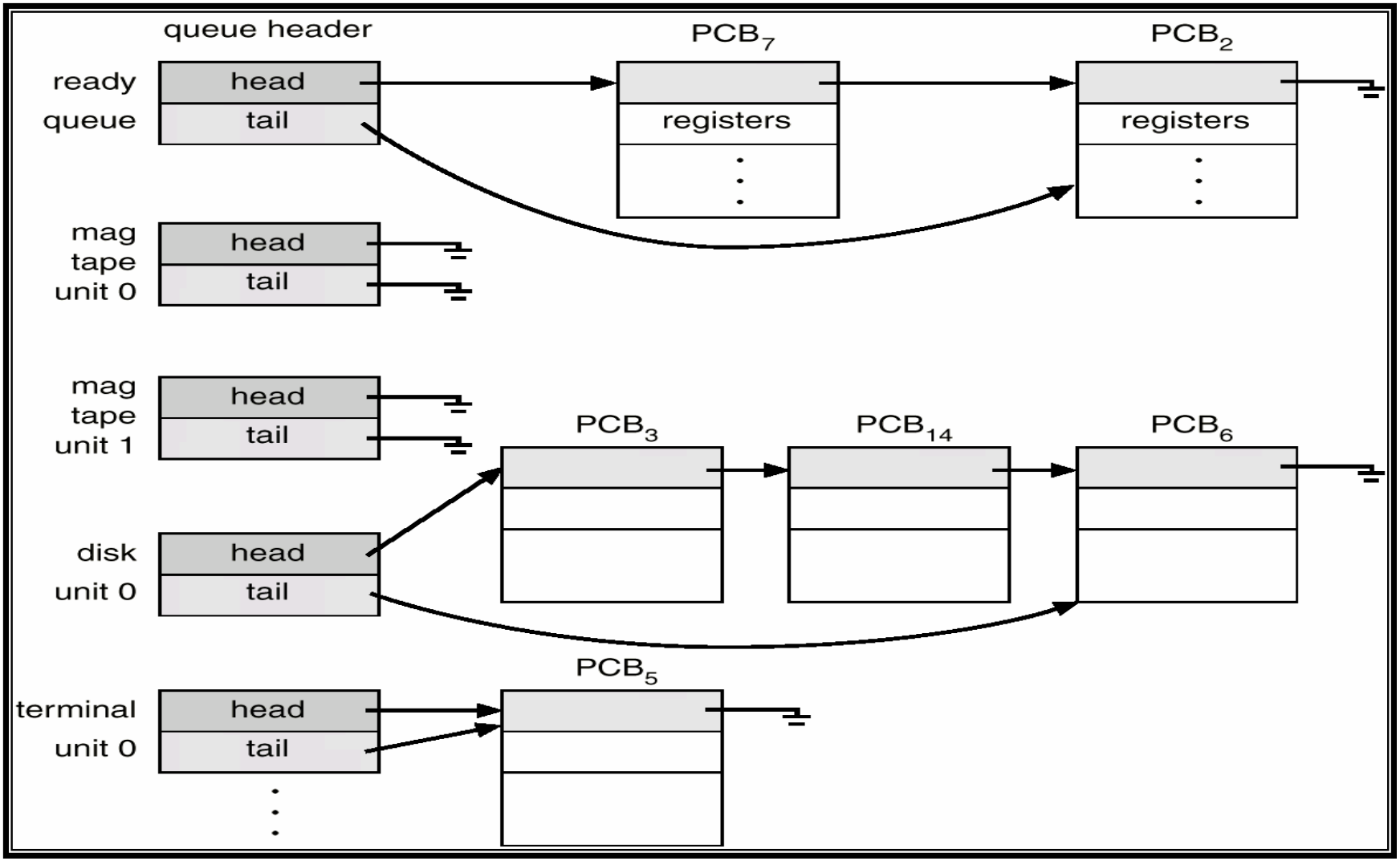


Process Scheduling Queues

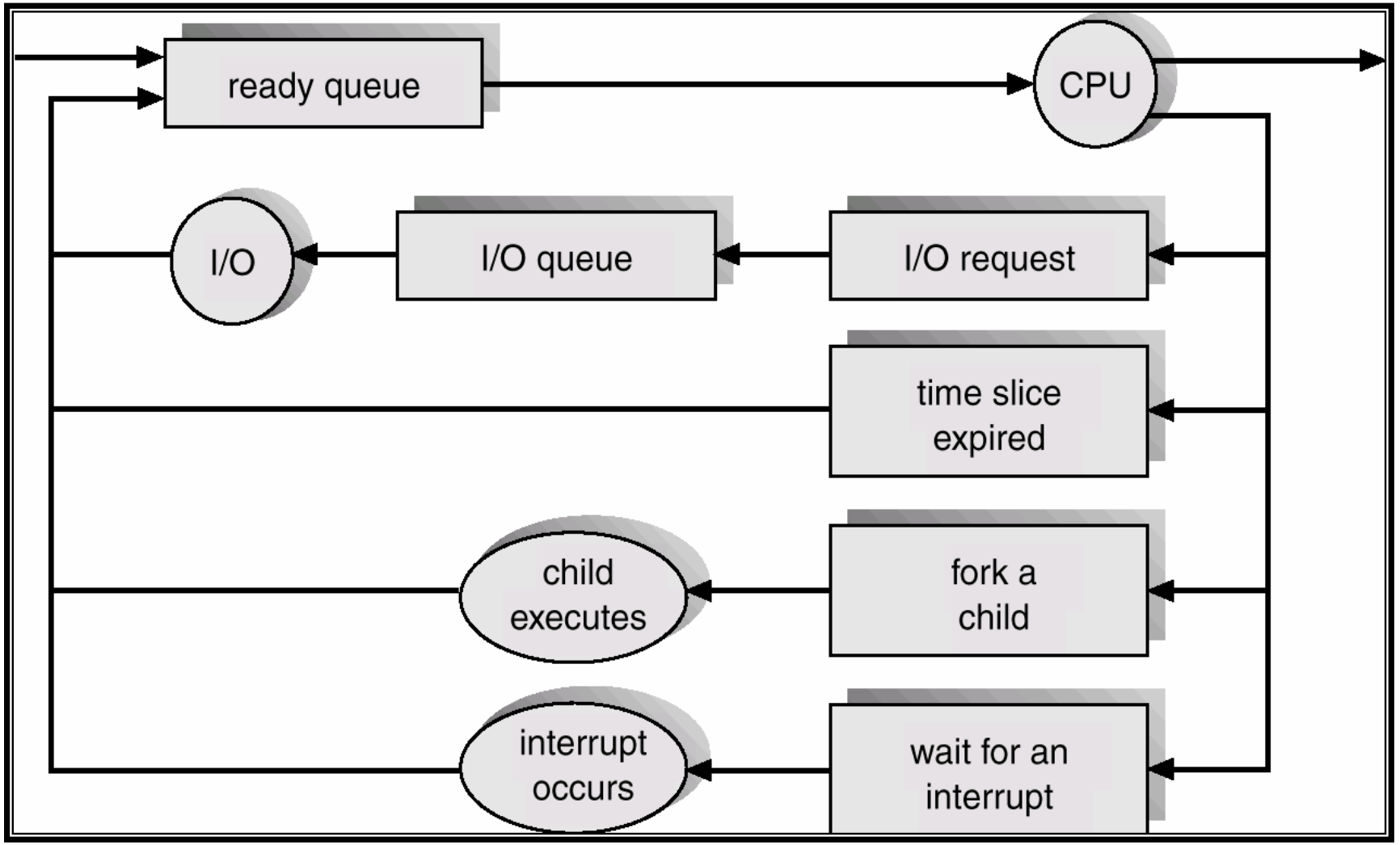
- **Job queue**: Set of all processes in the system.
- **Ready queue**: Set of all processes residing in main memory; ready and waiting to execute.
- **Device queues**: Set of processes waiting for an I/O device.

Processes migrate between the various queues

Example: Ready and Various Device I/O Queues



Process Scheduling



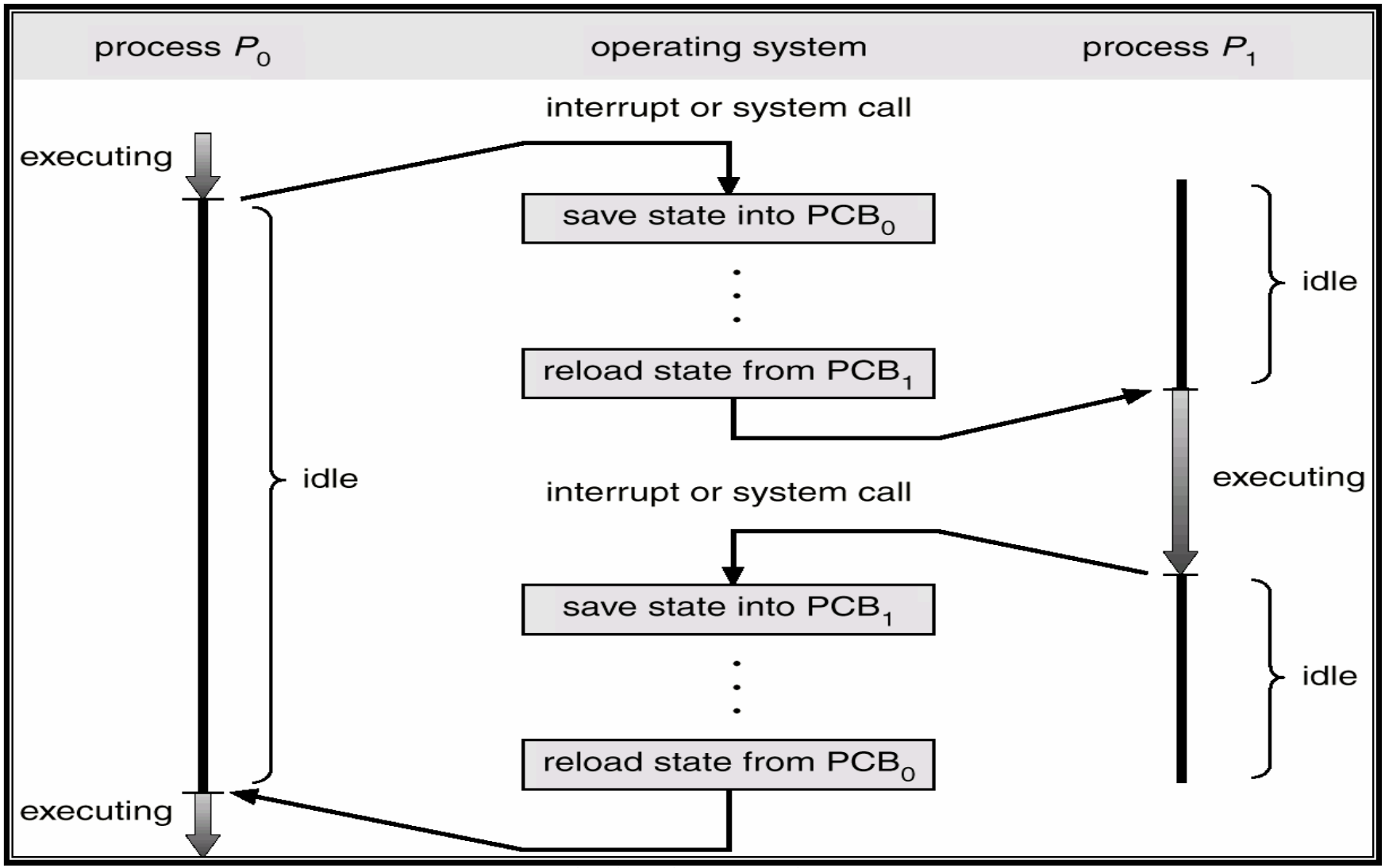
OS161 `kern/thread/thread.c`

```
/* States a thread can be in. */
typedef enum {
	S_RUN,
	S_READY,
	S_SLEEP,
	S_ZOMB,
} threadstate_t;
```

Context Switch

- CPU switching from one process to another process is called a **context switch**.
- Execution state of running process has to be **saved** and execution state of next process has to be **loaded (context is switched.)**.
- Time to save old and load new processes' execution state is called **context-switch time**.
- This time is **overhead**; The system does no useful work while switching. Needs to me small.
- Time depends on hardware support.

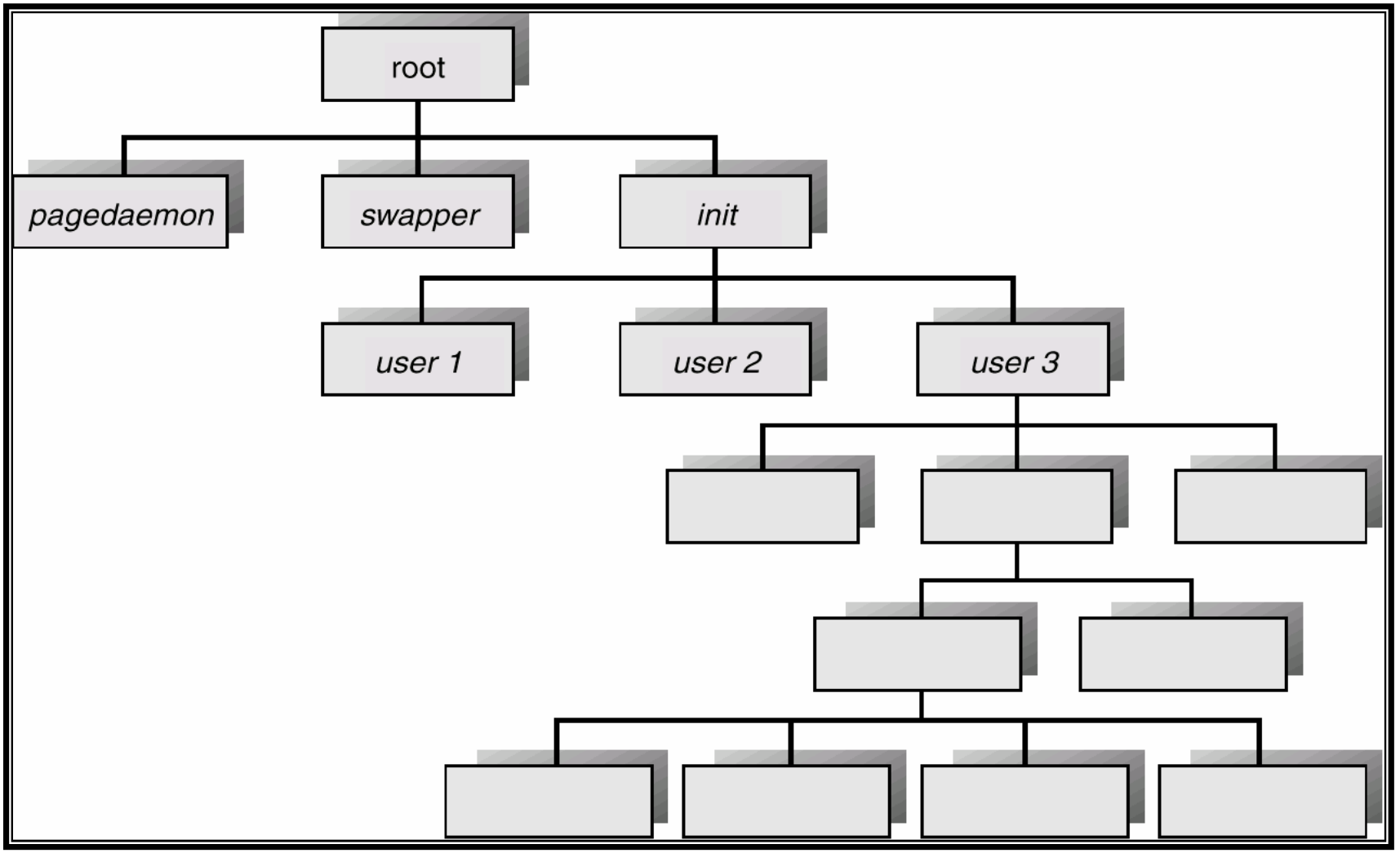
Context Switch Illustrated



Process Creation

- *Parent* processes create *child* processes, which in turn create other child processes forming **a tree of processes**
- Resource sharing (all, some, or nothing)
- Execution
  - Parent and child execute concurrently
  - Parent waits for child to terminate
  - Parent terminates prior to child process, which continues to execute
- The **init** process inherits processes whose parents have terminated (Unix)

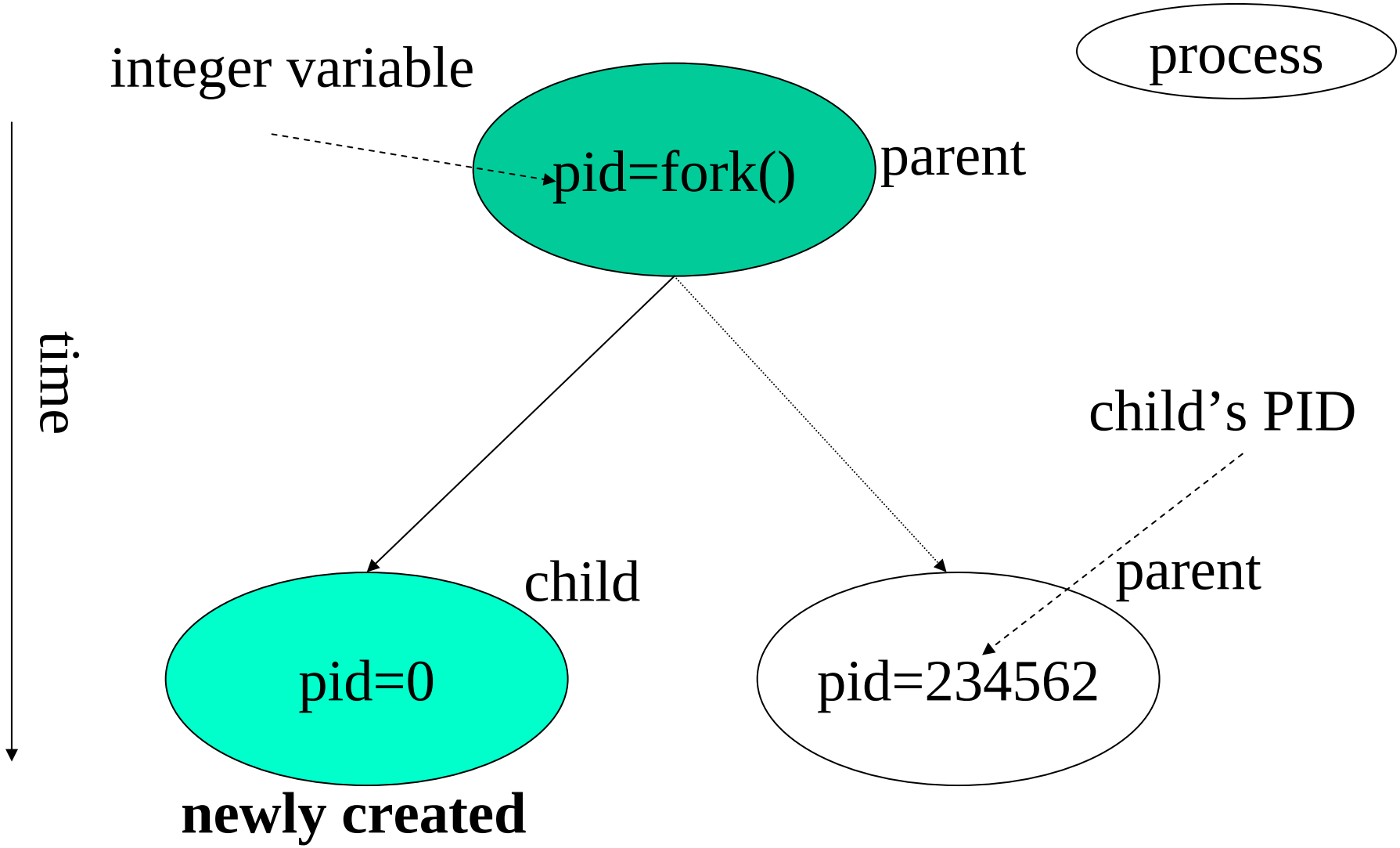
Process Tree on a Unix System



More on Process Creation

- Address space
  - Child is duplicate of parent process
  - Child has a program loaded into it's address space
- Unix examples
  - **Fork** system call creates a new process
  - **Exec** system call used after a fork to replace the process's address space with a new program
- Let's look at that in action …

```
#include <stdion.h>
void main(int argc, char* argv[]){
  int pid;
  pid = fork()
  if (pid < 0){  /* error occurred */
    fprintf(stderr, “Fork Failed”);
    exit(-1);
  }
  else if (pid == 0) {  /* child process */
    execlp (“/bin/ls”, “ls”, NULL);
  else {  /* parent process */
   /* parent will wait for child to complete */
   wait(NULL);
   printf(“Child completed”);
   exit(0);
  }
}
```


process
integer variable
pid=fork()
parent
time
child's PID
child
parent
pid=0
pid=234562
newly created

Summary

- Processes and their characterization
- Process control block
- PCB management, process states and state transitions
- Context switch
- Process creation