Anthony Ventresque
anthony.ventresque@ucd.ie

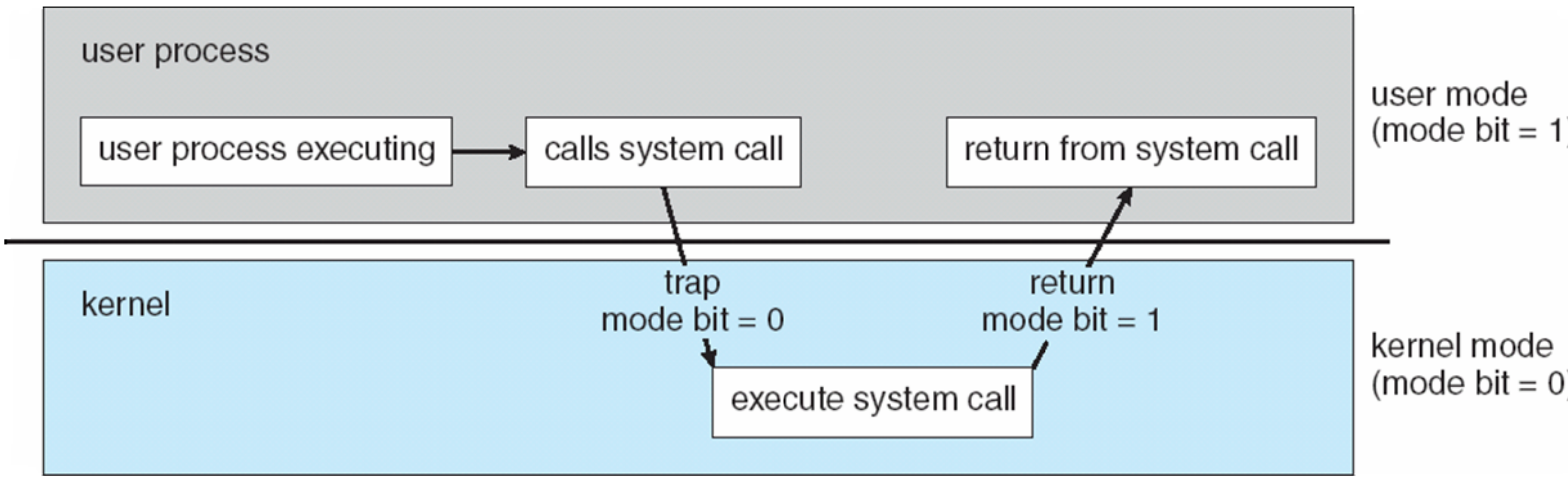
Operating Systems
COMP30640

Process Management I
Processes

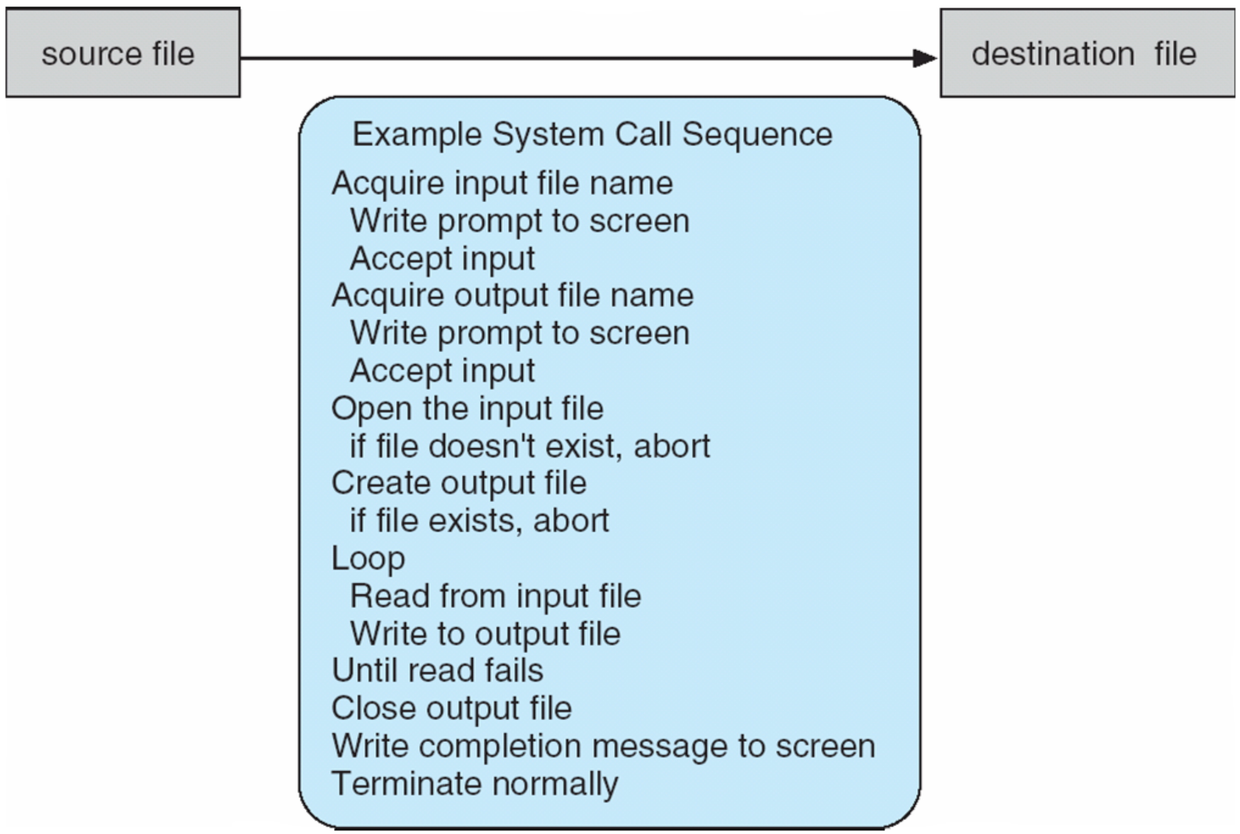
School of Computer Science, UCD

Scoil na Ríomheolaíochta, UCD

Last Week…


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

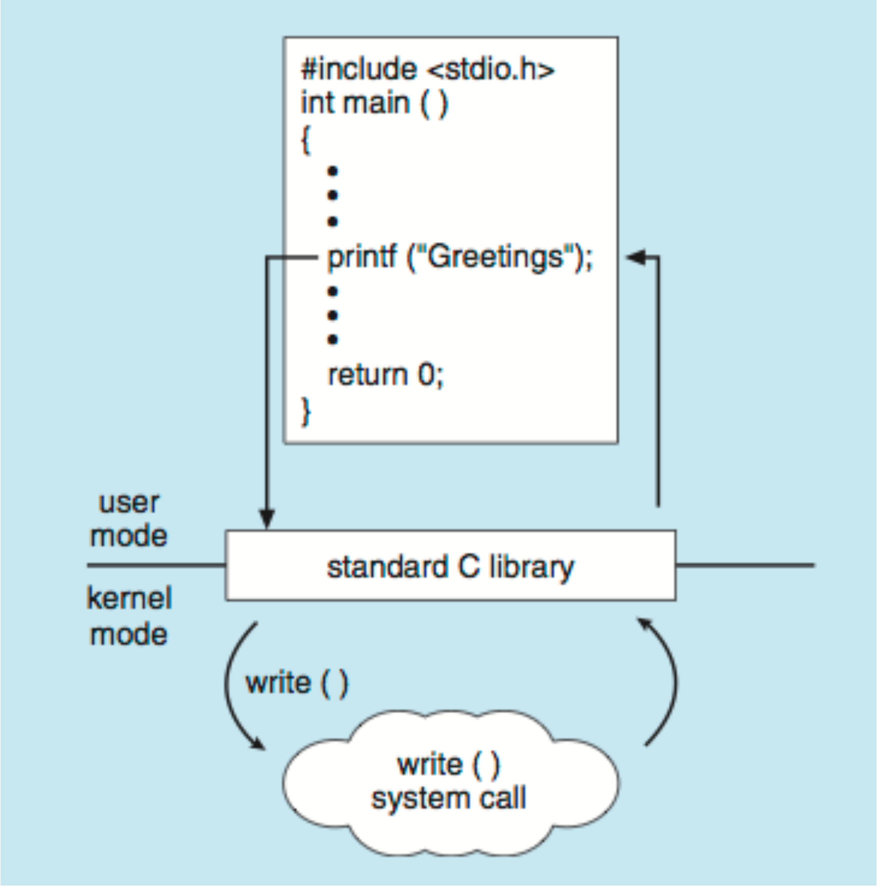
Last Week…



source file
destination file
Example System Call Sequence
Acquire input file name
Write prompt to screen
Accept input
Acquire output file name
Write prompt to screen
Accept input
Open the input file
if file doesn't exist, abort
Create output file
if file exists, abort
Loop
Read from input file
Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally

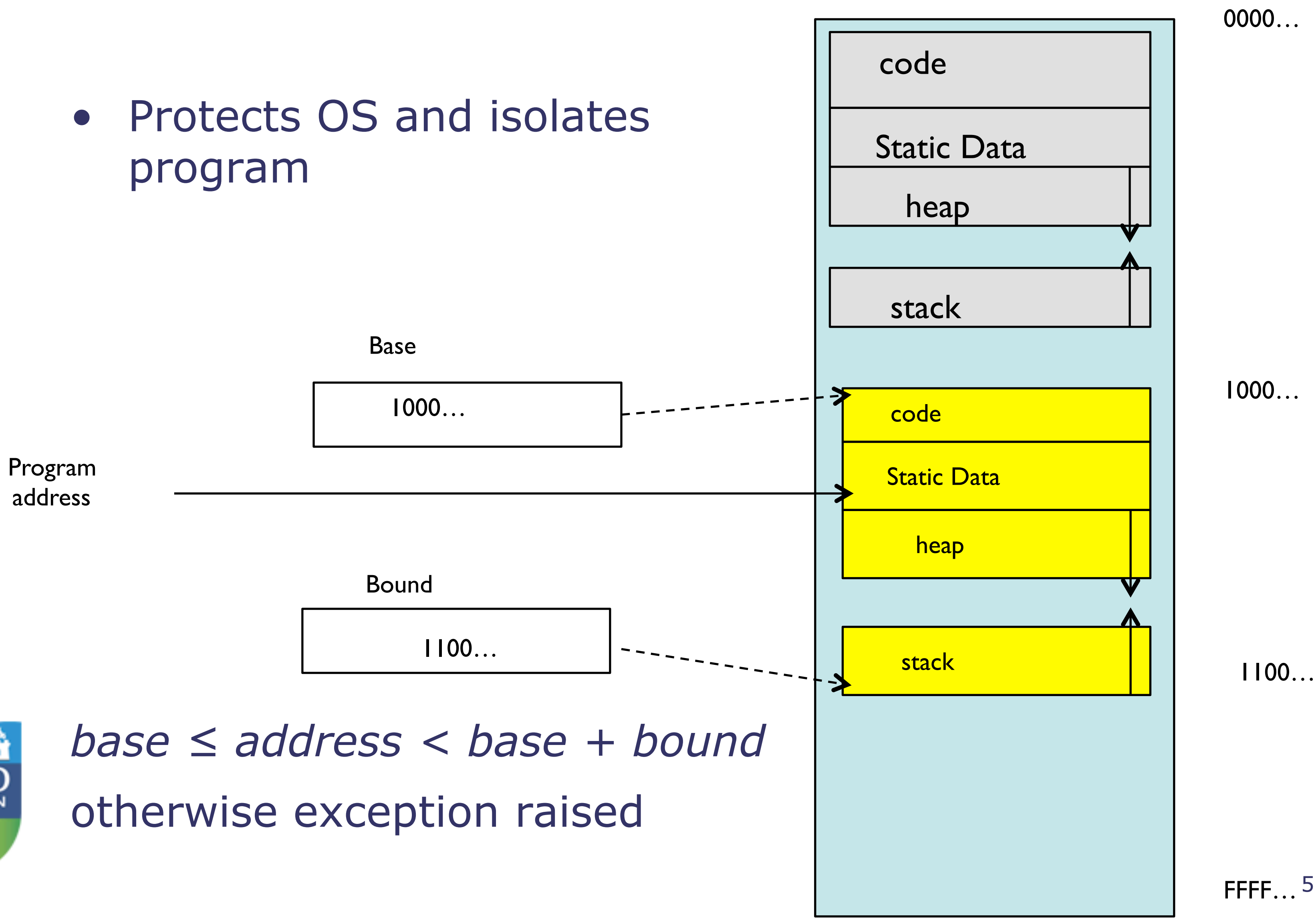
Last Week…

- C program invoking printf() library call, which calls write() system call



Last Week…

- Protects OS and isolates program



base ≤ address < base + bound

otherwise exception raised

Last Week…

- Apart from protecting memory and I/O, we must ensure that the OS always maintains control
  - A user program might get stuck into an infinite loop and never return control to the OS
- ***Timer***: Generates an interrupt after a fixed or variable amount of execution time
  - OS may choose to treat the interrupt as a fatal error (and stop program execution) or allocate more execution time
  - note: with time-sharing a timer interrupt is periodically generated in order to schedule a new process

Last Week…

- A script bash is a bash “text” in a text file
  - First line always contains the following:

    #!/bin/bash
  - a bash program needs to be made executable:

    chmod u+x my_script.sh
  - the .sh extension is just a convention
  - to execute the script:

    ./my_script.sh [arg1 arg2 ...]

Outline

- What is a Process (Process/Program)
- Program Control Block
- Process features:
  - Process Switching
  - Process Creation and Termination
  - Communication between Processes

Take home message:

A process is a program in execution, which forms the basis of all computation

OS Activities

- Early computer systems executed one program at a time
- All modern OS execute many kinds of activities concurrently (basis of multiprogramming)
  - user programs
  - batch jobs and command scripts
  - system programs
- Each "execution entity" is encapsulated in a process
  - in different OSs also called: task, job, actor,. . .
- The OS takes care of most aspects concerning processes:
  - it creates, deletes, suspends and resumes processes
  - it schedules & manages processes (resources, IPC, etc.)

Program vs. Process(s)

- Program is ***passive*** entity stored on disk (***executable file***), process is ***active***
  - Program becomes process when executable file loaded into memory

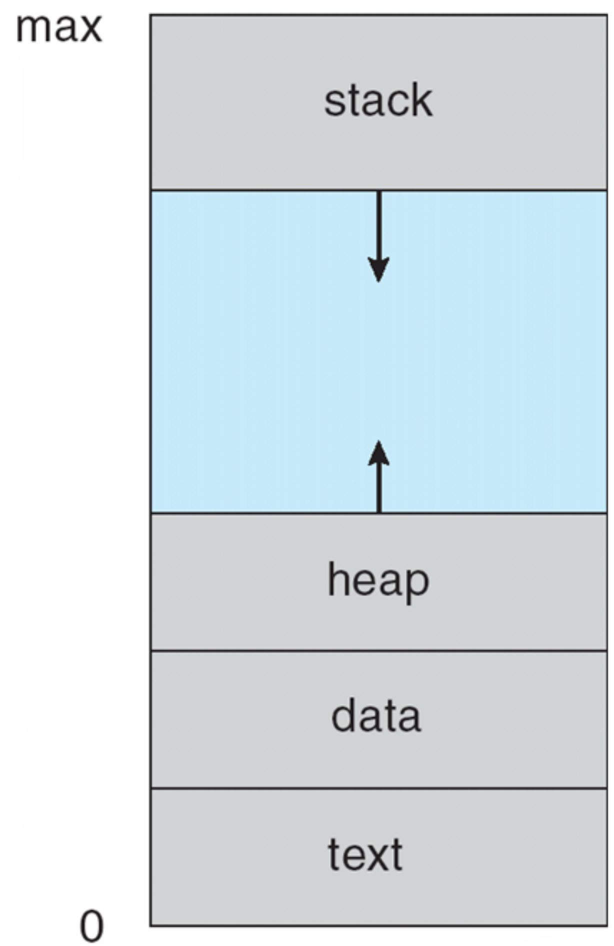
- ***A process is one instance of a program in execution***
  - A process requires a section of main memory to run
  - At any time, there may be more than one process running a different instance of the same program
    - e.g. several processes running the same editor
  - One program can spawn several processes

Process Abstraction

- ***Process***: an instance of a program, running with limited rights
  - ***Thread***: a sequence of instructions within a process
    - Potentially many threads per process (for now 1:1)
  - ***Address space***: set of rights of a process
    - Memory that the process can access
    - Other permissions the process has (e.g., which system calls it can make, what files it can access)

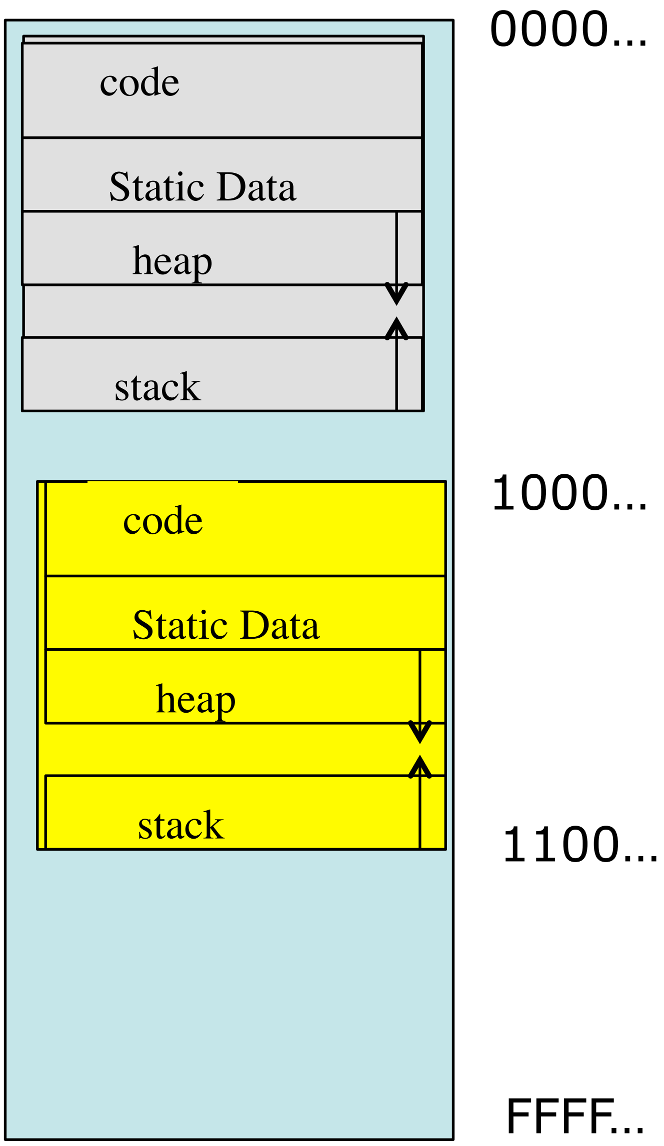
Process Concept

- A process includes all information needed to run its program. Process address space (main memory section), divided into:
  - ***Text region*** (executable code or program)
  - ***Data region*** (global variables)
  - ***Heap*** (containing memory dynamically allocated during run time)
  - ***Stack region*** (return address and local variables for active procedure calls)



Process Concept

- ***Each process runs in its own address space***
  - The address space of two different processes translates into two different ***physical*** addresses in main memory
  - A process cannot directly read or write memory addresses belonging to another process
  - Example: with base and bound registers, a process' address space goes from ***base to bound*** (which are different for all processes)

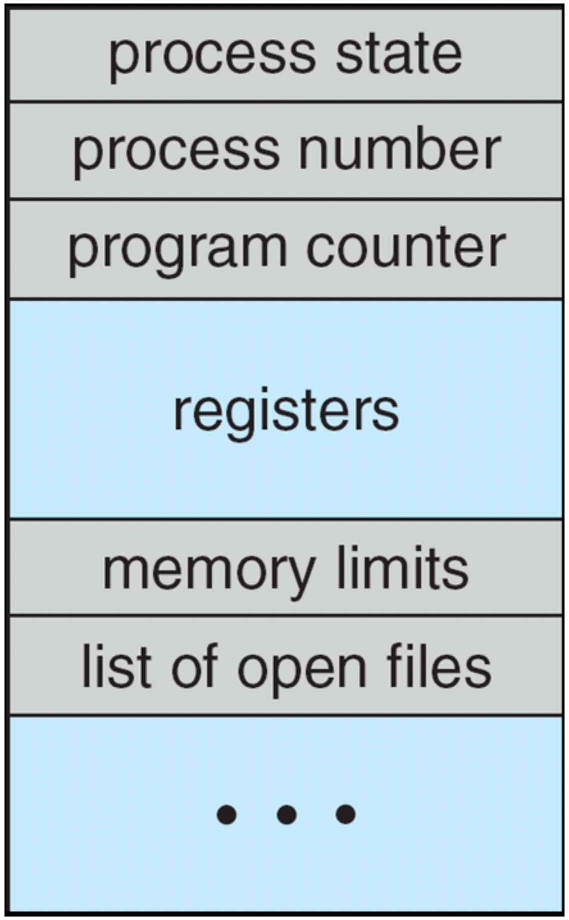


Process States and PCB

- The states a process may be in are either active or suspended
- ***Active*** states of a process:
  - ***Running***: using a processor to execute instructions
  - ***Ready***: executable, but all system processors (CPUs) are currently in use; usually, many more processes than processors in a system
  - ***Blocked***: waiting for an event to occur
- The process state is represented by a dynamic data structure called ***process control block (PCB)***, or process descriptor
  - PCB describes the status of all resources used by a process
  - PCB contains critical info. and must be kept in memory protected from user access
- Kernel Scheduler maintains data structure containing PCBs (***Process Table***)
  - Scheduling algorithm selects the next one to run
  - each entry contains a PID and a pointer giving the address of that process's PCB in memory

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 execute next
- ***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



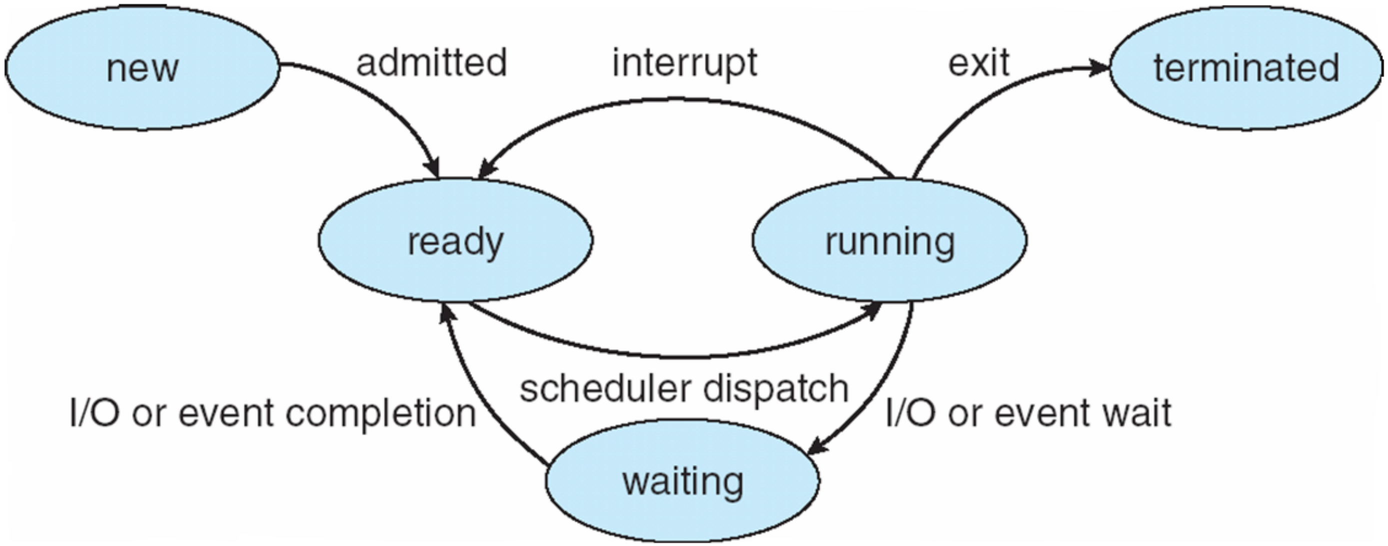
Process Operations

- A number of operations can be performed to create/ delete a PCB (and thus a process), or to change process information in the PCB
  - ***create***: creates and initialises a new PCB
  - ***delete*** (***destroy***, ***kill***): removes a process from the system
  - ***wait*** (***block***): execution ceases until a specific event occurs
  - ***signal***: indicates that a specific event has occurred
    - event: a relevant change in the status of an entity that a process requires in order to execute
    - two examples of I/O events:
      - event triggering a wait operation: “write file”
      - event completion, triggering a signal operation: “write file complete” (hardware interrupt)

Process Operations (ii)

- Operations in the PCB (con’td)
  - ***schedule*** (***dispatch***): assigns ready process to CPU
  - ***change priority***: alters the scheduling priority of a process
    - schedule is usually triggered by some system change altering the decision on which process should be currently running
    - it depends on the scheduling algorithm used by the OS
    - a time-out can trigger a schedule operation
- Operations involving suspended states:
  - ***suspend*** (***sleep***): suspends an active process
  - ***resume*** (***wake up***): places a suspended process into blocked or ready state
    - on being suspended, a process is removed from contention for time on a processor (i.e., contention for being in running state), without being actually deleted

Process State Diagram (Life Cycle of a Process)



- As a process executes, it changes state
  - ***New***: The process is being created
  - ***Ready***: The process is waiting to run
  - ***Running***: Instructions are being executed
  - ***Waiting***: Process waiting for some event to occur
  - ***Terminated***: The process has finished execution

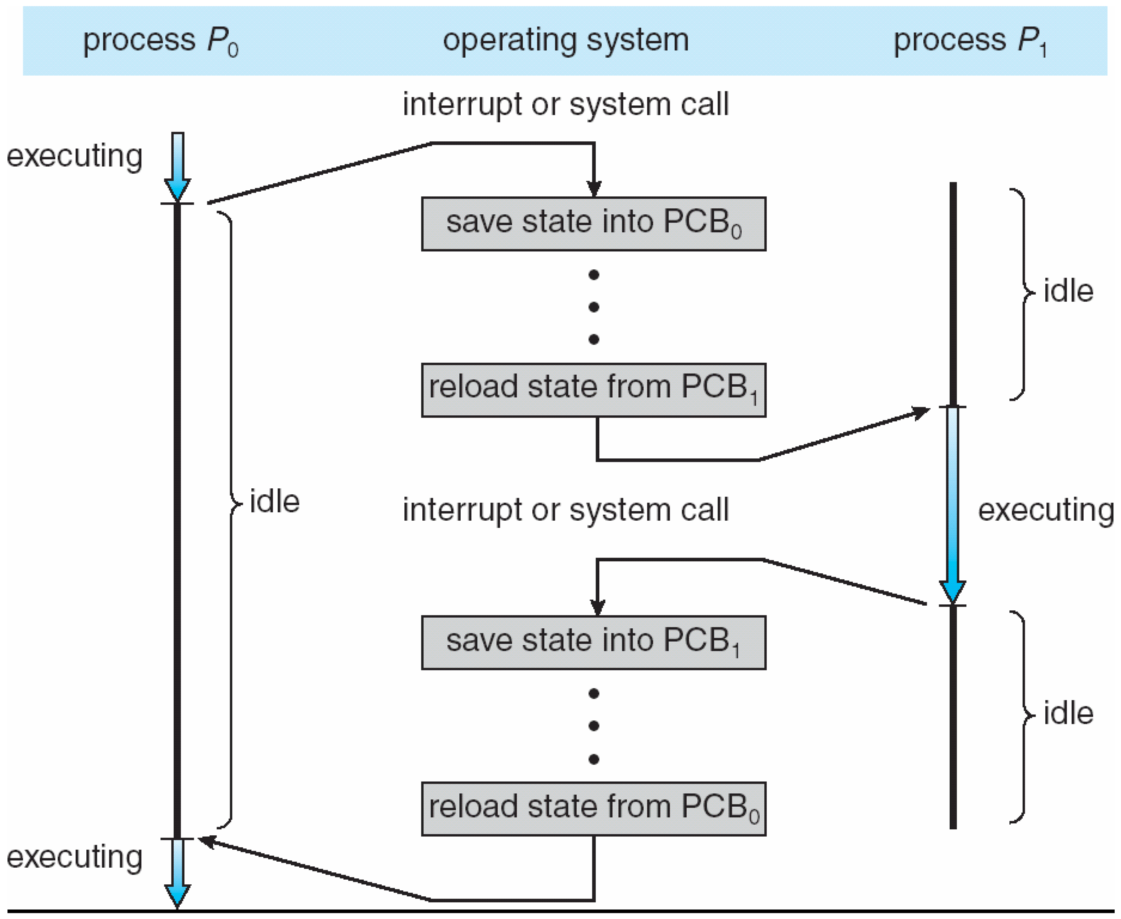
Context Switch

- A ***context switch*** happens when a running process stops execution, and another process (initially in the ready state) starts execution
  - CPU Switch From Process to Process

- Dispatching the ready process requires that the system:
  - Saves the state of the running process in its PCB
  - Loads the saved state from the ready process' PCB

- Context-switch time is ***overhead***. The system does not do useful work during a context switch; therefore:
  - Context switches should be quick (***hardware dependent***)
  - Number of context switches per unit time should be minimised (***software dependent***); however, consider interactivity
  - Overhead sets minimum practical switching time

Context Switch



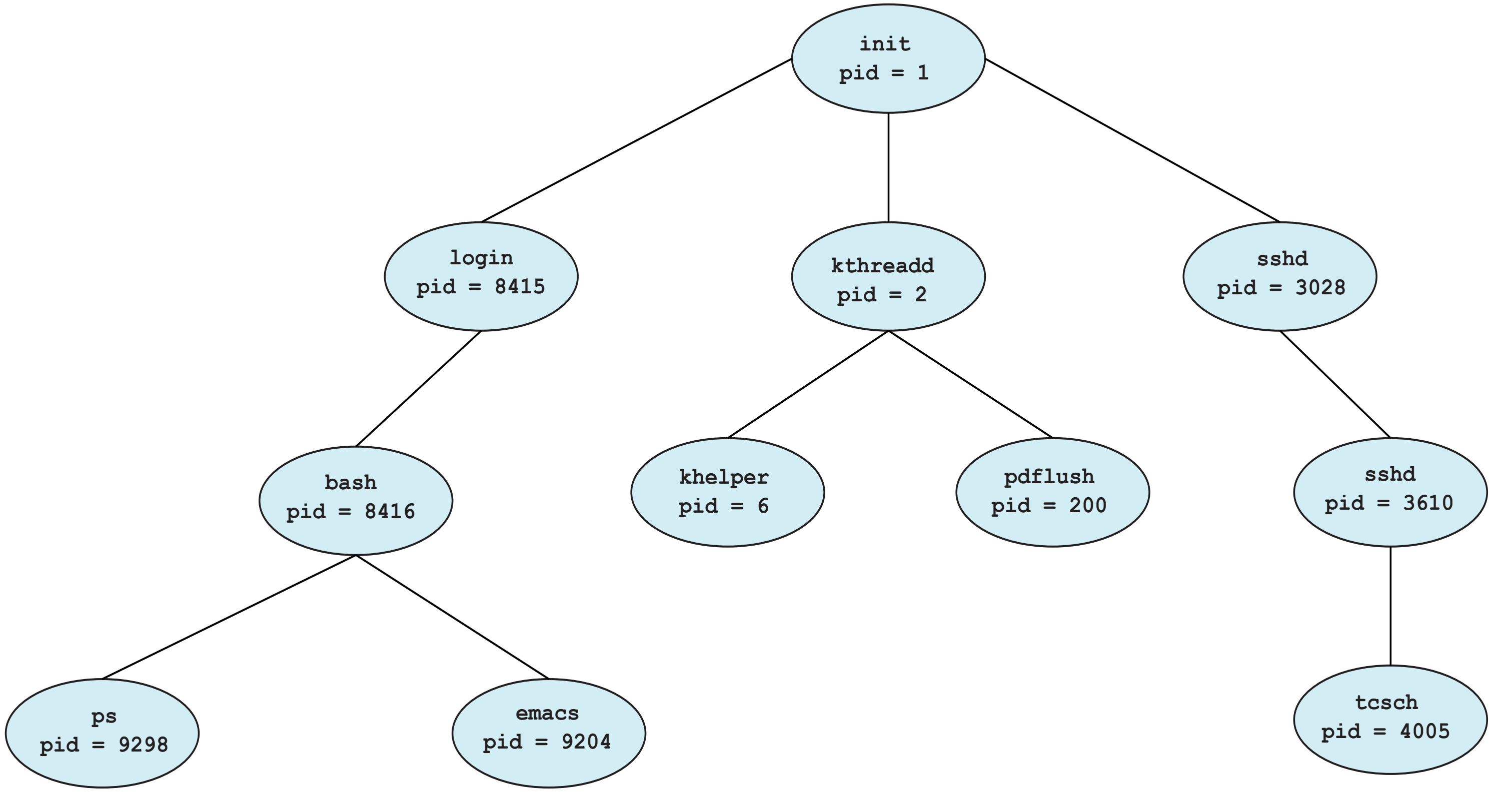
Process Creation and Termination

- Processes are ***created*** by two main events:
  - System boot
  - Execution of process creation system call by another process
    - examples: user request to create another process, initiation of a batch job, . . .
- Processes are ***terminated*** due to
  - Voluntary conditions: normal exit, error exit
  - Involuntary conditions: fatal error, killed by another process
- Creation and termination ***states***:
  - A created process goes to the ready or suspended-ready states (the very first process can go to the the running state directly)
  - Voluntary termination is from the running state
  - Involuntary termination may happen in any state

Process Creation

- ***Parent*** process create ***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



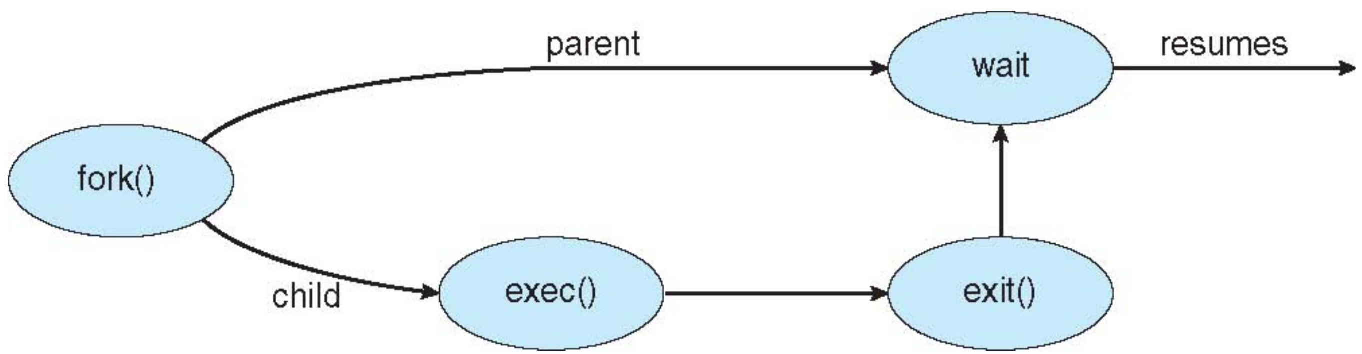
pstree

Documents — anthony@hibernia: ~ — ssh hibernia.ucd.ie — 64×22

```
anthony@hibernia:~$ pstree
init─┬─NetworkManager───2*[{NetworkManager}]
     ├─acpid
     ├─apache2───10*[apache2]
     ├─at-spi-bus-laun─┬─dbus-daemon
     │                 └─3*[{at-spi-bus-laun}]
     ├─at-spi2-registr───{at-spi2-registr}
     ├─atd
     ├─avahi-daemon───avahi-daemon
     ├─bluetoothd
     ├─console-kit-dae───64*[{console-kit-dae}]
     ├─cron
     ├─cupsd
     ├─2*[dbus-daemon]
     ├─dconf-service───2*[{dconf-service}]
     ├─dhclient
     ├─dnsmasq
     ├─fail2ban-server───2*[{fail2ban-server}]
     ├─gconfd-2
     ├─6*[getty]
     ├─gnome-screensav───2*[{gnome-screensav}]
     ├─gvfsd───{gvfsd}
```

Process Creation (cont’d)

- ***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



Process in an OS

The OS manages for the process:

- ***Resources***
  - I/O operations: open, close, read, write,. . .
  - memory operations: sbrk (used by malloc and free)
- ***IPC***
  - global data: shared memory segments (shmget, shmop,. . .)
  - message-based communications: pipes, sockets, streams
  - synchronisation: semaphores, etc. (semget, semop)
  - (note: examples above are Unix system calls)

- Process is the elementary resource sharing agent in an OS

Interprocess Communication (IPC)

Two basic IPC models:

- ***Shared memory***
  - A shared memory region usually resides in the space address of the process creating a shared segment
  - Fast (only creating and attaching a shared segment are system calls)
  - but: data format and access is controlled by the processes sharing the segment, not the OS (thus conflicts are possible)
- ***Message passing***
  - Messages are sent by one process and received by another via a buffer/mailbox/port in kernel space
  - Slower (send/receive operations are system calls)
  - but: processes never share memory and thus do not need to trust each other

Conclusion

- An instance of an executing program is a process consisting of an address space and one or more threads of control
- As a process executes it changes state (***new, ready, running, waiting, terminated***) and transitions between states with operations (***create, delete (destroy, kill), wait (block), signal, schedule (dispatch), change priority, suspend (sleep), resume (wake up)***)
- ***Parent*** process create children processes, which, in turn create other processes, forming a ***tree*** of processes
- Interprocess communication can take place using shared memory and message passing