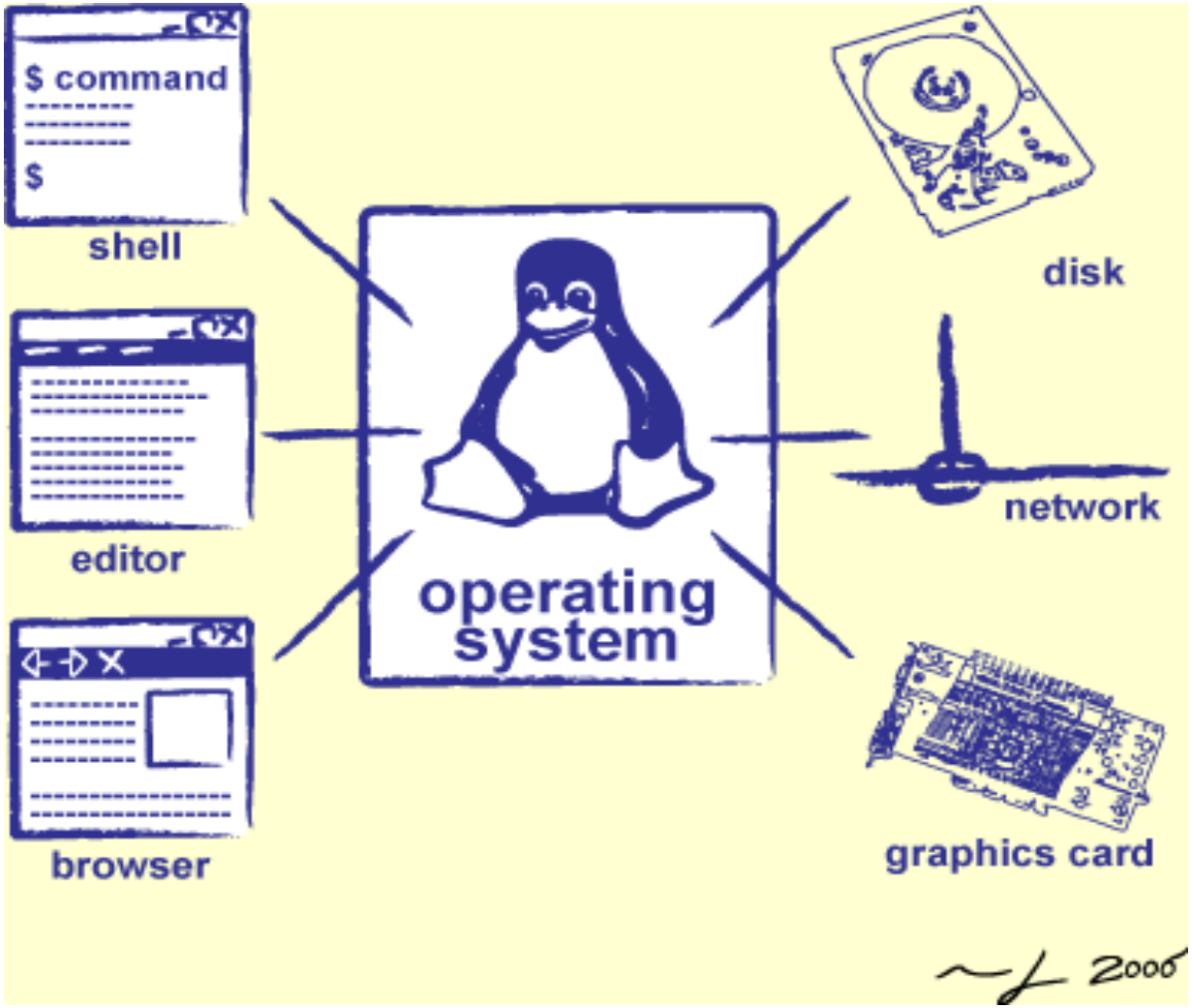
UNIVERSITY of STIRLING

# Concurrent and Distributed Systems
# CSCU9V5

Andrea Bracciali

abb@cs.stir.ac.uk

4B86


$ command
$
shell
editor
browser
operating system
disk
network
graphics card
2006

# PART 1: processes

# Introduction

- About Processes
  - Execution of a program
  - Processes – an Operating System abstraction
  - Life cycle of a process
- Process Implementation
  - Parts of a process
  - Process Creation and Termination
- Cooperating Processes
  - Producer - Consumer Problem
  - Race Conditions

# About Processes

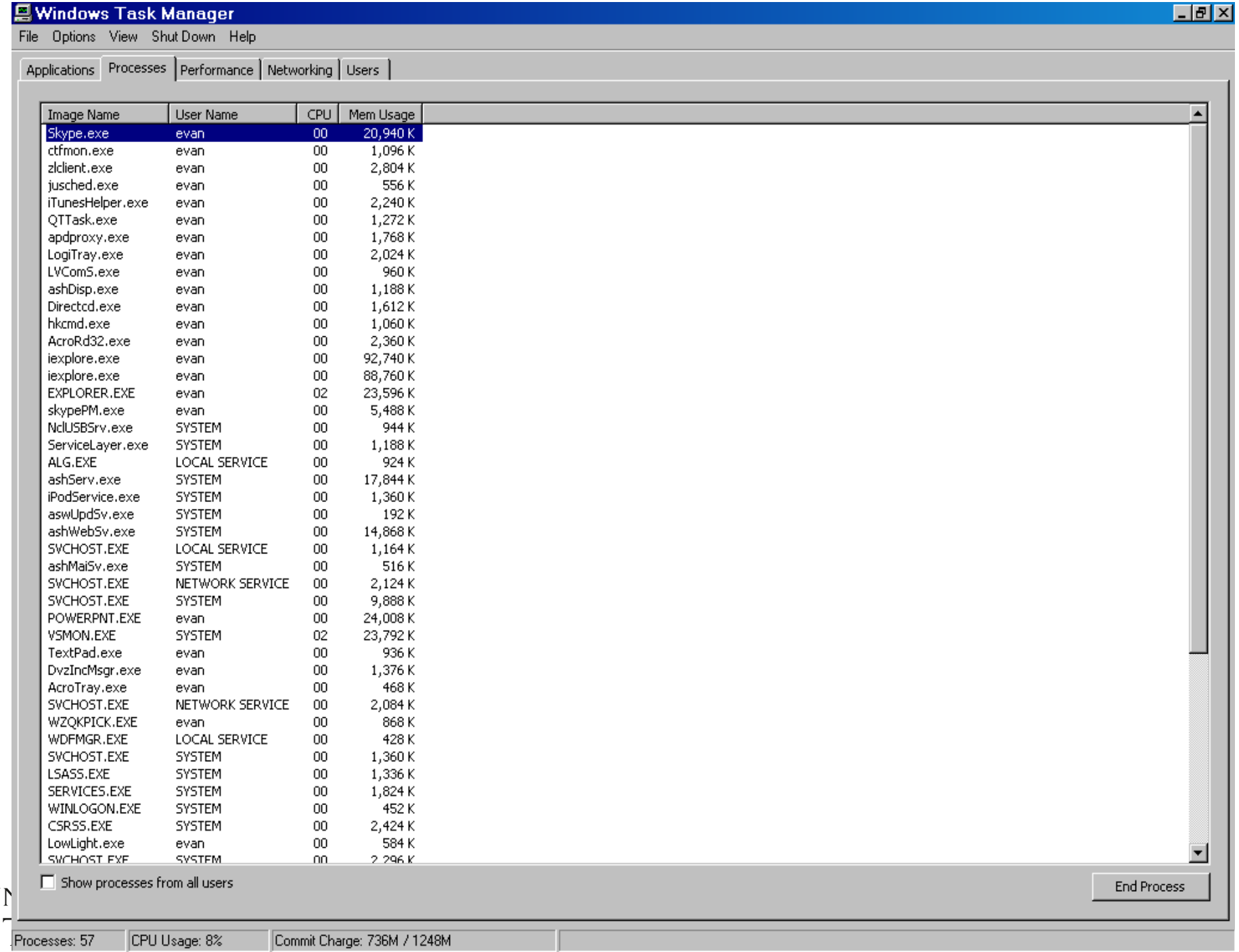
- Informally: a program in execution
- Computer execute predefined actions
  - Actions are specified by a program
  - Program is a *self-contained* entity
- Execution of a program requires resources
  - E.g., CPU
  - Running programs compete for CPU
  - Programs do not know when they get the CPU
- Actions of CPU and program logic are unrelated

# Program – Process Relationship

- A program is a static text that specifies a range of instructions which have to be executed by one or more processors (passive entity).

- A Process is an entity executing the range of instructions, which are specified by the program.

  Its thread of execution is somewhere in the middle between the first and the last of the actions specified by the program (active entity).


Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
Image Name User Name CPU Mem Usage
Skype.exe evan 00 20,940 K
ctfmon.exe evan 00 1,096 K
zlclient.exe evan 00 2,804 K
jusched.exe evan 00 556 K
iTunesHelper.exe evan 00 2,240 K
QTTask.exe evan 00 1,272 K
apdproxy.exe evan 00 1,768 K
LogiTray.exe evan 00 2,024 K
LVComS.exe evan 00 960 K
ashDisp.exe evan 00 1,188 K
Directcd.exe evan 00 1,612 K
hkcmd.exe evan 00 1,060 K
AcroRd32.exe evan 00 2,360 K
iexplore.exe evan 00 92,740 K
iexplore.exe evan 00 88,760 K
EXPLORER.EXE evan 02 23,596 K
skypePM.exe evan 00 5,488 K
NclUSBSrv.exe SYSTEM 00 944 K
ServiceLayer.exe SYSTEM 00 1,188 K
ALG.EXE LOCAL SERVICE 00 924 K
ashServ.exe SYSTEM 00 17,844 K
iPodService.exe SYSTEM 00 1,360 K
aswUpdSv.exe SYSTEM 00 192 K
ashWebSv.exe SYSTEM 00 14,868 K
SVCHOST.EXE LOCAL SERVICE 00 1,164 K
ashMaiSv.exe SYSTEM 00 516 K
SVCHOST.EXE NETWORK SERVICE 00 2,124 K
SVCHOST.EXE SYSTEM 00 9,888 K
POWERPNT.EXE evan 00 24,008 K
VSMON.EXE SYSTEM 02 23,792 K
TextPad.exe evan 00 936 K
DvzIncMsgr.exe evan 00 1,376 K
AcroTray.exe evan 00 468 K
SVCHOST.EXE NETWORK SERVICE 00 2,084 K
WZQKPICK.EXE evan 00 868 K
WDFMGR.EXE LOCAL SERVICE 00 428 K
SVCHOST.EXE SYSTEM 00 1,360 K
LSASS.EXE SYSTEM 00 1,336 K
SERVICES.EXE SYSTEM 00 1,824 K
WINLOGON.EXE SYSTEM 00 452 K
CSRSS.EXE SYSTEM 00 2,424 K
LowLight.exe evan 00 584 K
Show processes from all users
End Process
Processes: 57
CPU Usage: 8%
Commit Charge: 736M / 1248M

# Processes Lifecycle

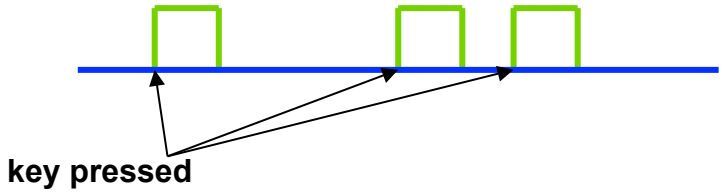
- Number of processes >> number of CPUs
- Some processes cannot run
- Processes waiting for resources
  - CPU (‘ready processes’)
  - Other resources (‘blocked processes’)
- Operating System provides for liveliness
  - State transitions of a process

# Example: Multitasking

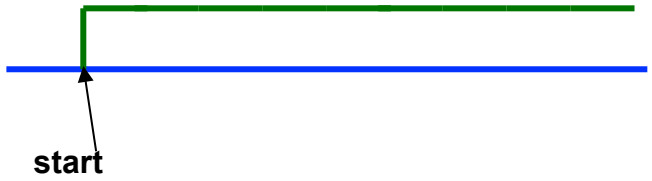
- The OS virtual machine can (in effect) run several programs at the "same" time.

- Efficient usage of CPU:

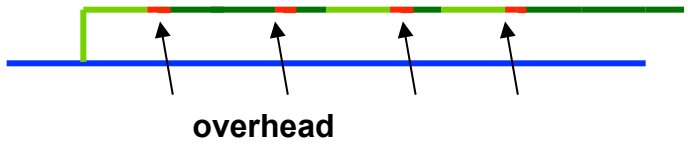
I/O bound process (editor):




CPU bound process (sci calc.):



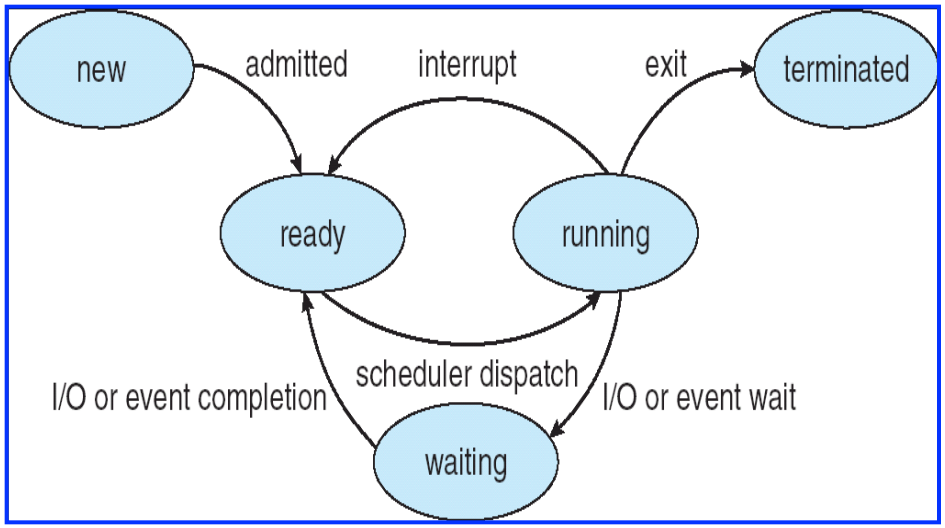

Efficient sharing of CPU:



# Process States

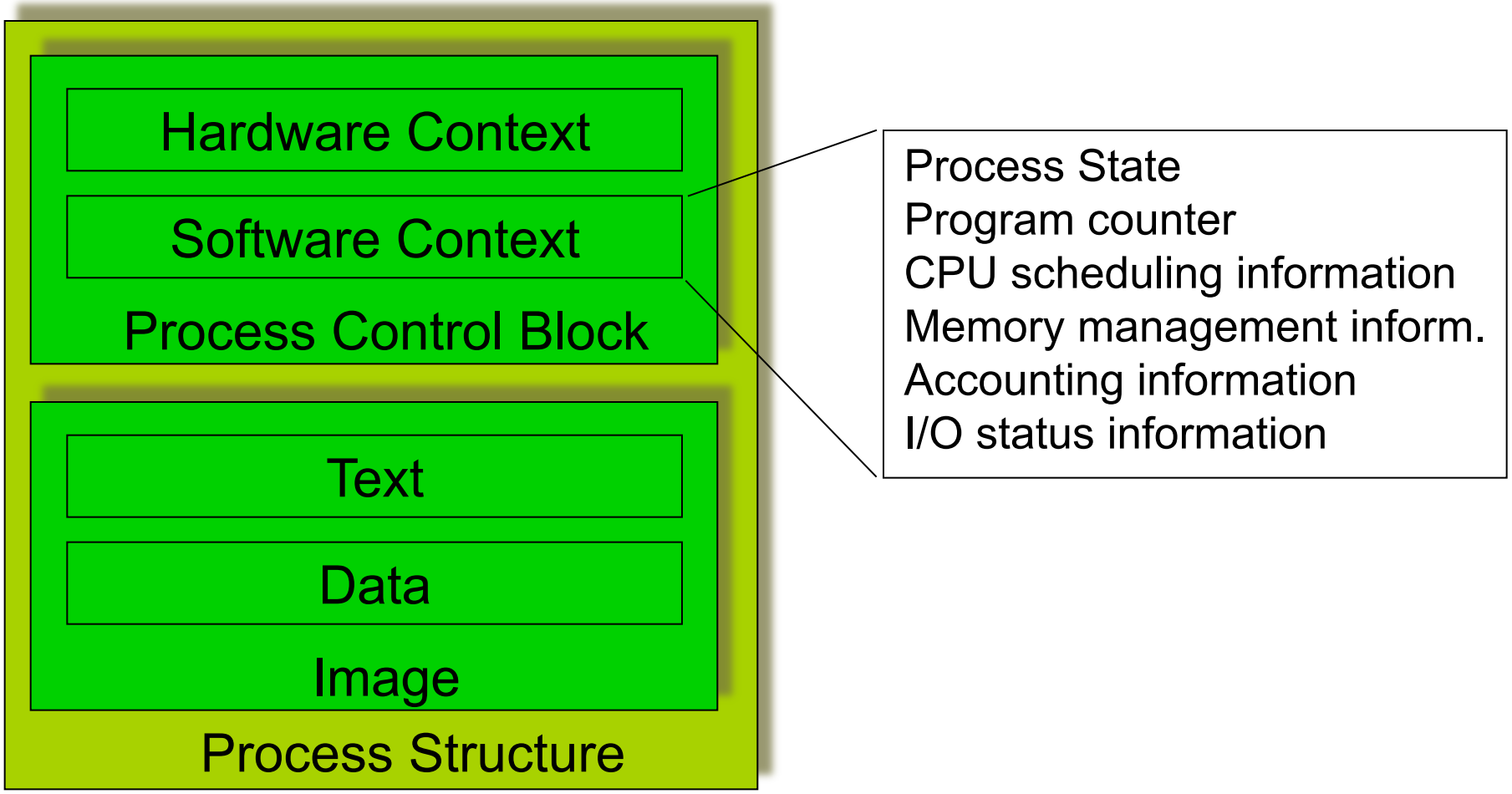
- As a process executes, it changes state
  - initiated: The process is being created.
  - running: Instructions are being executed.
  - waiting: The process is waiting for an event to occur.
  - ready: The process is waiting to be assigned.
  - terminated: The process has finished execution.

# Process Lifecycle

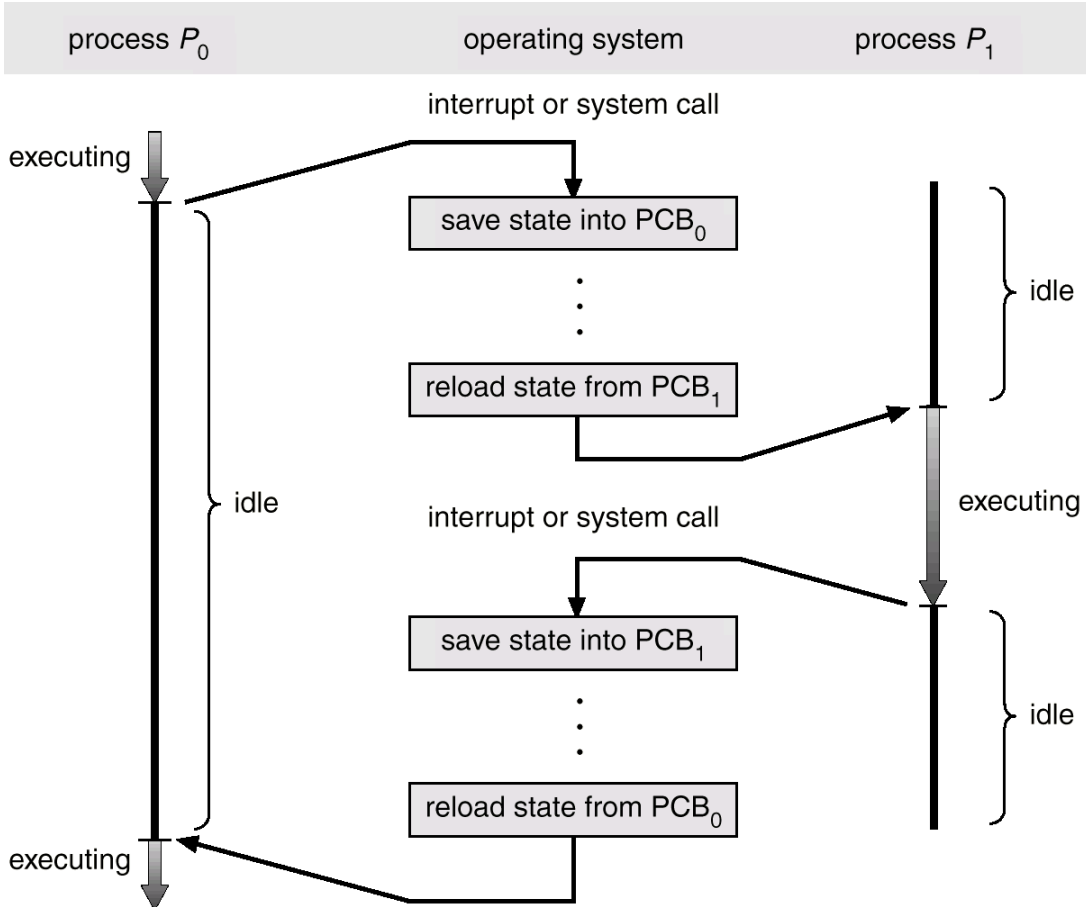



- **new** process are created (a program is launched) and put in the **ready** queue (list)
- then the use of the CPU is assigned to a ready process, which becomes **running**
- a running process can either **terminate**, suspend while **waiting** for some event to occur or be forced to **release** the CPU (interrupted), for instance because it is being running for too long or because the OS has to load a new page in memory, and hence be put back to the ready queue
- a waiting process can eventually become **ready again**, when the event it was waiting for occurs.

# Parts of a Process



# 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.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

# Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.
- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.

# Example in c

```
#include <stdio.h>

void main(int argc, char *argv[])
    {
    int pid;

    /* fork another process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork
                                Failed\n");

        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
    /* parent waits for the child to
                                    complete */
        wait(NULL);
        printf("Child Complete\n");
        exit(0);
    }
}
```

# Process Termination

- Process executes last statement and asks the operating system to delete it.
  - Output data from child to parent.
  - Process' resources are deallocated by operating system (open files, physical and virtual memory, I/O buffers).

- Parent may terminate execution of children processes.
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting (e.g., OS may not allow child to continue if its parent terminates).

# Cooperating Processes

- Independent process cannot affect or be affected by the execution of another process.
- Cooperating process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information` sharing (shared files)
  - Computation speed-up (split-up of a task into subtasks and run them in parallel; Note: number of processors!)
  - Modularity (divide a system into separate processes)
  - Convenience (a user has many tasks; e.g. printing, editing, compiling)

# Producer-Consumer Problem

- Paradigm for cooperating processes, producer produces information that is consumed by a consumer.
  - Examples: printer queue, keyboard buffer
  - Consumer and Producer processes need to be synchronised
  - Buffer may be provided by OS (IPC mechanism) or be explicitly coded by the programmer
  - *unbounded-buffer* places no practical limit on the buffer size
    - Producer can always produce elements
    - Consumer gets blocked when there are no elements
  - *bounded-buffer* assumes that there is a fixed buffer size.
    - Producer is blocked when the buffer is full
    - Consumer is blocked when the buffer is empty

# Race Conditions

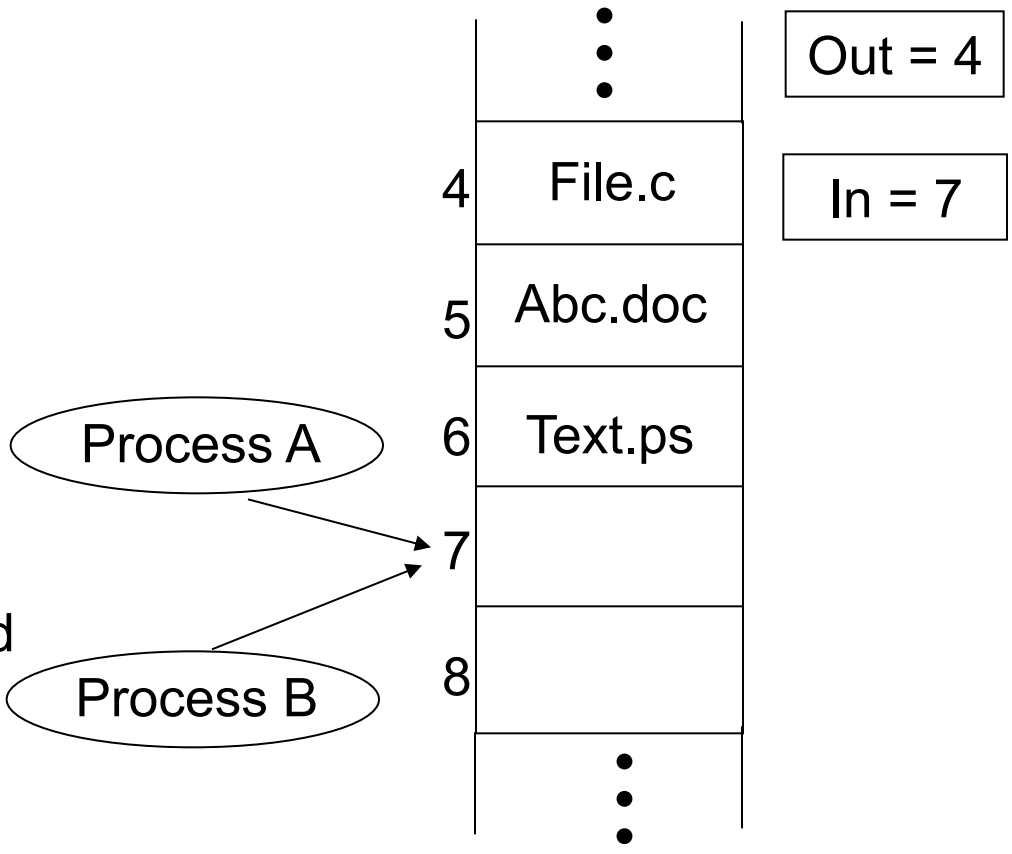
- Concurrent access to shared data may result in data inconsistency (the bank example!).
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.
- Example: Print Spooler

# Example of Race Condition

- Print Spooler
  - User processes put file names into a spooler directory
  - Printer daemon periodically checks for any new files in the spooler directory
    - If there are any – print the file and remove file name from spooler directory
  - Directory has infinite number of slots
  - Two globally available variables:
    - *out* points at the next file to be printed
    - *in* points at the next available slot
  - At a time: slots 0 – 3 are empty (files printed)
    slots 4 – 6 are full (files to be printed)

# Example cont.

- Almost simultaneously processes A and B want to print
- Process A reads *in* (value 7) and assigns it to local variable
- Process context switch occurs to process B
- Process B reads *in* (value 7) and stores a file in slot 7, updates *in* to 8
- Process A runs again, continuing from where it stopped
- Process A reads local variable and stores a file at slot 7
- Process B's file is erased!
- RACE CONDITION!



# Summary

- Process is an Operating System abstraction (program in execution)
- One program may result in many processes
- Processes may be interrupted while processing (process states)
- Processes cooperate (communicate)
- Race Conditions