Processes I (CS-351)

Agenda

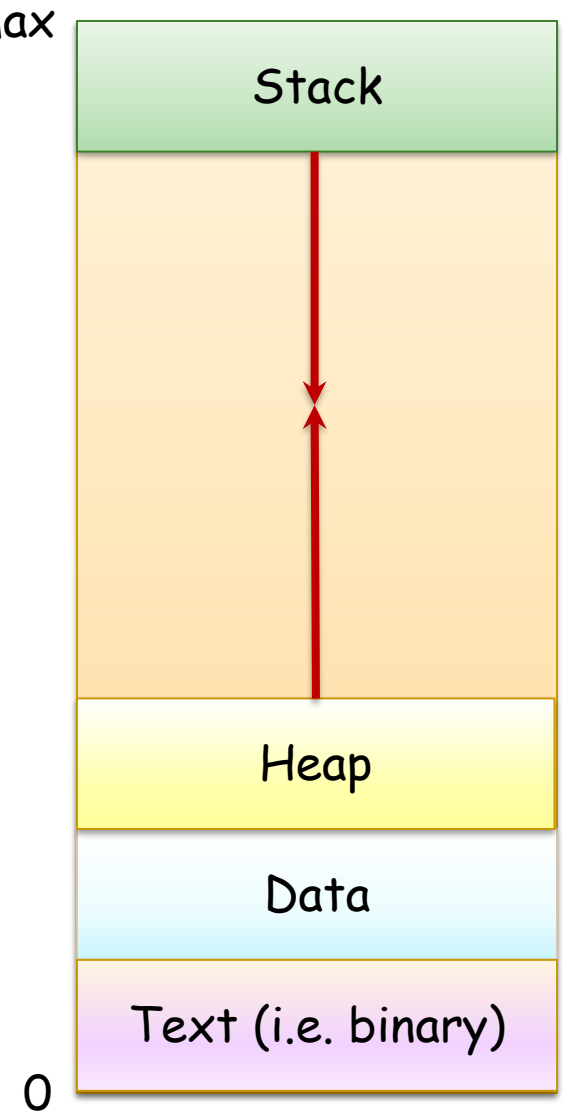
- What is a process?
- Process memory and state
- Process Control Block (PCB)
- Process Scheduling
- Operations on Processes: Parents and Children
- Interprocess Communications

Processes

- Process: a unit of work on the system.
- Recall: Program vs. Process:
  - Program: is a set of instructions (a passive entity).
  - Process: a program in execution (an active entity).
- Processes need resources: CPU, memory, files, and I/O devices.
- Application vs. Process: an application may consist of multiple processes e.g., Google Chrome.

Process Memory

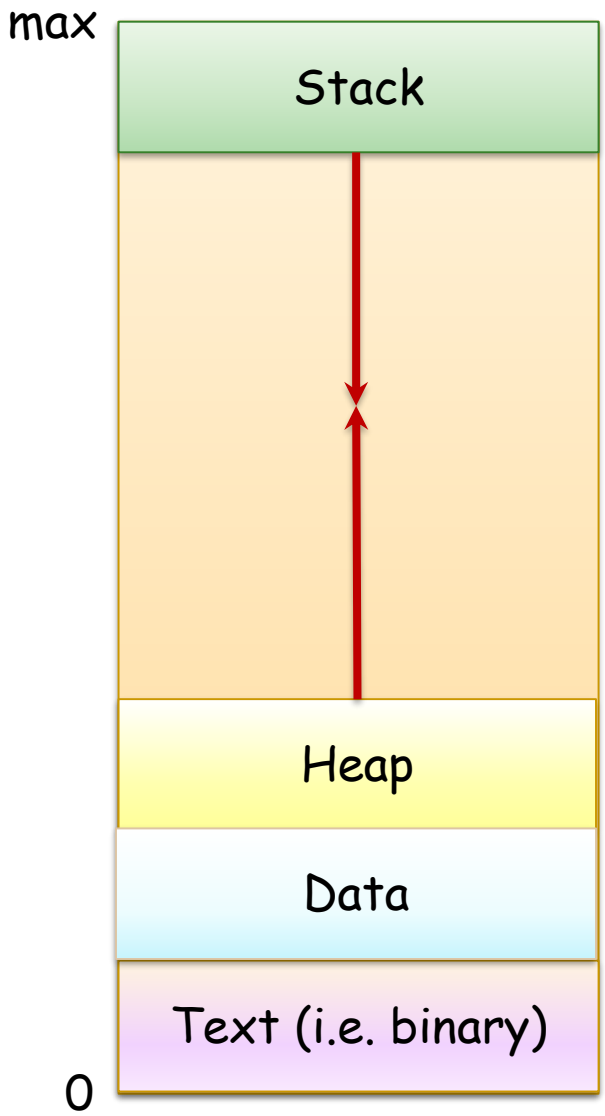
- Stack: stores function parameters, local variables, and return address.
- Heap: contains dynamic memory allocated during process runtime.
- Data: contains global variables.
- Text: stores the program instructions (i.e., the executable).



Process Memory: Example

```
int a = 1;
int f(int c) { int b = c+1; return b;}

int main()
{       int d = 2;
        char* arr = new char[100];
        for(int i = 0; i < 100; ++i)
                arr[i] = '\0';
        f(d);
        delete arr;
        return 0;
}
```



Student Participation: Process Memory

Link the two columns

- Stack
- Heap
- Data
- Text

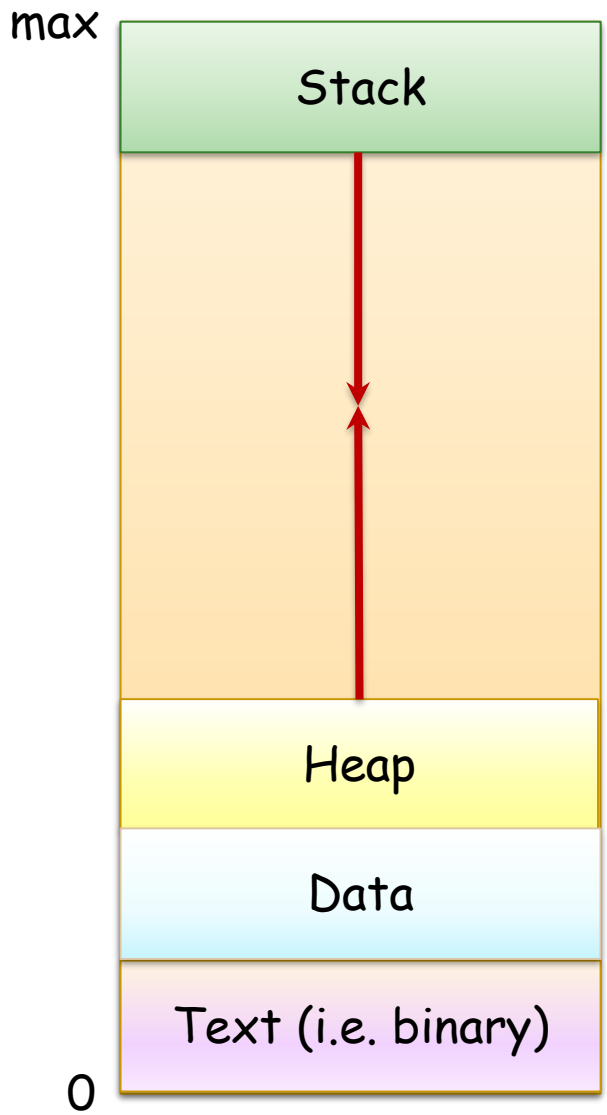
- a
- b
- c
- d
- arr
- i
- new char[100]

Process Memory: Example

```
int a = 1;
int f(int c) { int b = c+1; return b;}

int main()
{       int d = 2;
        char* arr = new char[100];
        for(int i = 0; i < 100; ++i)
                arr[i] = '\0';
        f(d);
        delete arr;
        return 0;
}
```

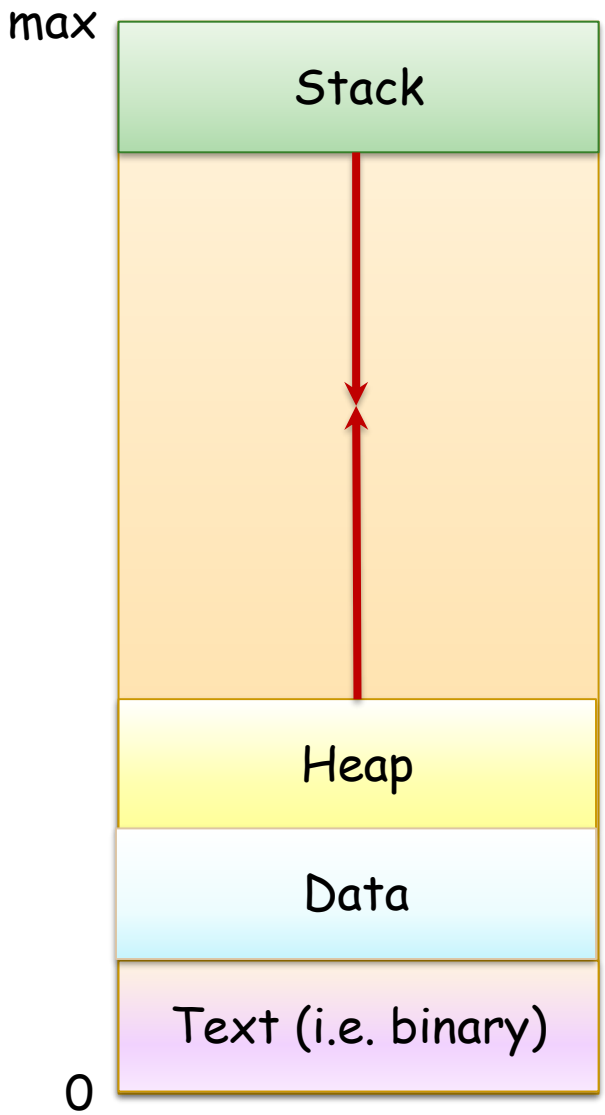
Data



Process Memory: Example

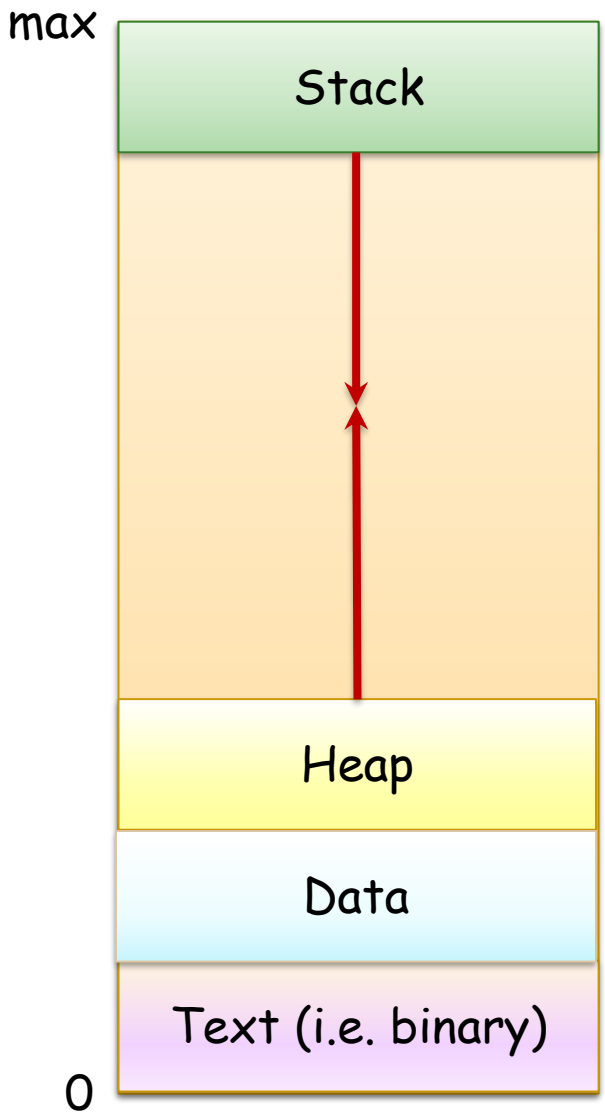
```
int a = 1;
int f(int c) { int b = c+1; return b;}
int main()
{       int d = 2;
        char* arr = new char[100];
        for(int i = 0; i < 100; ++i)
                arr[i] = '\0';
        f(d);
        delete arr;
        return 0;
}
```

Stack



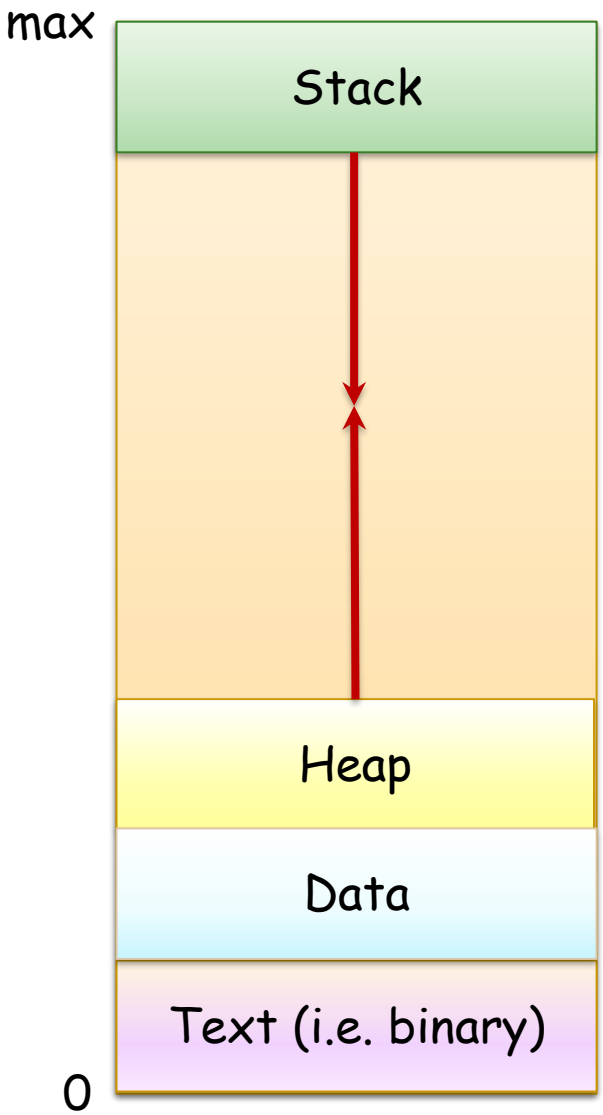
Process Memory: Example

```
int a = 1;
int f(int c) { int b = c+1; return b;}
int main()
{       int d = 2;                          Heap
        char* arr = new char[100];
        for(int i = 0; i < 100; ++i)
                arr[i] = '\0';
        f(d);
        delete arr;
        return 0;
}
```



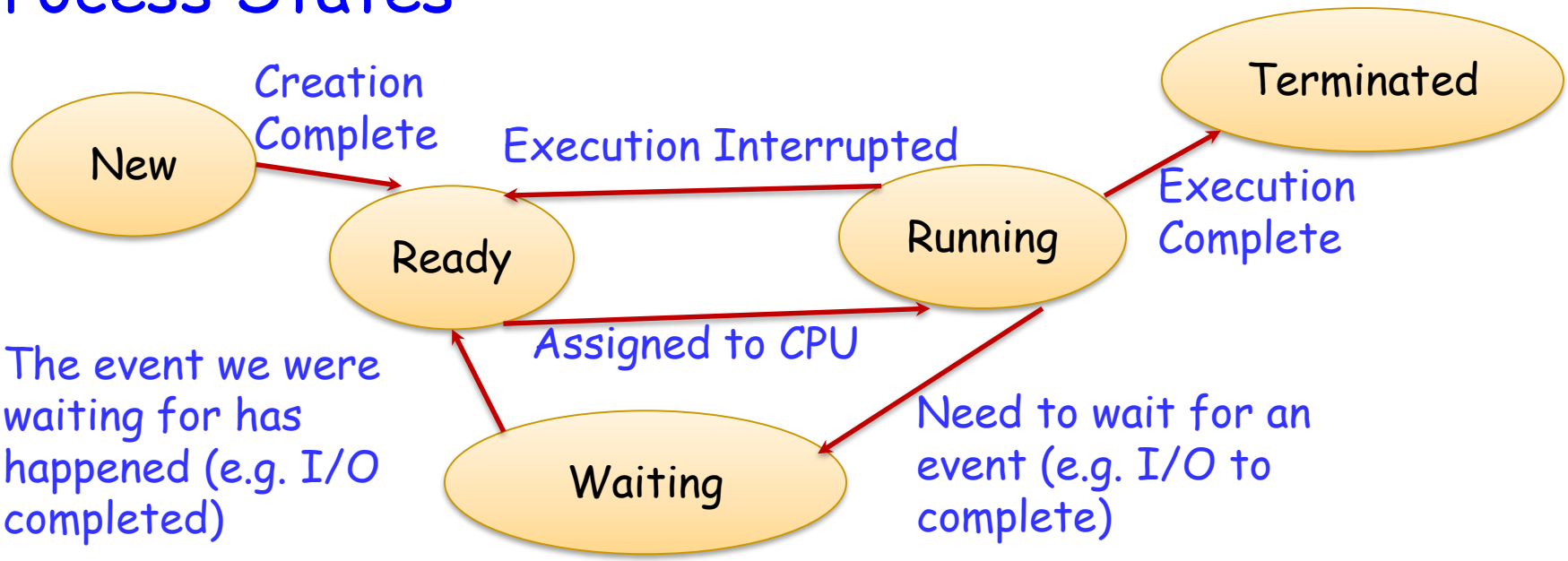
Process Memory: Example

```
int a = 1;
int f(int c) { int b = c+1; return b;}
int main()
{       int d = 2;
        char* arr = new char[100];
        for(int i = 0; i < 100; ++i)
                arr[i] = '\0';
        f(d);
        delete arr;
        return 0;
}
```



Text (i.e., executable instructions)

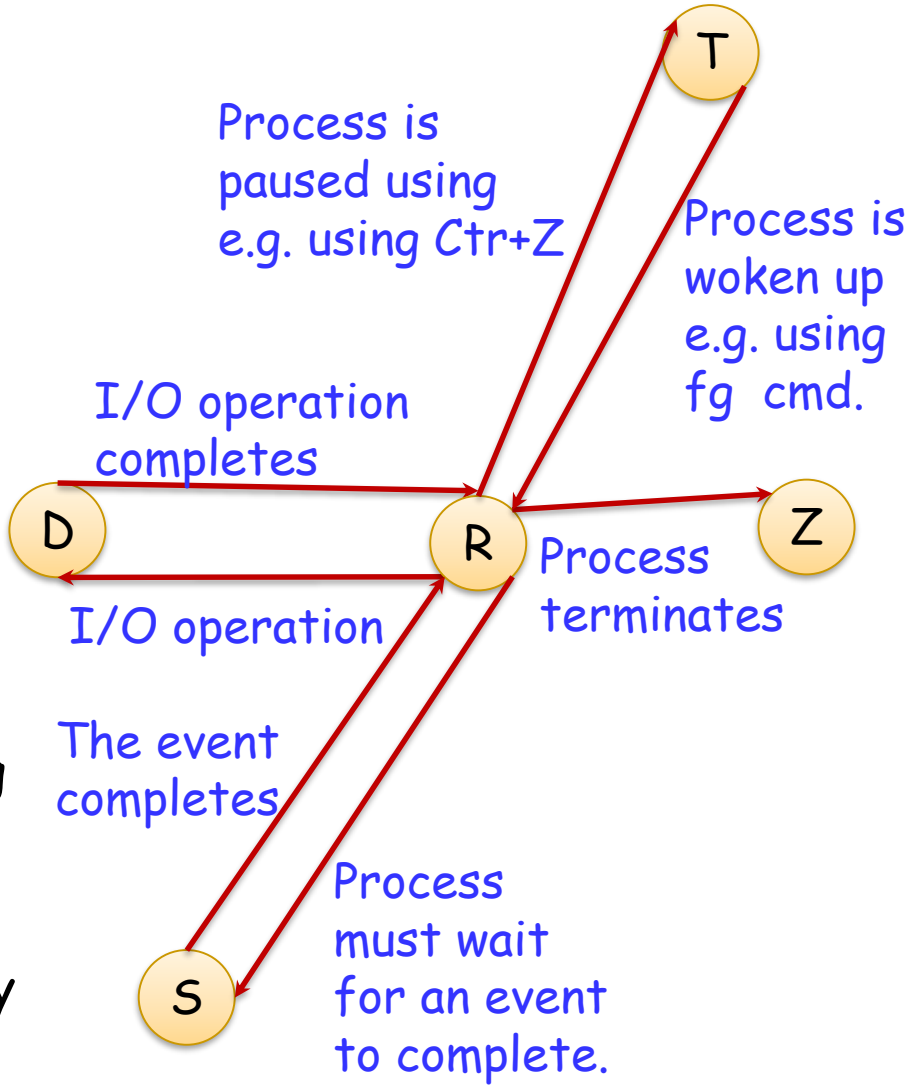
Process States



- The states of a process:
  - New: Process being created.
  - Running: Instructions are being executed.
  - Waiting: process is waiting for some event (e.g., I/O completion).
  - Ready: process waiting to be assigned to a processor.
  - Terminated: the process has finished execution.

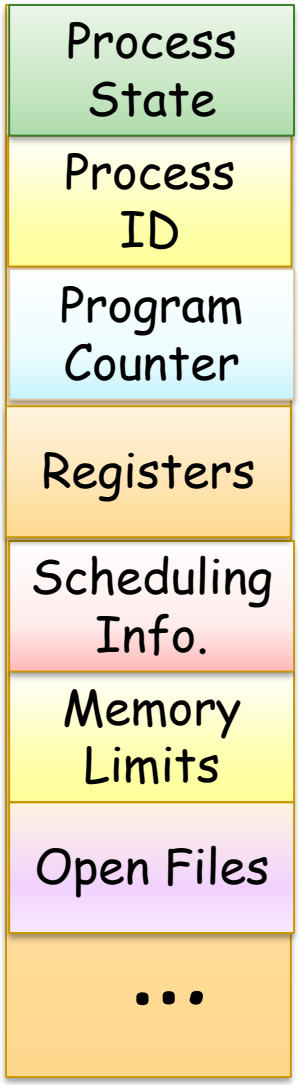
Process States in Linux

- Can learn process state using the ps utility:
  - Example: ps –aux
  - Interpreting the output of ps:
    - **R** the process is running or runnable (on run queue).
    - **D** uninterruptible sleep (usually I/O)
    - **S** interruptible sleep (waiting for an event to complete)
    - **Z** defunct/zombie, terminated but not reaped by its parent (discussed later).
    - **T** stopped, either by a job control signal or because it is being traced



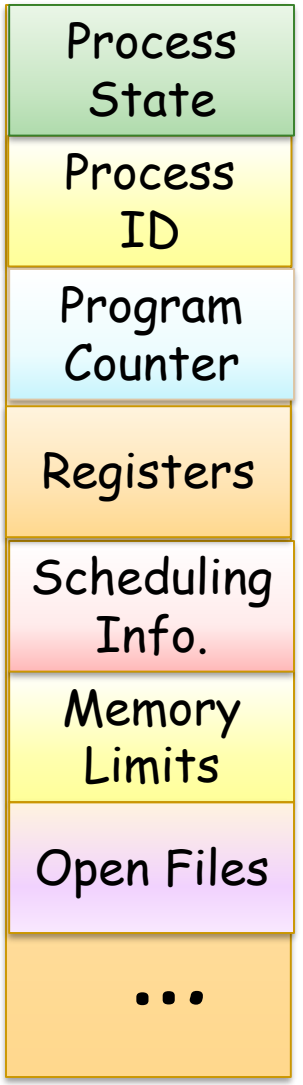
Process Control Block

- To represent each process, the OS uses a Process Control Block (PCB):
- PCB Components:
  - Process State: the state of the process e.g. Running.
  - Process ID: a unique ID associated with the process e.g. 123.
  - Program Counter: the address of the next instruction to be executed.
  - CPU Registers: the current values of the accumulators, stack pointers, etc.
  - CPU-Scheduling Information: e.g. priority and other info. needed for assigning the process to the CPU.



Process Control Block

- PCB Components (Contd):
  - Memory Management Information: info. about memory belonging to the process.
  - Accounting Information: e.g. how long the process was running etc.
  - I/O Status Information: list of I/O devices used by the process, list of open files, etc.
- Example: in Linux OS, PCB is represented using struct task_struct which contains:
  - struct mm_struct mm – which stores memory information.
  - struct files_struct *files – list of open files
  - and much more.



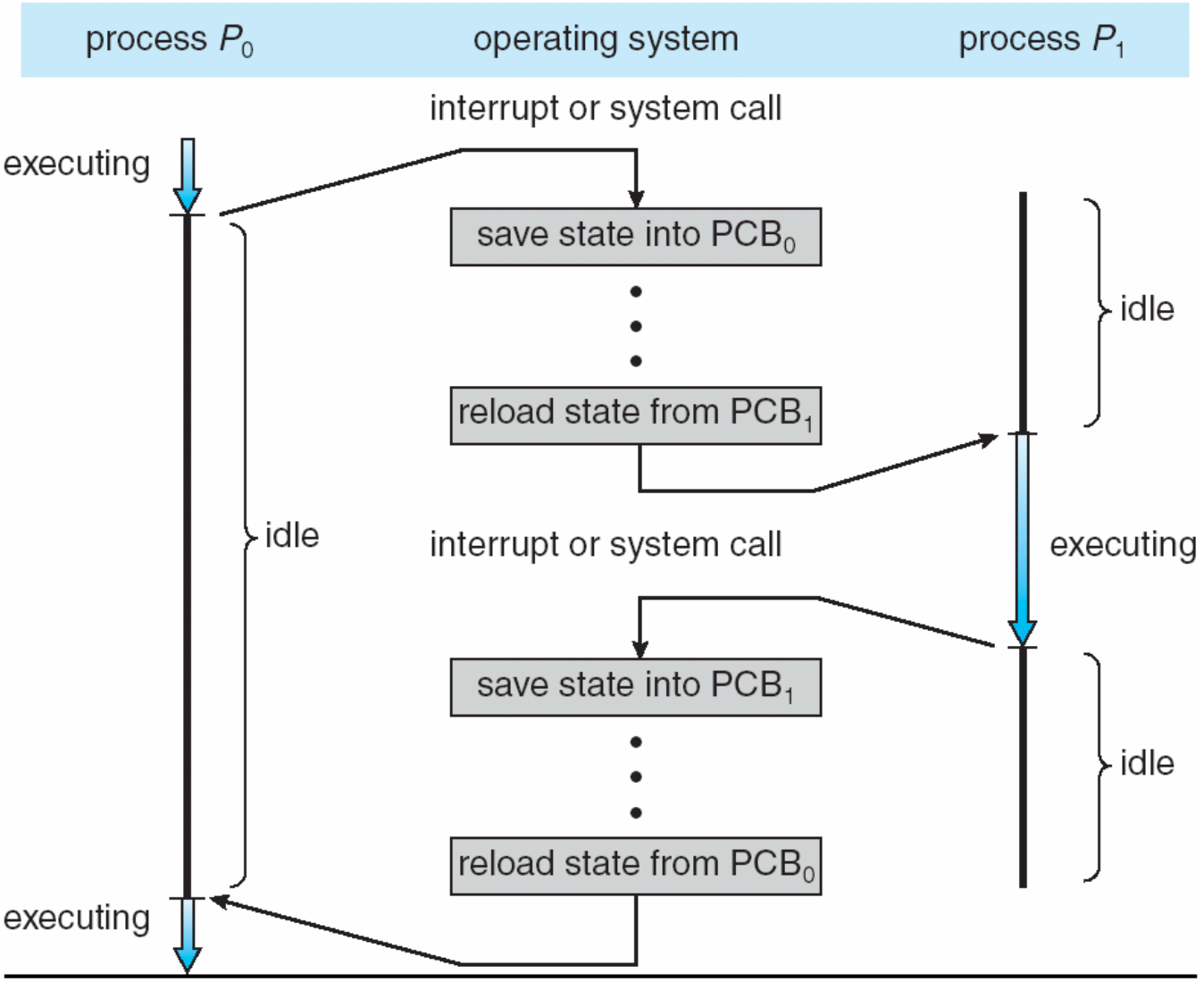
Student Participation - Understanding the process concept via a student library analogy

- A process can be compared to the act of reading a book checked out by a student from the reserved books section of a library.

- 1) The student corresponds to the _____.
- 2) The book corresponds to the _____.
- 3) The library reading room corresponds to the _____.
- 4) The librarian corresponds to the _____.
- 5) The check-out record corresponds to the _____.

- *Select from: CPU, Memory, Program, PCB, OS*

Process Control Block: Usage

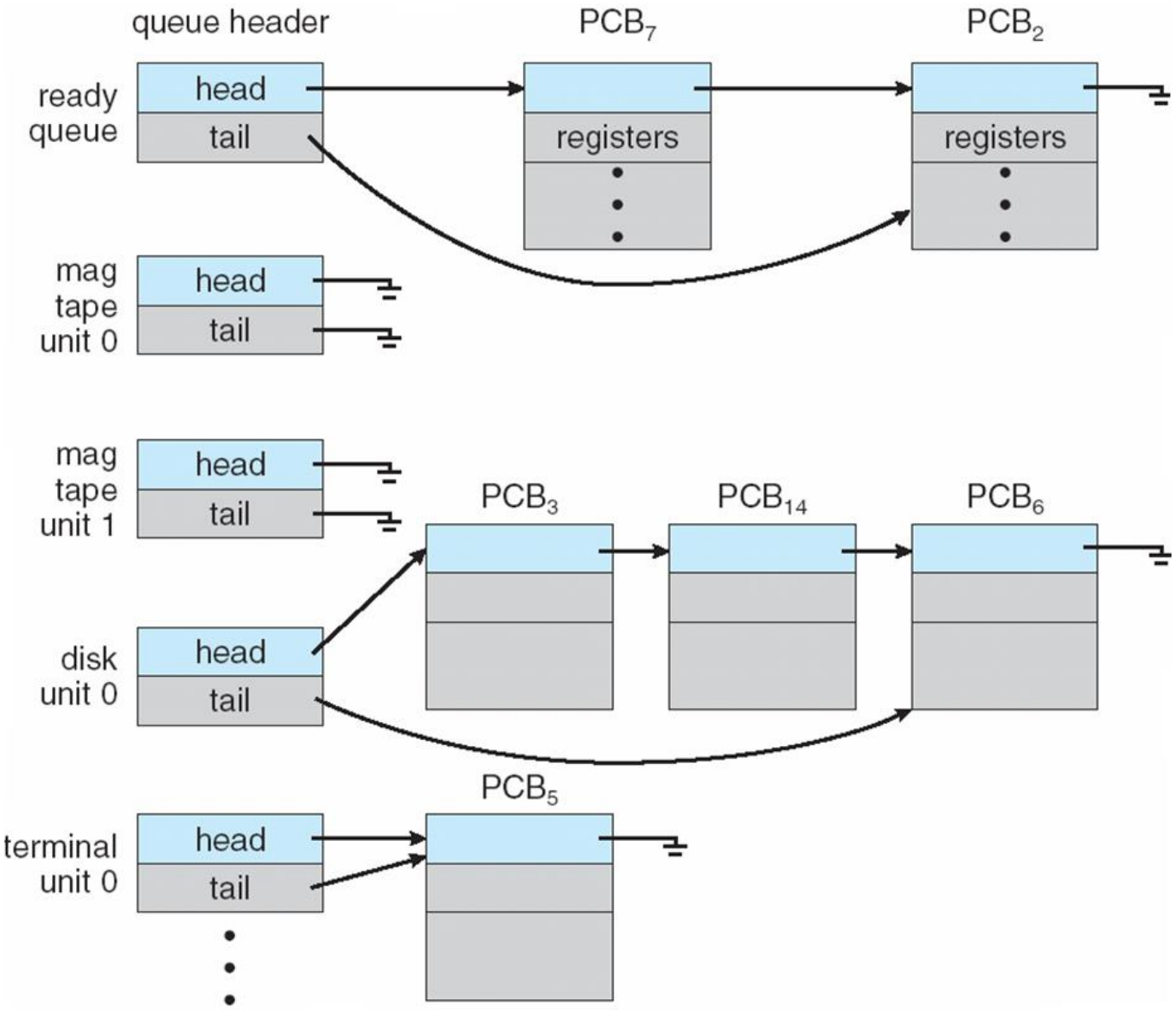


Process Scheduling: The Queuing Mechanism

- Objective of multiprogramming (i.e., supporting multiple processes): have some process running at all times, to increase CPU utilization.
- Solution: use a process scheduler: decides which available processes get the CPU.
- Process scheduler implementation consists of:
  - Job queue: contains all processes on the system.
  - Ready queue: contains processes that are in the main memory and are ready to execute.
  - Device queue (1 per device): contains all processes waiting to use a particular device.

Process Scheduling: The Queuing Mechanism

- Example of queues:

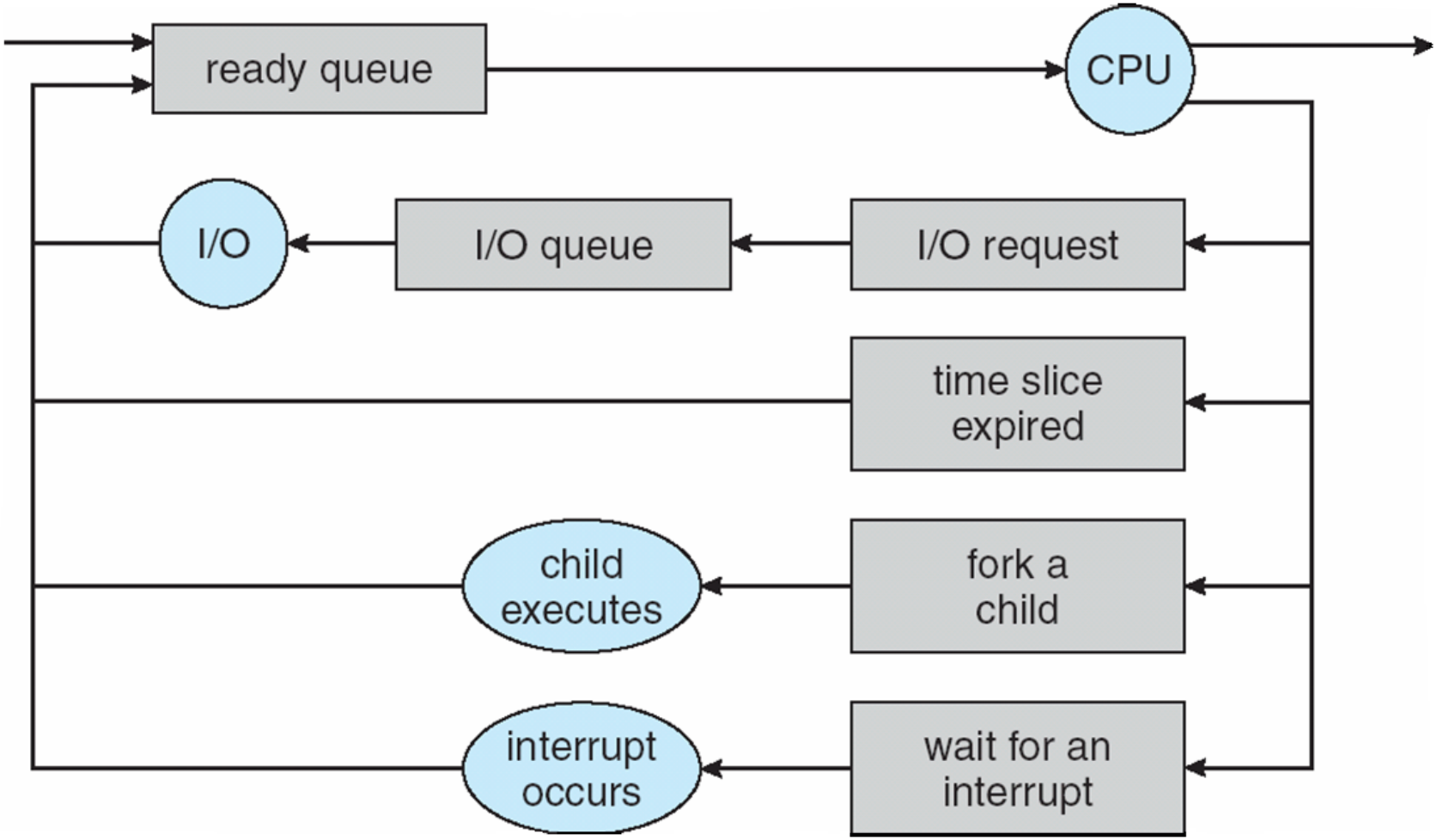


Process Scheduling: The Queuing Mechanism in Action

- 1. New process is placed on a ready queue.
- 2. The process is dispatched i.e., selected for execution.
- 3. During execution:
  - Process may be queued on the device queue due to an I/O request.
  - Process can create a new subprocess and wait for its termination.
  - Process can be interrupted and removed from the CPU.
  - Process time slice (i.e., time allotted for its CPU use) expires:
    - Process is removed from the CPU, and
    - is placed on the the ready queue until scheduled to run for another time slice.

Process Scheduling: The Queuing Mechanism in Action

- Boxes = queues
- Circles = actions



Process Scheduling: Schedulers

- Assigning processes to queues and selecting processes for execution, is the job of a process scheduler.
- Types of schedulers:
  - Short-term
  - Long-term
  - Medium-term

Process Scheduling: Schedulers: Short-term

- Short-term scheduler: decides which process in memory gets the CPU.
  - Invoked very frequently (often at least once every 100 msecs)
  - Has to be efficient.

Process Scheduling: Schedulers: Long-term

- Question: What if we have more processes than can fit into memory?
- Answer: no problem! Spool (i.e., temporarily store) some processes on a mass storage device (e.g., hard drive).
- Long-term scheduler: selects spooled processes to load from the mass storage device into main memory.
  - Executes less frequently than a short-term scheduler.

Process Scheduling: Schedulers: Long-term

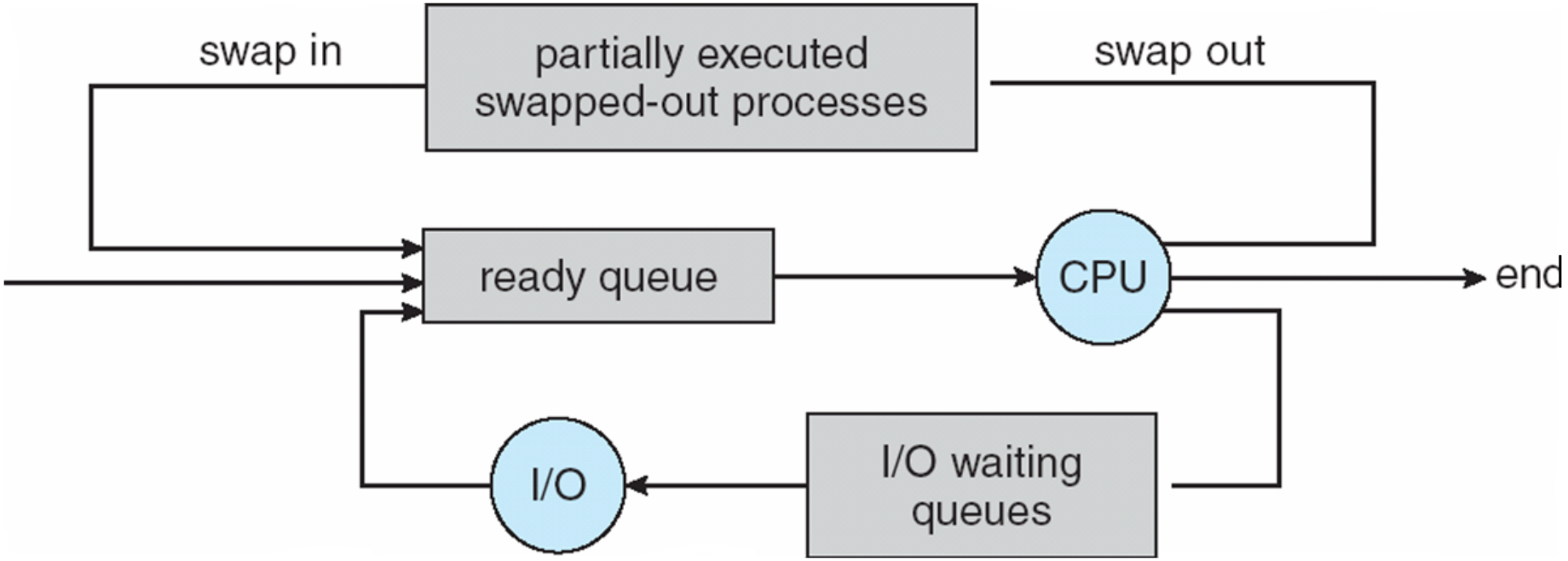
- Key idea: maximize resource utilization by selecting a mix of CPU bound and I/O bound processes:
  - I/O bound processes: processes that spend more time doing I/O operations than CPU computations.
  - CPU bound processes: processes that spend more time doing CPU computations than I/O operations.
- Why is mixing important?
  - If all selected processes are I/O-bound, then ready queue will always be empty i.e., nothing to execute on the CPU!
  - If all selected processes are CPU-bound, the device queues will be empty i.e., the devices go unused.
- Not present on all systems: e.g., Windows and Unix.

Process Scheduling: Schedulers

- Medium-term scheduler:
  - Removes some processes from memory in order to improve the process mix.
  - Later, the removed processes can be brought back into memory.
  - The job of the medium-term scheduler is a.k.a. swapping.

Process Scheduling: Schedulers

- Medium-term scheduler in action:



Process Scheduling: Context Switching

- Context switch: switching CPU between processes:
- context switch is the process of storing the state of a process or of a thread, so that it can be restored and execution resumed from the **same point** later. This allows multiple processes to **share** a **single** CPU, and is an essential feature of a multitasking operating system
  - 1. Save CPU state of the currently executing process into a PCB.
  - 2. Select another process.
  - 3. Use the saved PCB of the selected process to initialize the CPU.
  - 4. Let the selected process resume execution.
- Context switch time is pure overhead!

Student Participation - Who causes a context switch

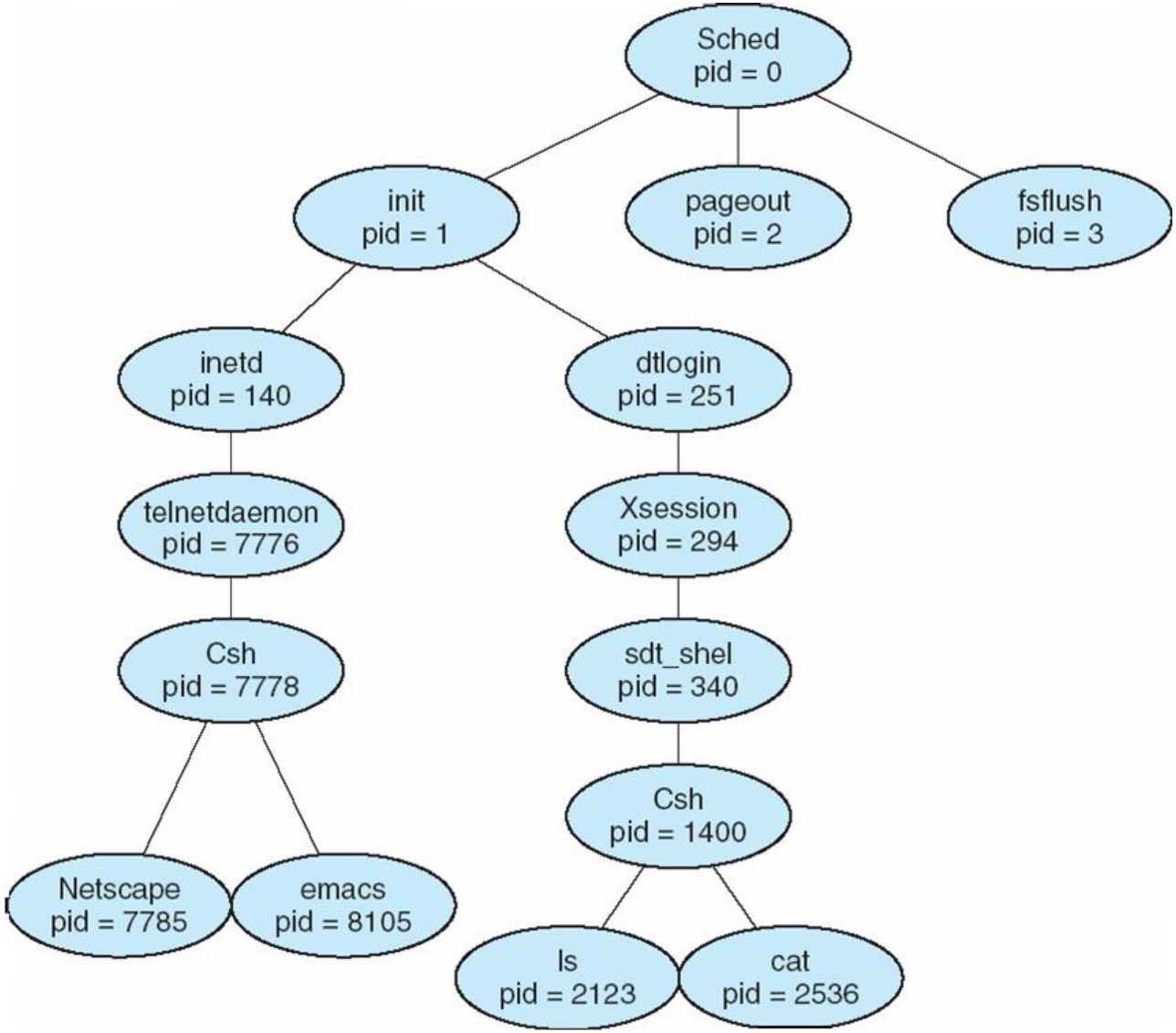
- A context switch may be caused by _____ .
- 1) the currently running process p
  - True
  - False
- 2) the OS
  - True
  - False
- 3) some other process q
  - True
  - False

Operations on Processes

- A process may create new processes by issuing a process creating system call i.e., asking the OS to create another process.
  - Parent process: the creator process.
  - Child process: the process created by the parent process.
  - Child processes can create their own child processes.
  - Process tree: a model of parent-child relationships.

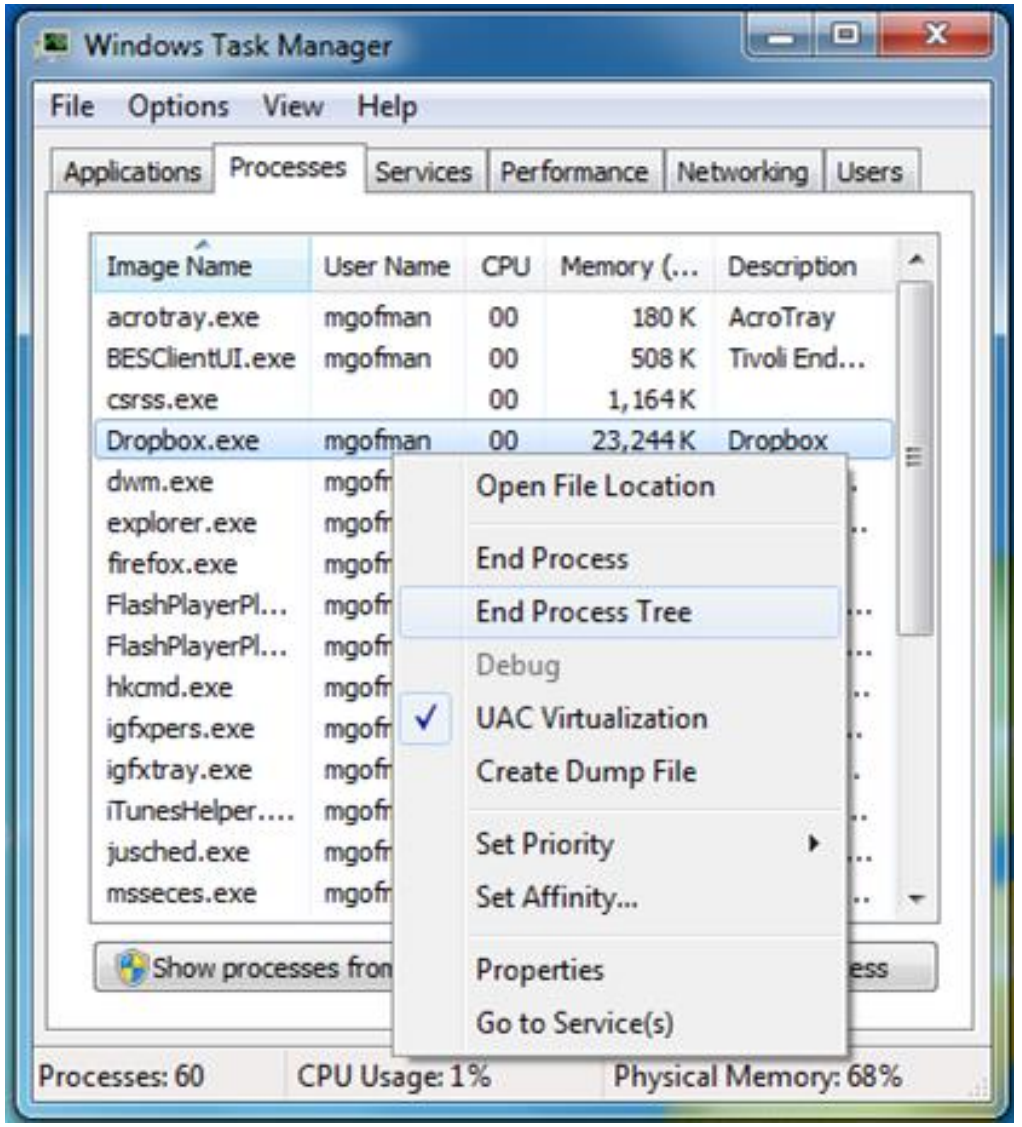
Operations on Processes: Process Tree Example

- Process tree of Solaris processes.



Operations on Processes: Process Tree Example

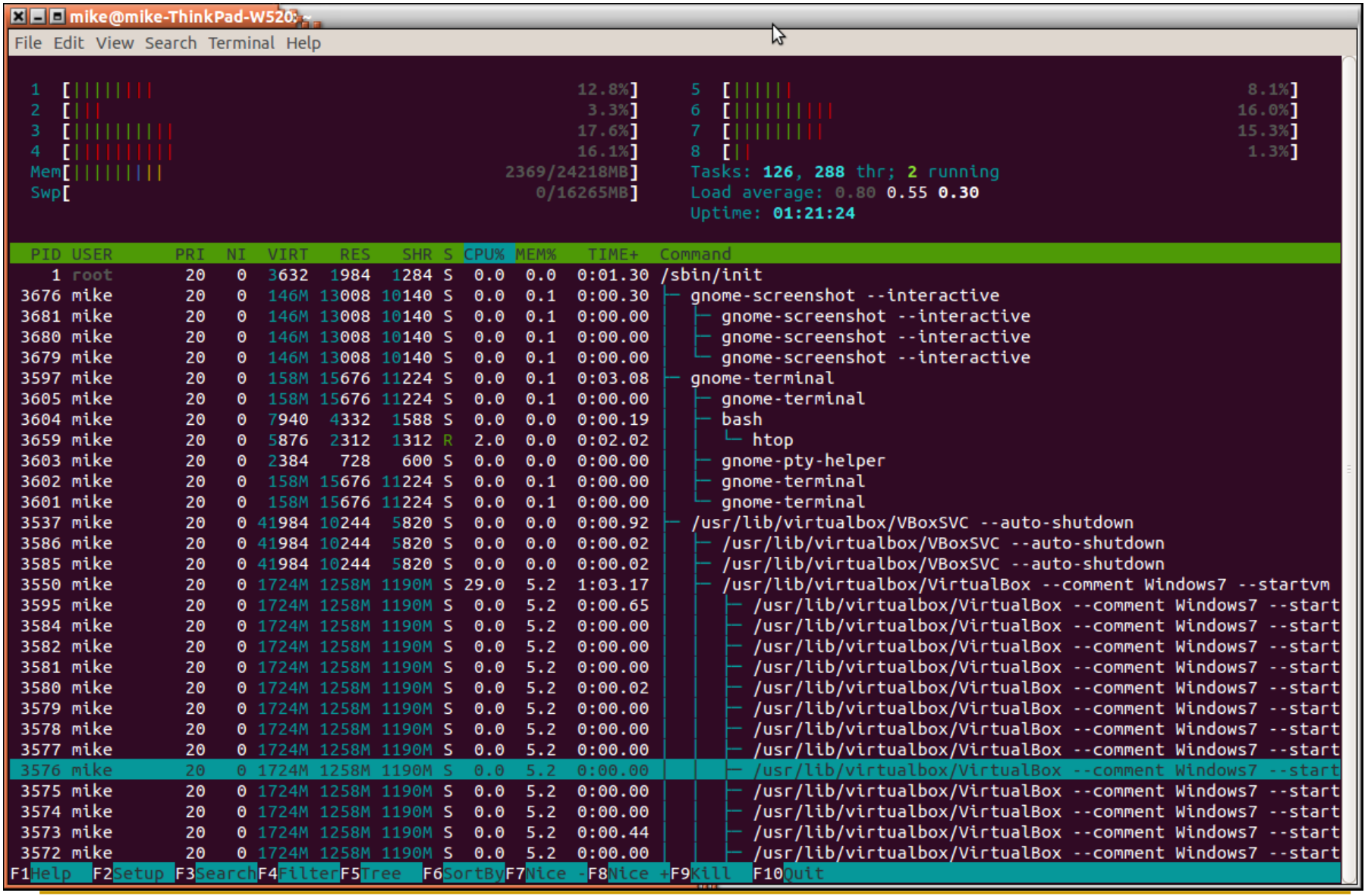
- Windows: "End Process Tree" terminates the selected process and all its descendants.



Shortcut: Ctrl + Shift + Esc

Operations on Processes: Process Tree Example

- Linux: process displayed using the "htop" program.



Operations on Processes: Parent-Child Relations

- After creating a child process a parent process may:
  - continue executing, or
  - wait for the child process to terminate.
- The child process can either:
  - be a duplicate of the parent process (i.e., has the same program and data), or
  - it may be running a new program.