OPERATING SYSTEMS

Process Concept

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Some Literature uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion

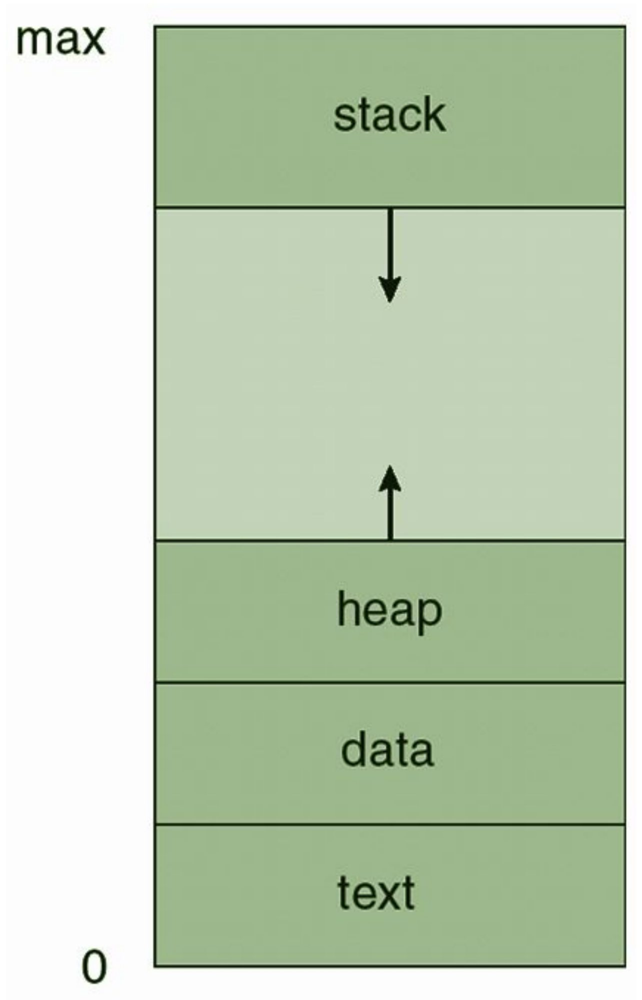
Process Concept

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
- Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
    - Consider multiple users executing the same program

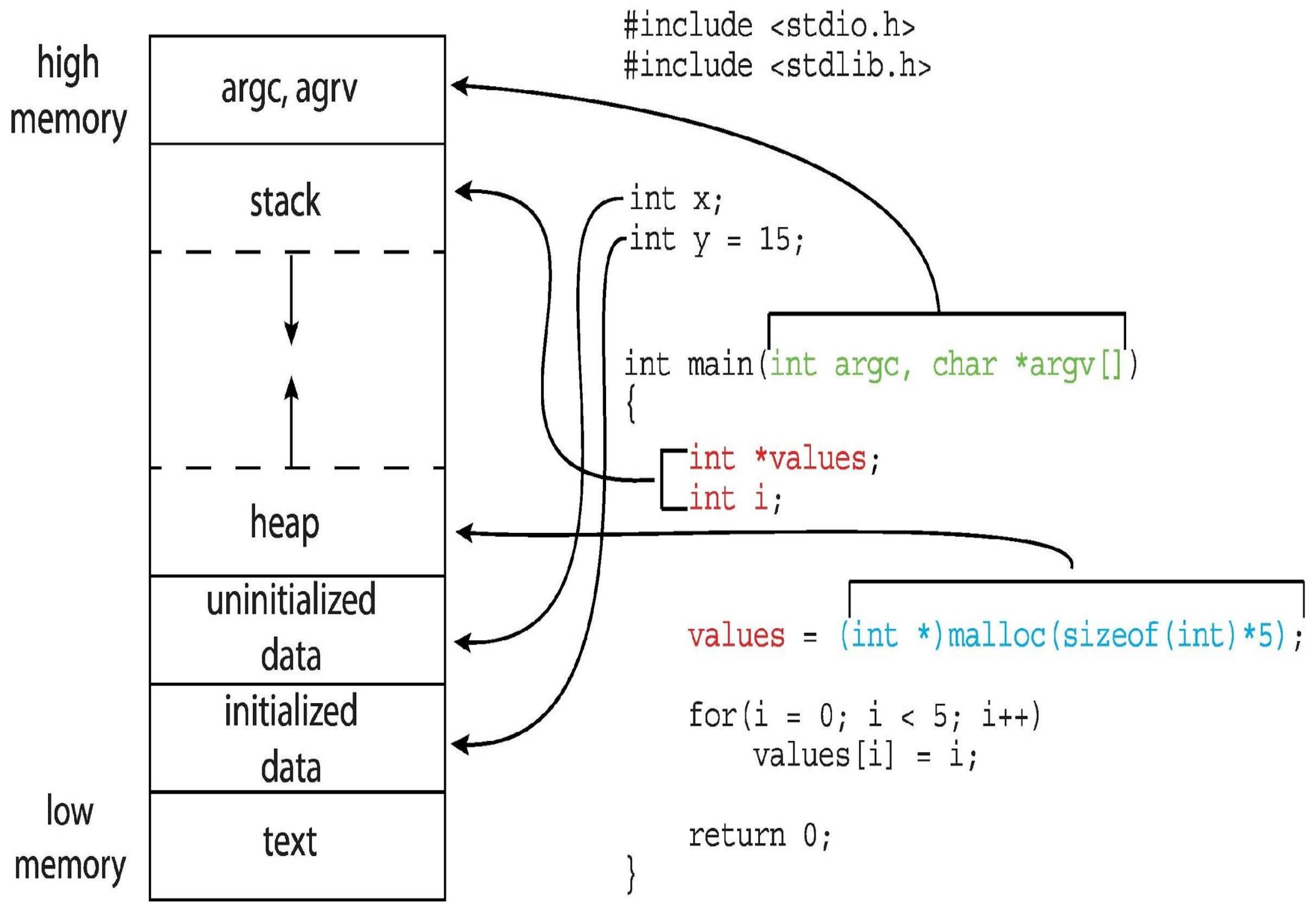
Process Concept

- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

Process in Memory



Memory Layout of a C Program

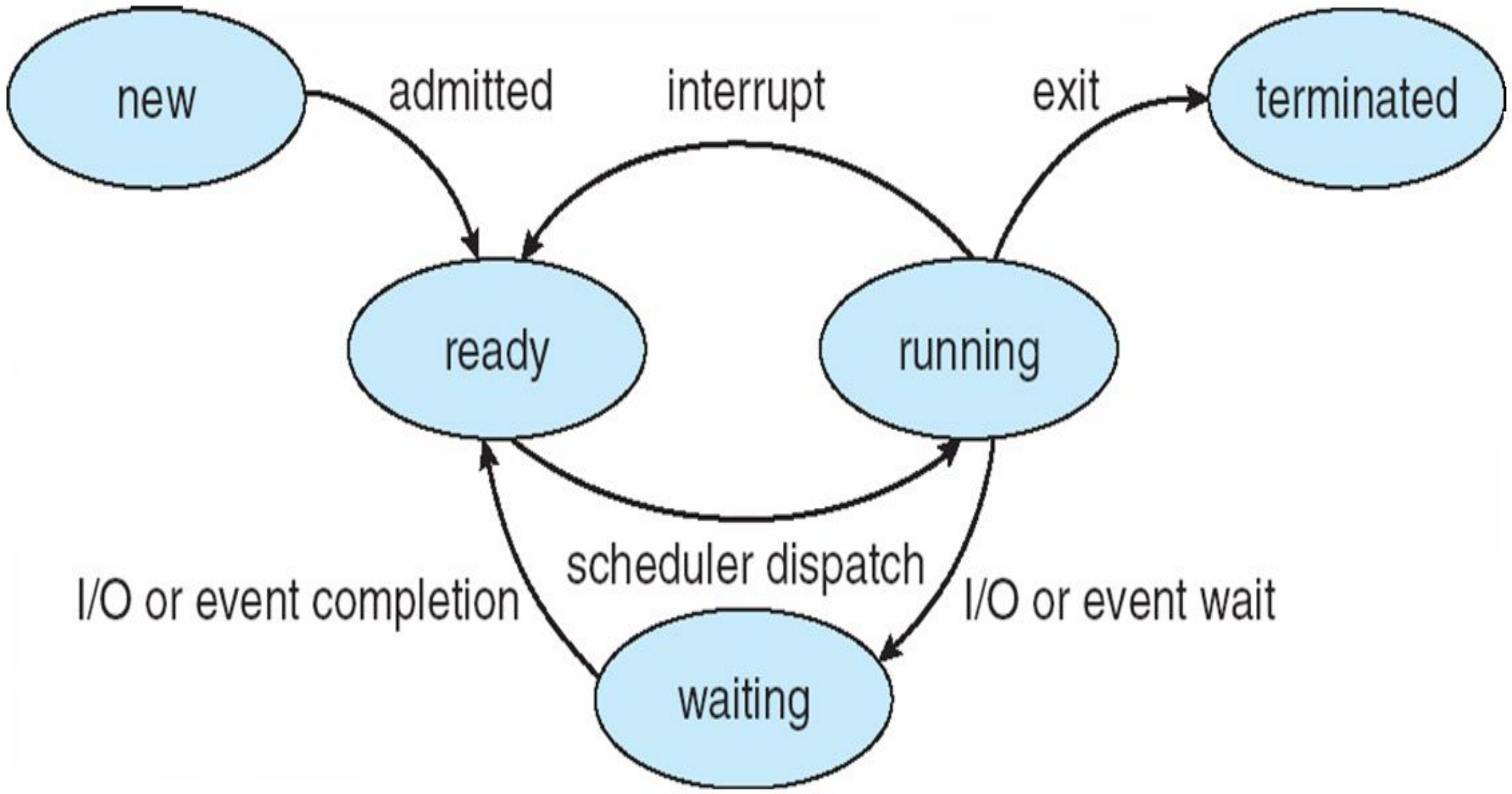


```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
   int *values;
   int i;

      values = (int *)malloc(sizeof(int)*5);

      for(i = 0; i < 5; i++)
         values[i] = i;

      return 0;
}
```

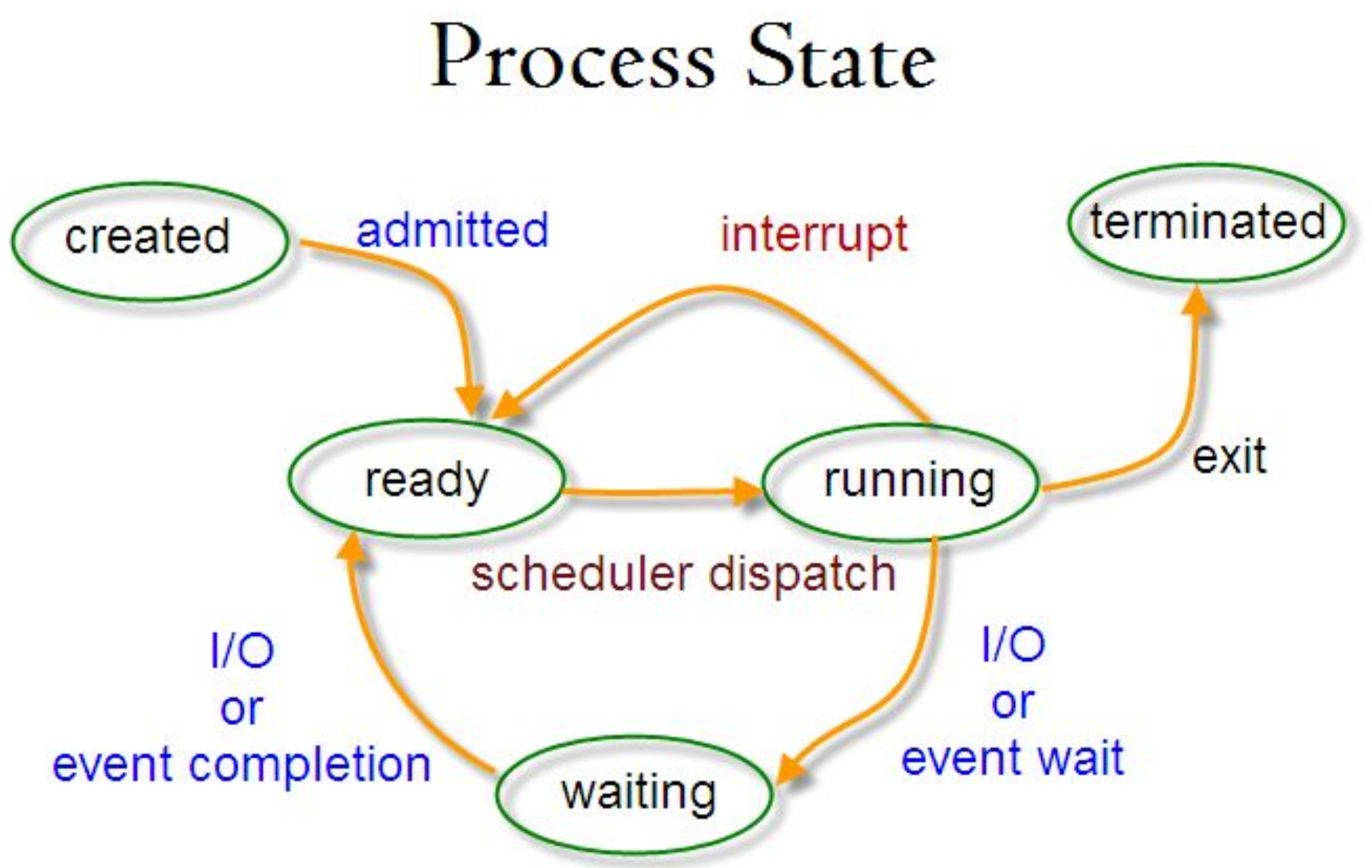
Process State

- 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 occur
  - **Ready**: The process is waiting to be assigned to a processor
  - **Terminated**: The process has finished execution

Process State Diagram



Process State Diagram - Linux

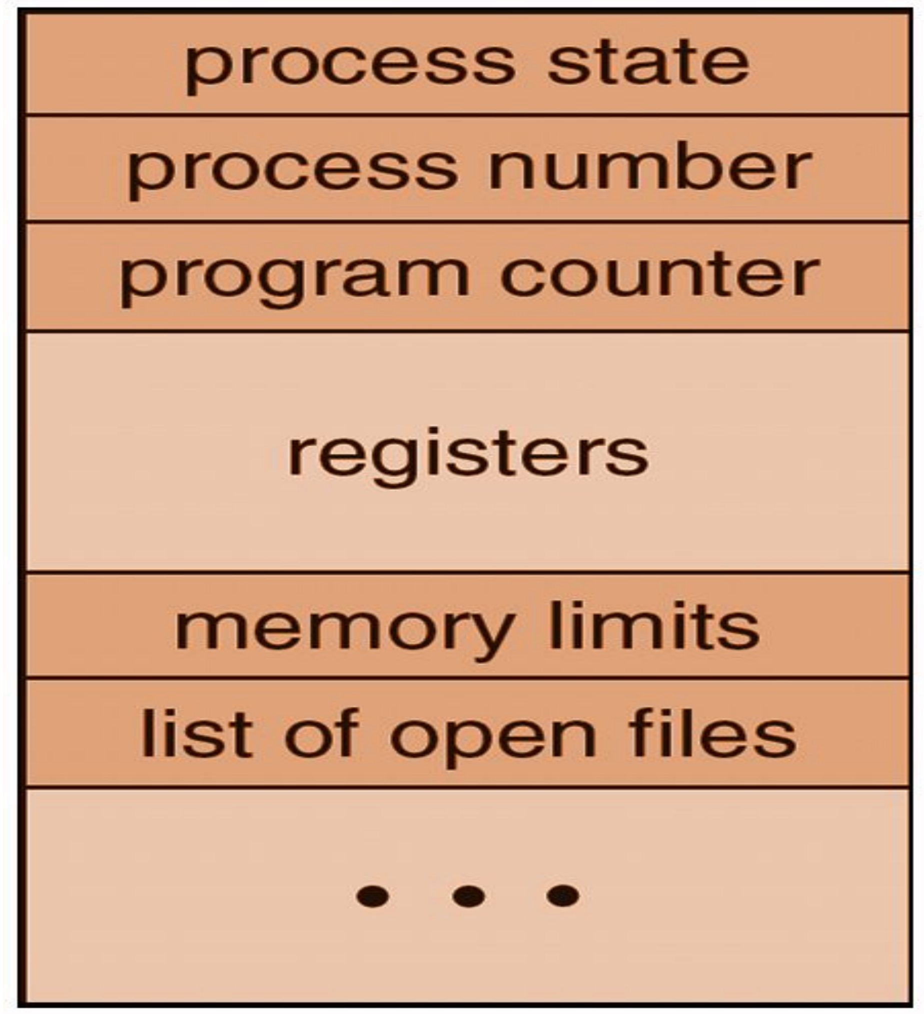


Process Control Block (PCB)

Information associated with each process (also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

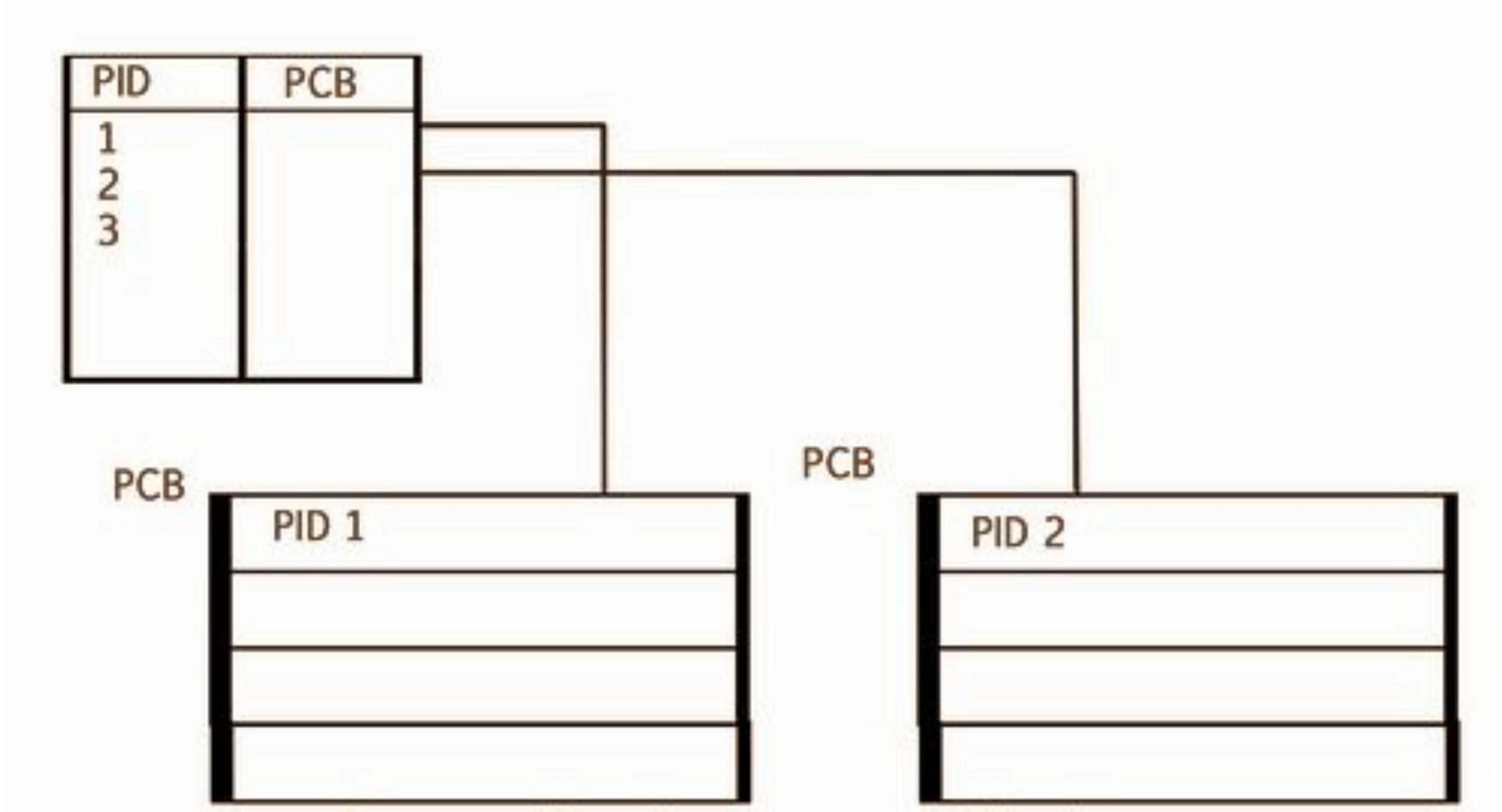
PCB Representation



- **Pointer –** It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
- **Process state –** It stores the respective state of the process.
- **Process number –** Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
- **Program counter –** It stores the counter which contains the address of the next instruction that is to be executed for the process.
- **Register –** These are the CPU registers which includes: accumulator, base, registers and general purpose registers.
- **Memory limits –** This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.
- **Open files list –** This information includes the list of files opened for a process.

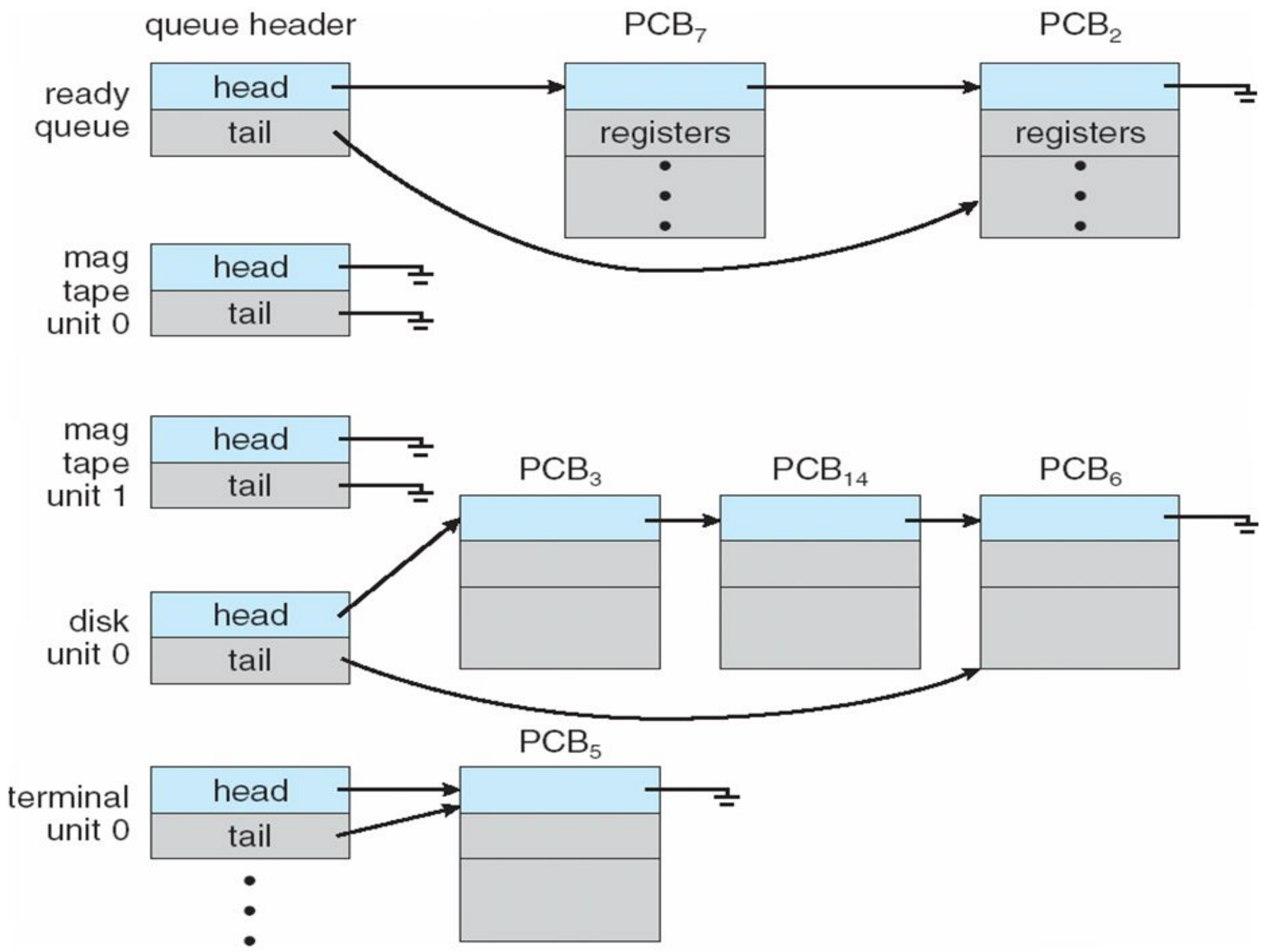
OPERATING SYSTEMS

Ready Queue And Various I/O Device Queues

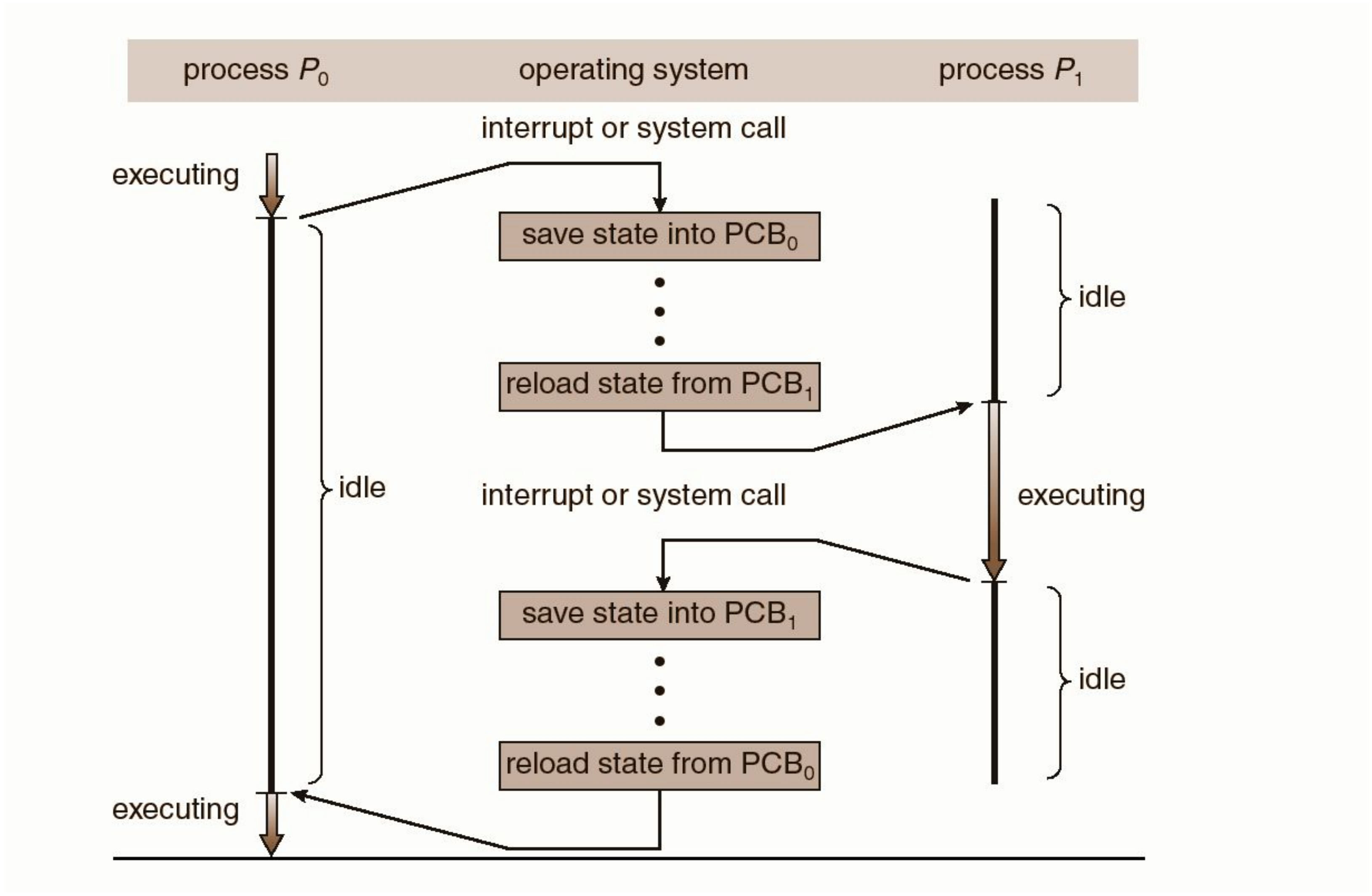


Process table and process control block

Ready Queue And Various I/O Device Queues



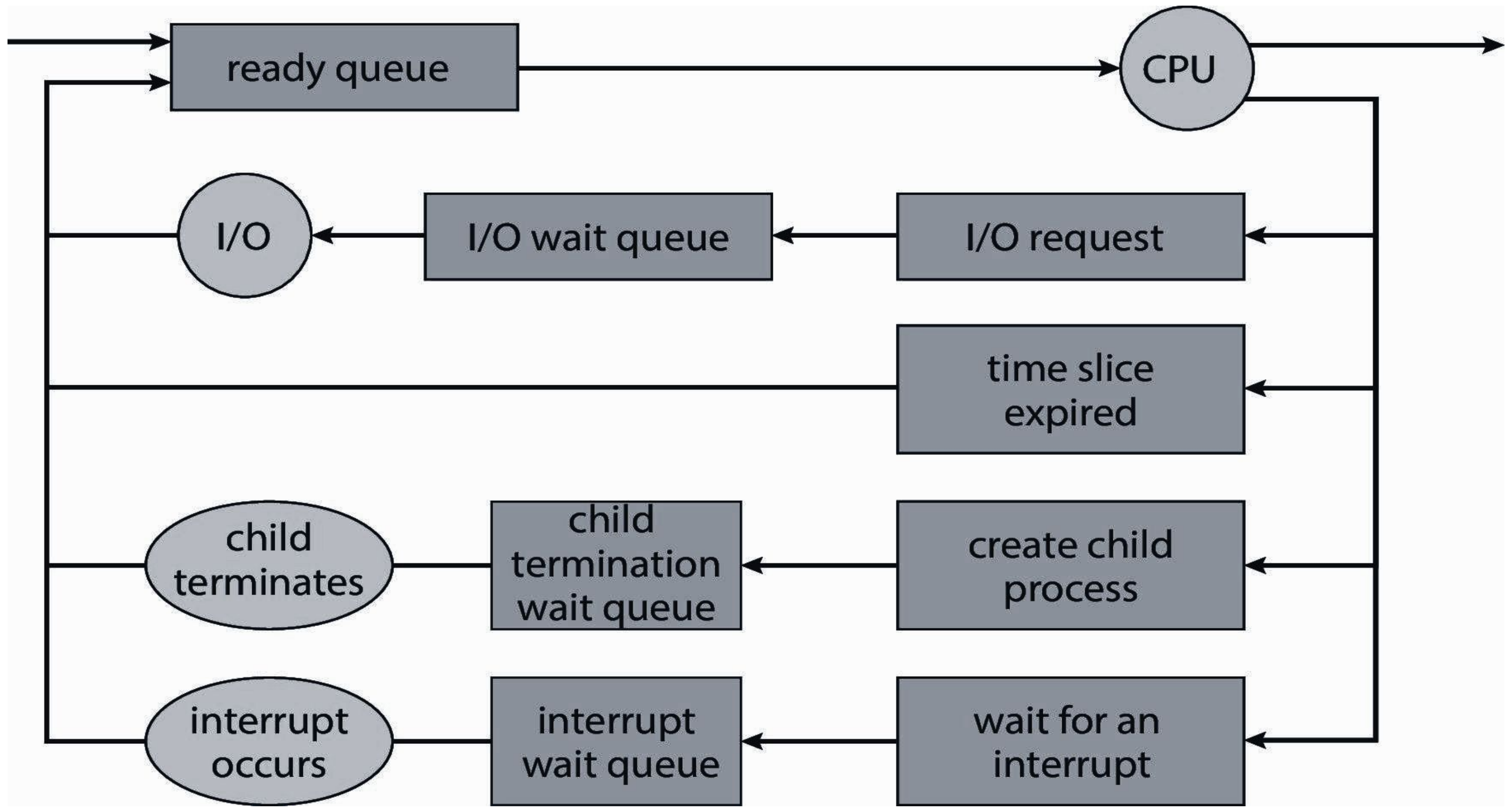
CPU switch from process to process



Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
    - **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 among the various queues

Representation of Process Scheduling



Schedulers

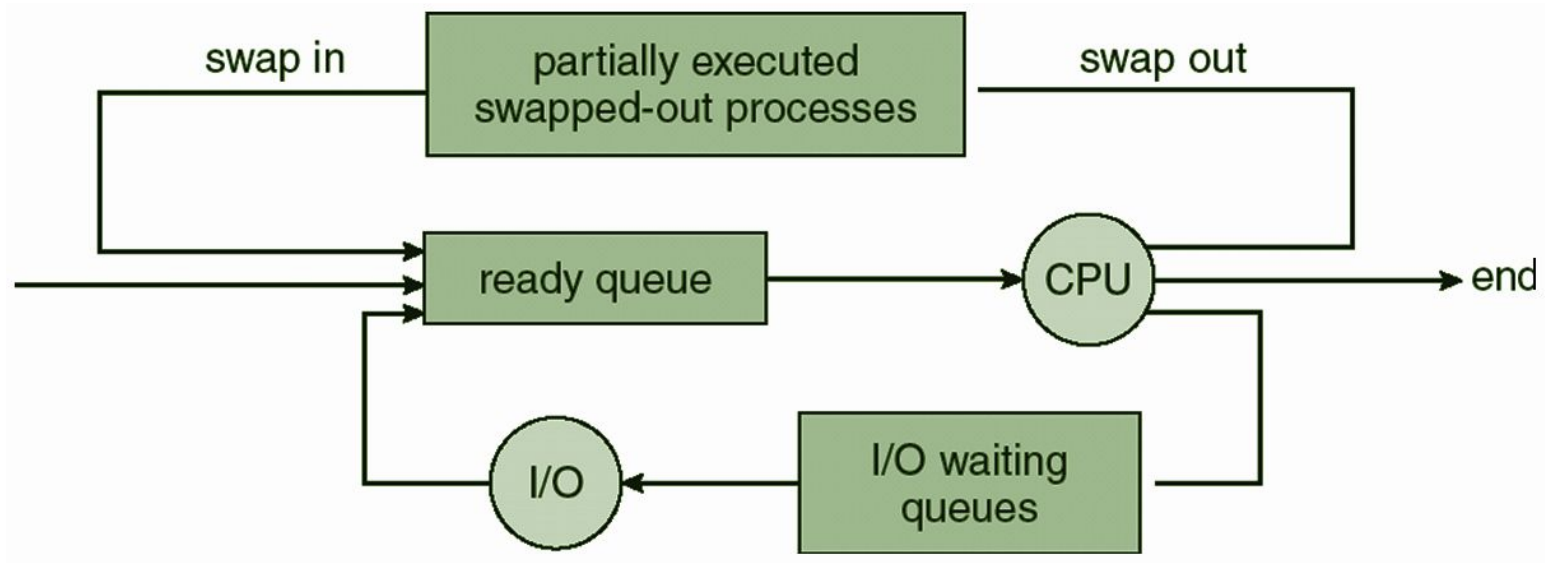
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**

Schedulers

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Representation of Process Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



Context Switch

- When CPU switches to another process, 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 is represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU - multiple contexts loaded at once

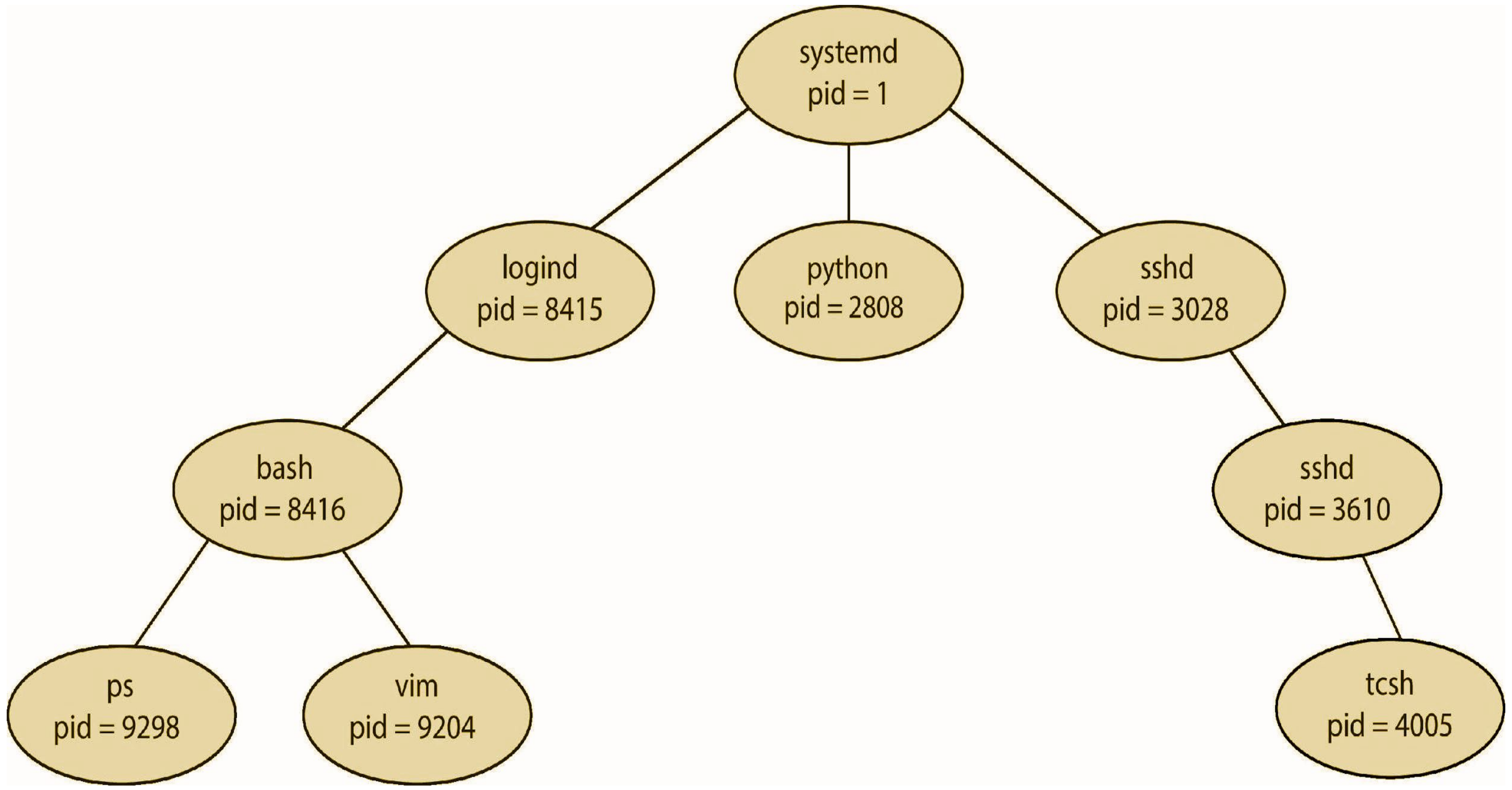
Operations on Processes

- System must provide mechanisms for:
  - process creation
  - process termination

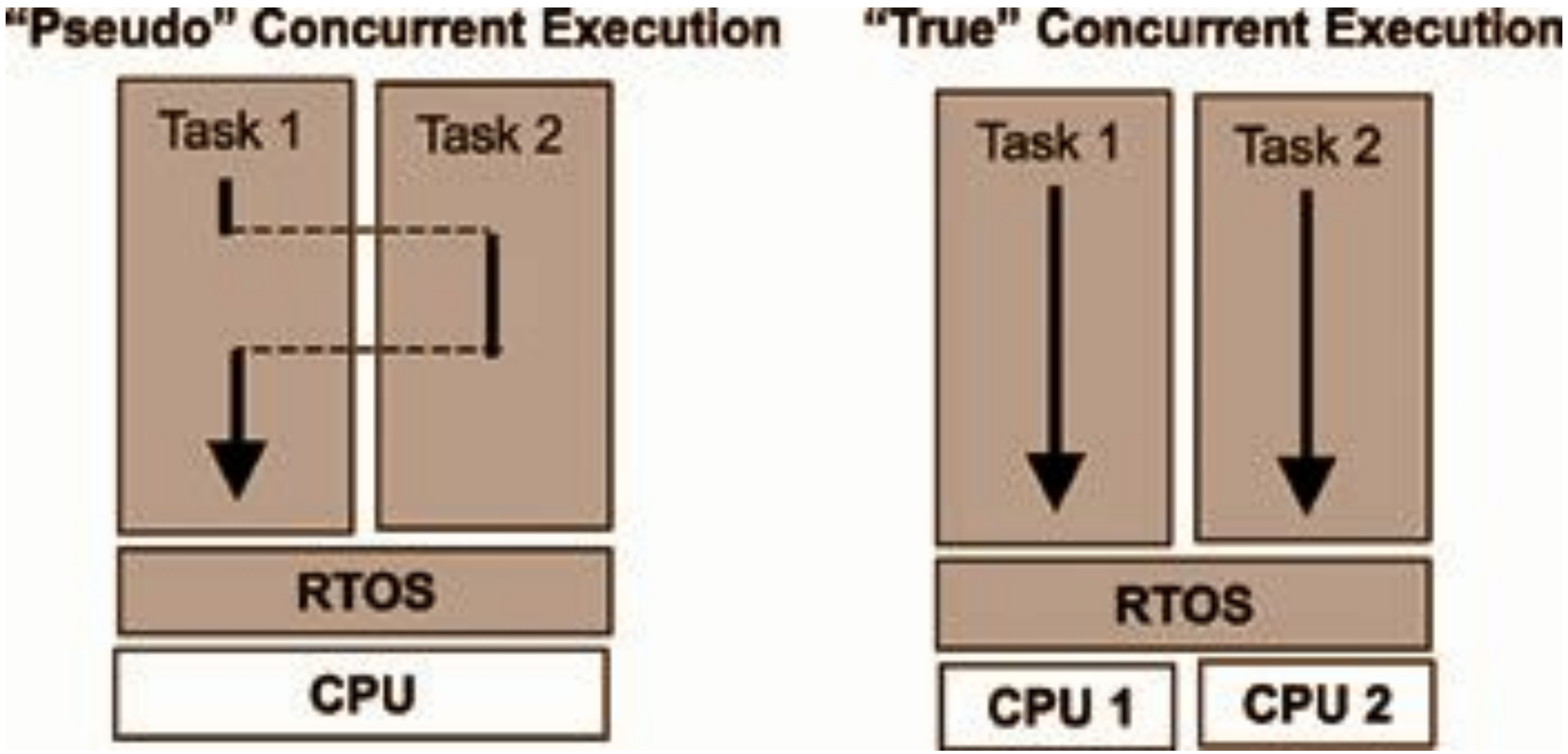
Process Creation

- **Parent** process creates **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

A tree of processes in Linux

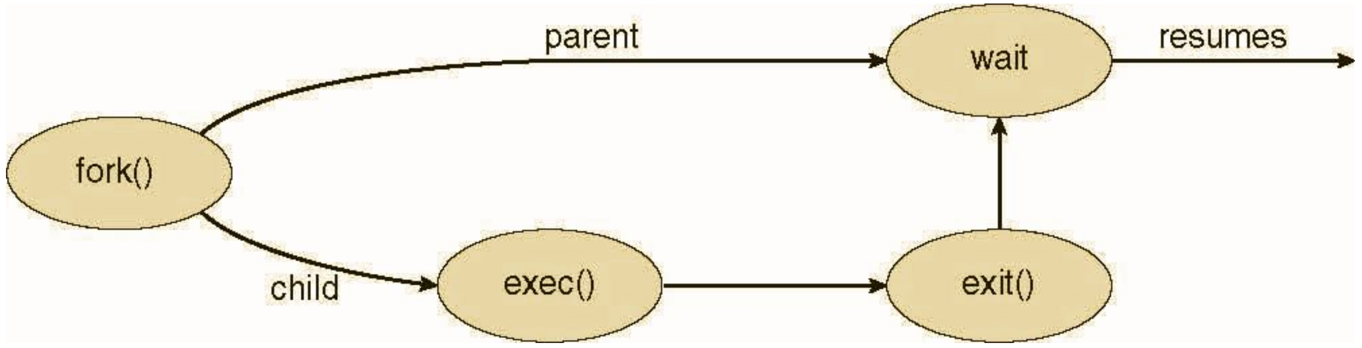


Pseudo Concurrency and True Concurrency



Process creation using fork()

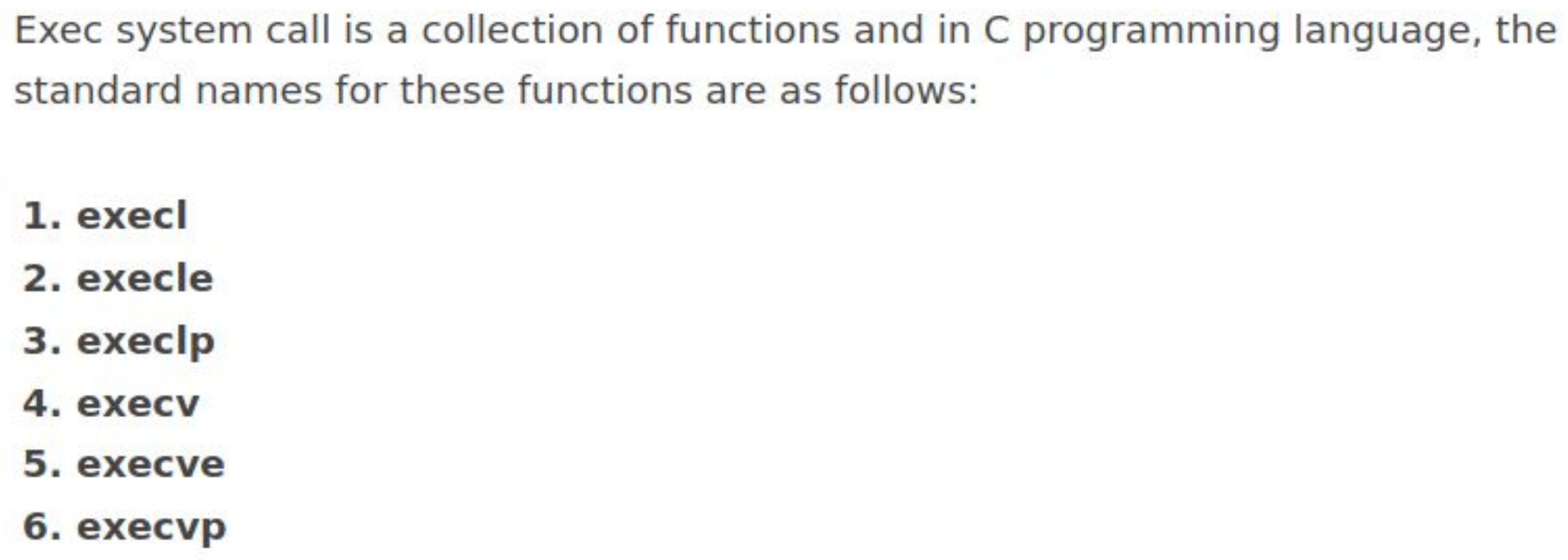
- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork()** system call creates new process
  - **exec()** system call used after a **fork()** to replace the process' memory space with a new program



Use of fork() - Try this code - To understand forking a process

```
i#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
  printf("\nHello B.Tech Computer Science! with pid=>%d",getpid());
   printf("\nHello B.Tech Computer Science! with ppid=>%d",getppid());

   fflush(stdout); // is required some times on some systems
                                  //before because the stdout
                                  // else it repeats the printf statements has to be flushed out
   fork();
   printf("\nHello B.Tech Computer Science! after fork with pid=>%d",getpid());

   printf("\nHello B.Tech Computer Science! after fork with ppid=>%d",getppid());

         printf("\n\n\n\n");
   return 0;
}
```

Process creation using fork()



Exec system call is a collection of functions and in C programming language, the standard names for these functions are as follows:

1. **execl**
2. **execle**
3. **execlp**
4. **execv**
5. **execve**
6. **execvp**

https://man7.org/linux/man-pages/man3/exec.3.html

Process Termination

- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
  - Returns status data from child to parent (via **wait()**)
  - Process' resources are deallocated by operating system

- Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Process Termination

- Some operating systems do not allow child to exist if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.

- The parent process may wait for termination of a child process by using the **wait()**system call**.** The call returns status information and the pid of the terminated process
- **pid = wait(&status);**
- If no parent waiting (did not invoke **wait()**) process is a **zombie**
- If parent terminated without invoking **wait** , process is an **orphan**

C Program forking Separate Process

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

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;
}
```

Use of fork() - Try the code given in the notes section and give your opinion on zombie process

Hint: Use Two terminal one to run a.out and other to run top command and identify the zombie process by looking at the process id from the output of a.out. This you have to do with in sleep time specified in the parent process

Use of fork() - Try the code given in the notes section and give your opinion on orphan process

Hint: Use Two terminal one to run a.out and other to run ps -ef | grep <username>command and identify the orphan process by looking at the process id from the output of a.out. This you have to do with in sleep time specified in the child process.

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com