03. Processes

CS 4352 Operating Systems

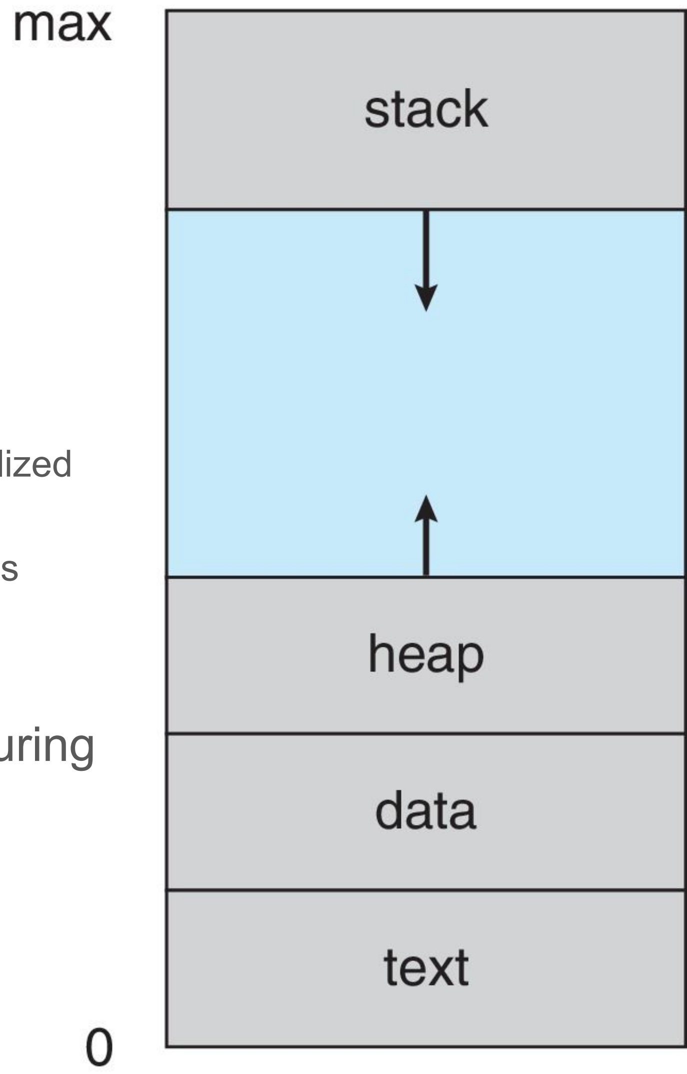
Adopted from slides of Drs. Silberschatz, Huang, & Bryant

Process Concept

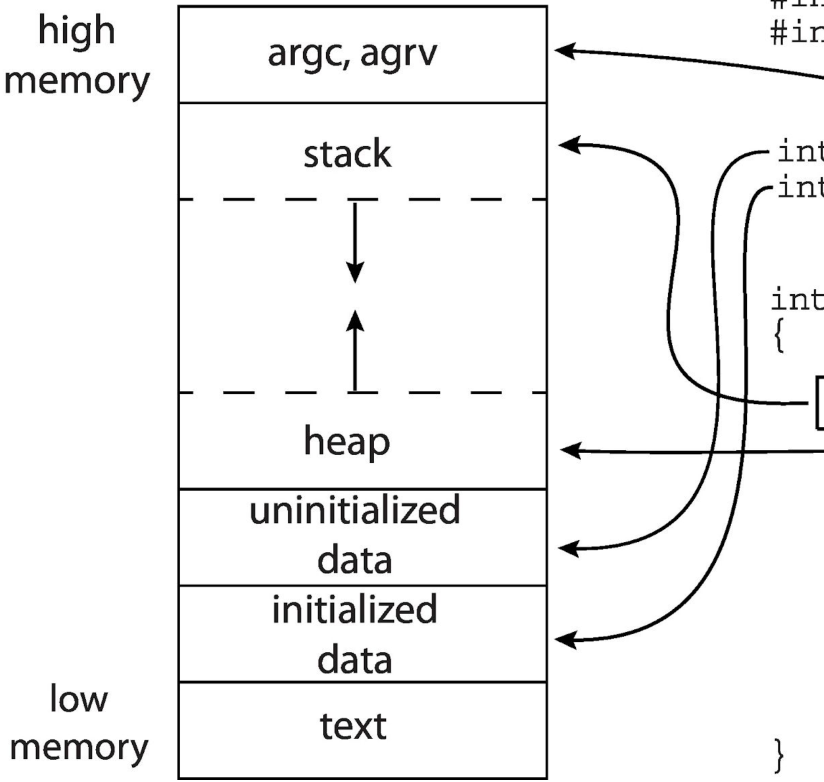
- A process is an instance of a running program
  - One of the most profound ideas in computer science
  - Sometimes also called a job or a task
  - Program is **passive** entity stored on disk (executable file); process is **active**
  - One program can be several processes
    - Consider multiple users executing the same program
  - Current state including the contents in CPU registers and memory
- Process provides each program with two key abstractions:
  - Logical control flow
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called **context switching**
  - Private address space
    - Each program seems to have exclusive use of main memory.
    - Provided by kernel mechanism called **virtual memory**

Process in Memory

- The program code, also called **text** section
  - **Program counter** contains the next instruction to execute
- **Data** section containing global variables
  - There is also a BSS (Block Started by Symbol) for uninitialized global variables
  - Better Save Space → don’t occupy space in the object files
- **Stack** containing temporary data
  - Function parameters, return addresses, local variables
- Heap containing memory dynamically allocated during run time

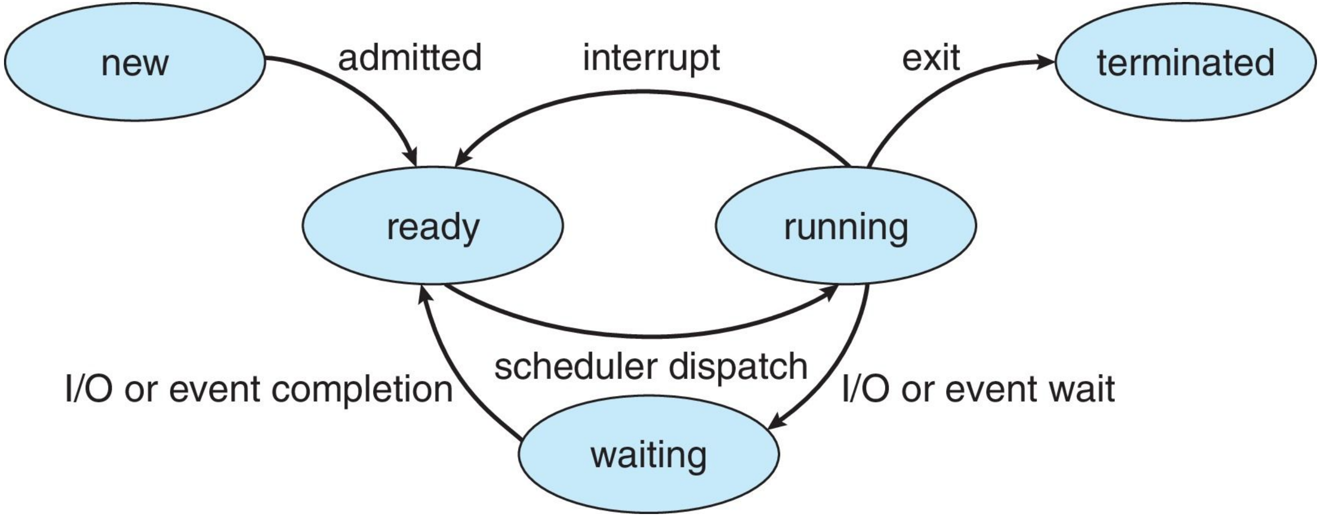


Memory Layout of a C Program




```
#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;
}
```

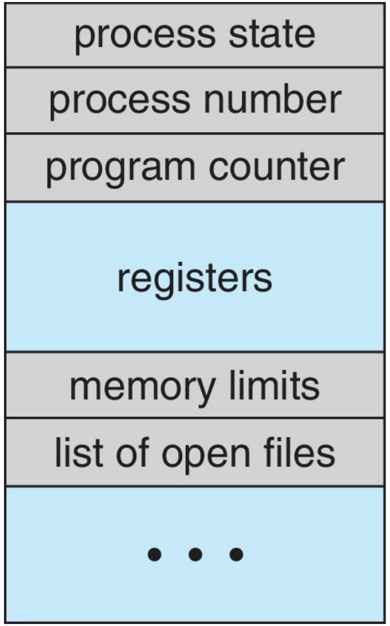
Process State



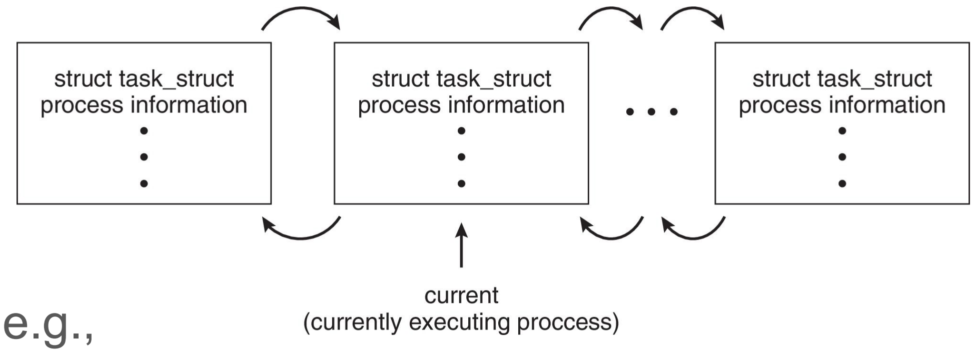
- As a process executes, it changes state
  - New: The process is being created
  - Running: Instructions are being executed
  - Waiting: The process is waiting for some event to occur
  - Ready: The process is waiting to be assigned to a processor
  - Terminated: The process has finished execution

Process 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, elapsed clock time, time limits
  - I/O status information – I/O devices allocated to process, list of open files

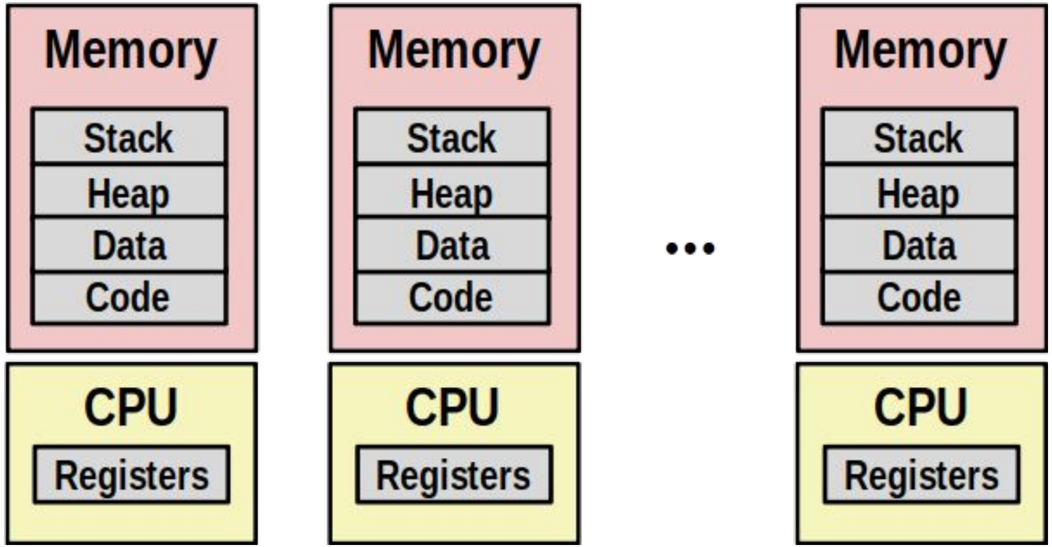


PCB in Linux



- The **struct task_struct**
    - Defined in include/linux/sched.h
- It has a large number of members, e.g.,
    - pid t_pid; /* process identifier */
    - long state; /* state of the process */
    - unsigned int time_slice; /* scheduling information */
    - struct task_struct *parent; /* this process's parent */
    - struct list_head children; /* this process's children */
    - struct files_struct *files; /* list of open files */
    - struct mm_struct *mm; /* address space of this process */

The Illusion



- Computer runs many processes simultaneously
  - Applications for one or more users
    - Web browsers, email clients, editors, …
  - Background tasks
    - Monitoring network & I/O devices

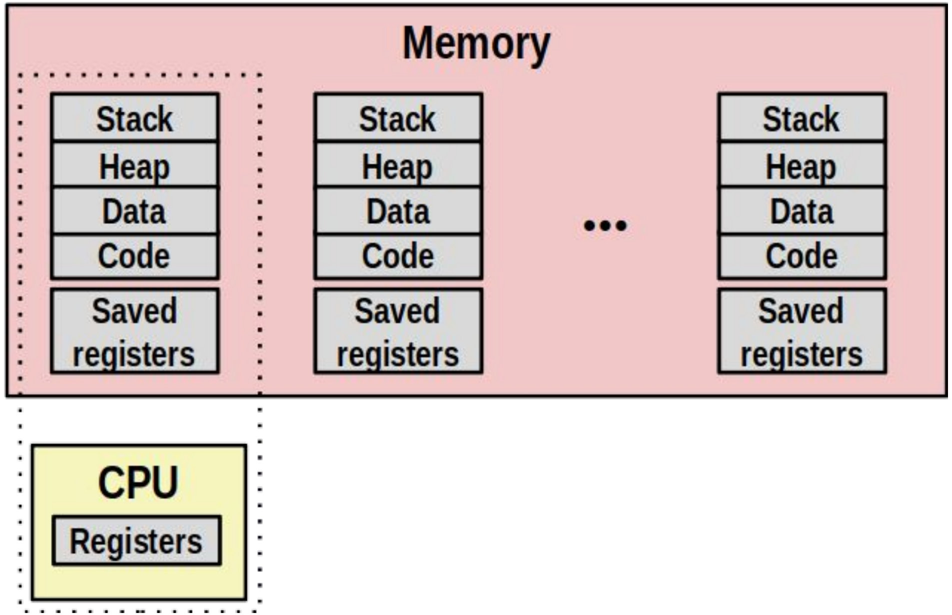
Top Command

zzk@isis: ~

File Edit View Search Terminal Help

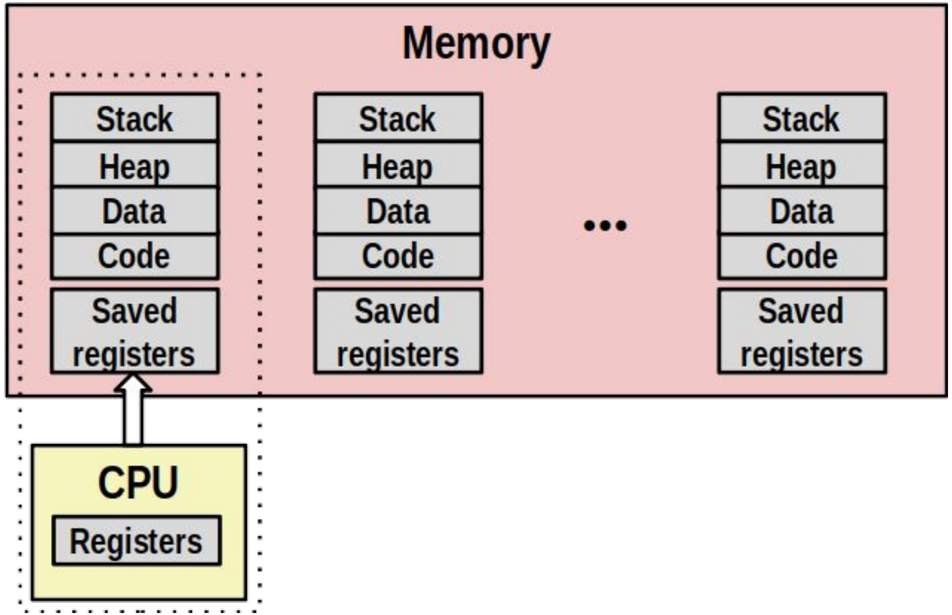
```
top - 18:43:28 up 10:36,  1 user,  load average: 0.24, 0.38, 0.22
Tasks: 343 total,   1 running, 268 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.1 sy,  0.0 ni, 99.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 32690432 total, 25633280 free,  2641388 used,  4415764 buff/cache
KiB Swap: 97654784 total, 97654784 free,        0 used. 28789092 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14655 zzk       20   0 5059244 196092 116664 S   1.7  0.6   7:39.56 chrome
11717 zzk       20   0 1225520 390404 201136 S   1.0  1.2   8:44.65 chrome
 4121 zzk       20   0 4103040 226904 119512 S   0.7  0.7   4:57.40 gnome-shell
 4355 zzk       20   0 1539264 112964  69736 S   0.7  0.3   2:54.14 fdm
27678 zzk       20   0   43820   3884   3108 R   0.7  0.0   0:00.25 top
 1776 debian-+  20   0  109612  53084  15236 S   0.3  0.2   0:28.22 tor
 2505 root      20   0 1394484  11272   9692 S   0.3  0.0   0:36.08 teamviewerd
11758 zzk       20   0  586388 110956  67548 S   0.3  0.3   3:37.56 chrome
26858 root      20   0       0      0      0 I   0.3  0.0   0:00.34 kworker/5:0
27060 zzk       20   0 9171384 252144  96644 S   0.3  0.8   1:04.02 chrome
27108 root      20   0       0      0      0 I   0.3  0.0   0:00.18 kworker/4:2
    1 root      20   0  225816   9612   6756 S   0.0  0.0   0:08.71 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.04 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:+
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_+
    7 root      20   0       0      0      0 S   0.0  0.0   0:00.09 ksoftirqd/0
    8 root      20   0       0      0      0 I   0.0  0.0   0:11.88 rcu_sched
```

The (Traditional) Reality



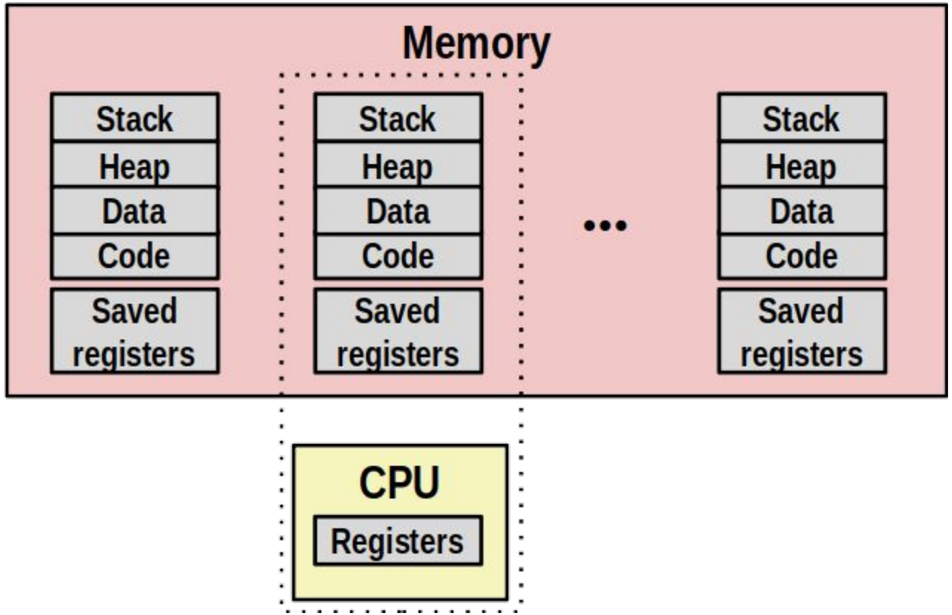
- Single processor executes multiple processes concurrently
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system (later in course)
  - Register values for non-executing processes saved in memory

The (Traditional) Reality



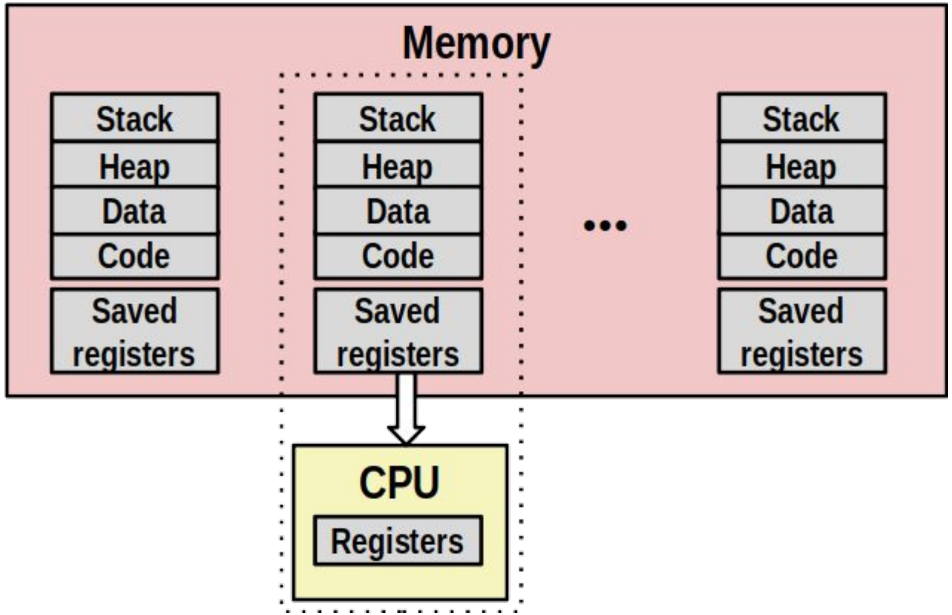
- Save current registers in memory

The (Traditional) Reality



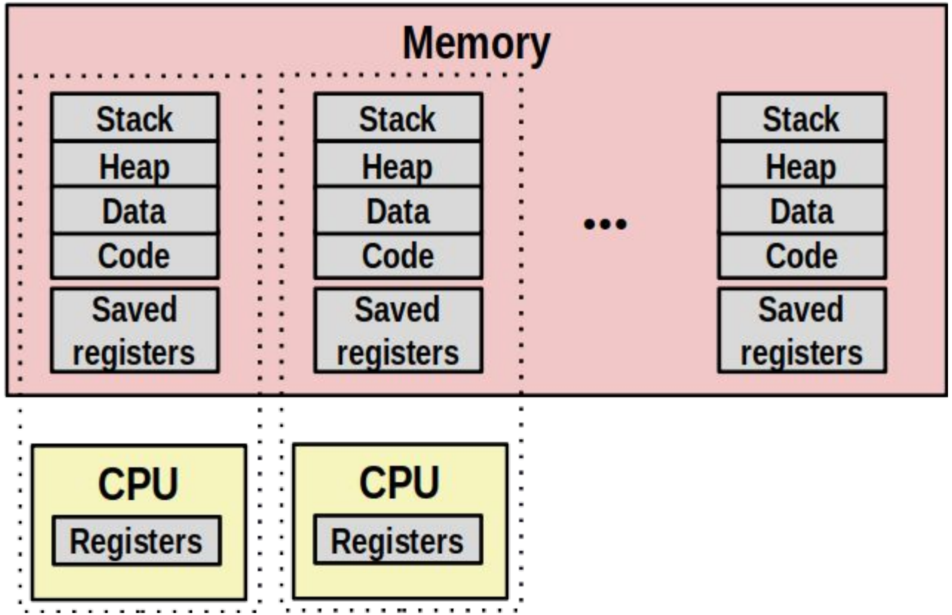
- Schedule next process for execution

The (Traditional) Reality



- Load saved registers and switch address space (context switch)

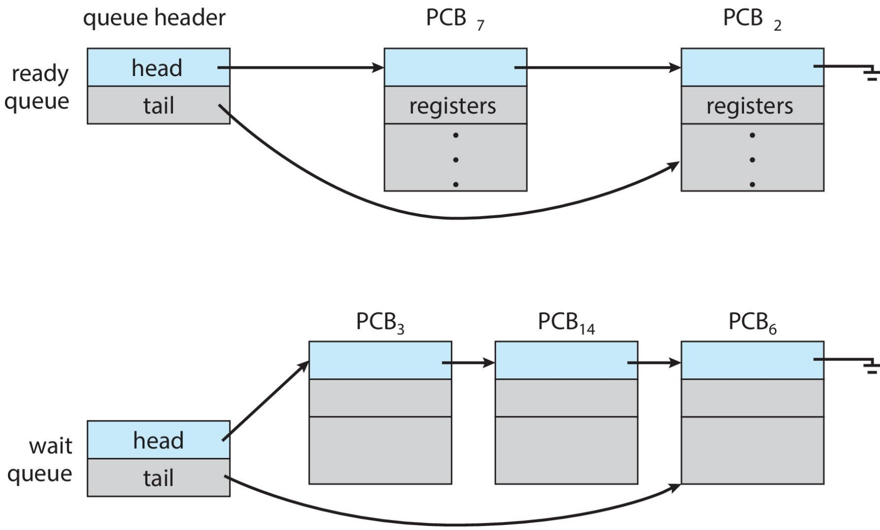
The (Modern) Reality



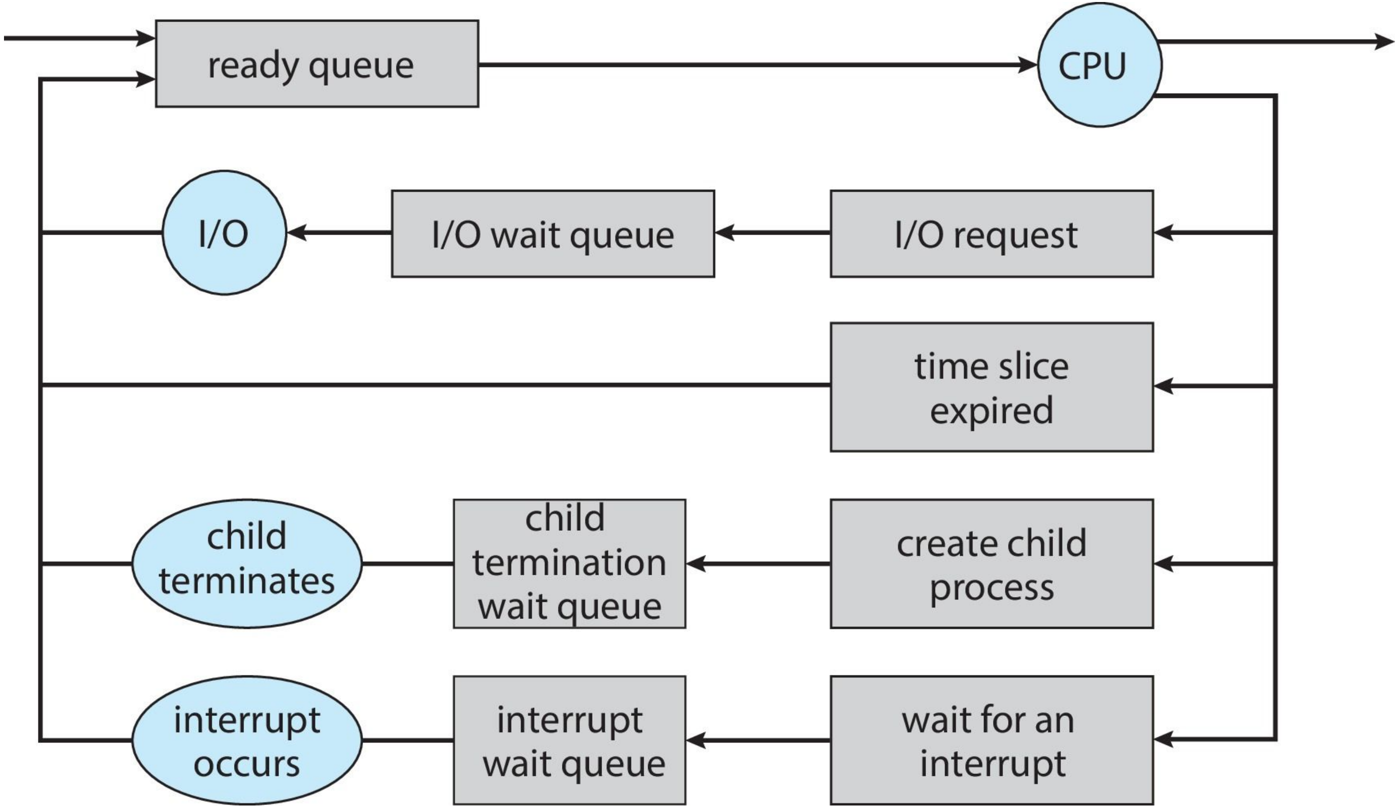
- Multicore processors
  - Multiple CPUs on single chip
  - Share main memory (and some of the caches)
  - Each can execute a separate process
    - Scheduling of processors onto cores done by kernel

Scheduler

- **Scheduler** selects among available processes for execution on CPU cores
  - Goal -- Maximize CPU use, quickly switch processes onto CPU cores
- Maintains scheduling queues of processes
  - **Ready queue** – set of all processes ready and waiting to execute
  - **Waiting queues** – set of processes waiting for an event (i.e., I/O)
  - Processes migrate among the various queues

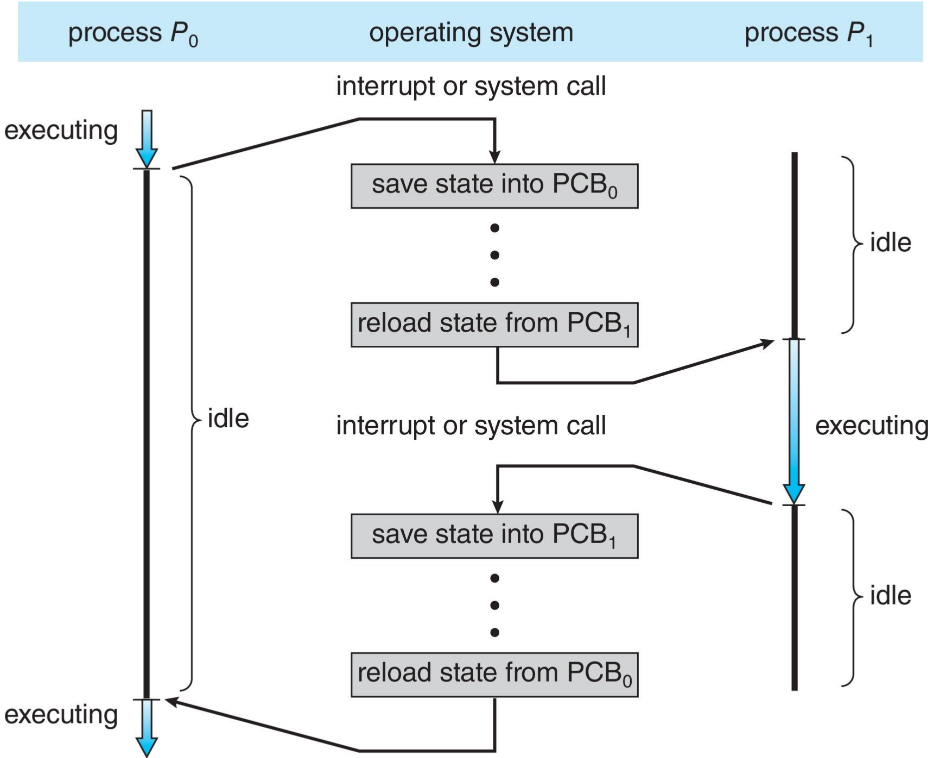


Representation of Process Scheduling



CPU Switch From Process to Process

A **context switch** occurs when the CPU switches from one process to another



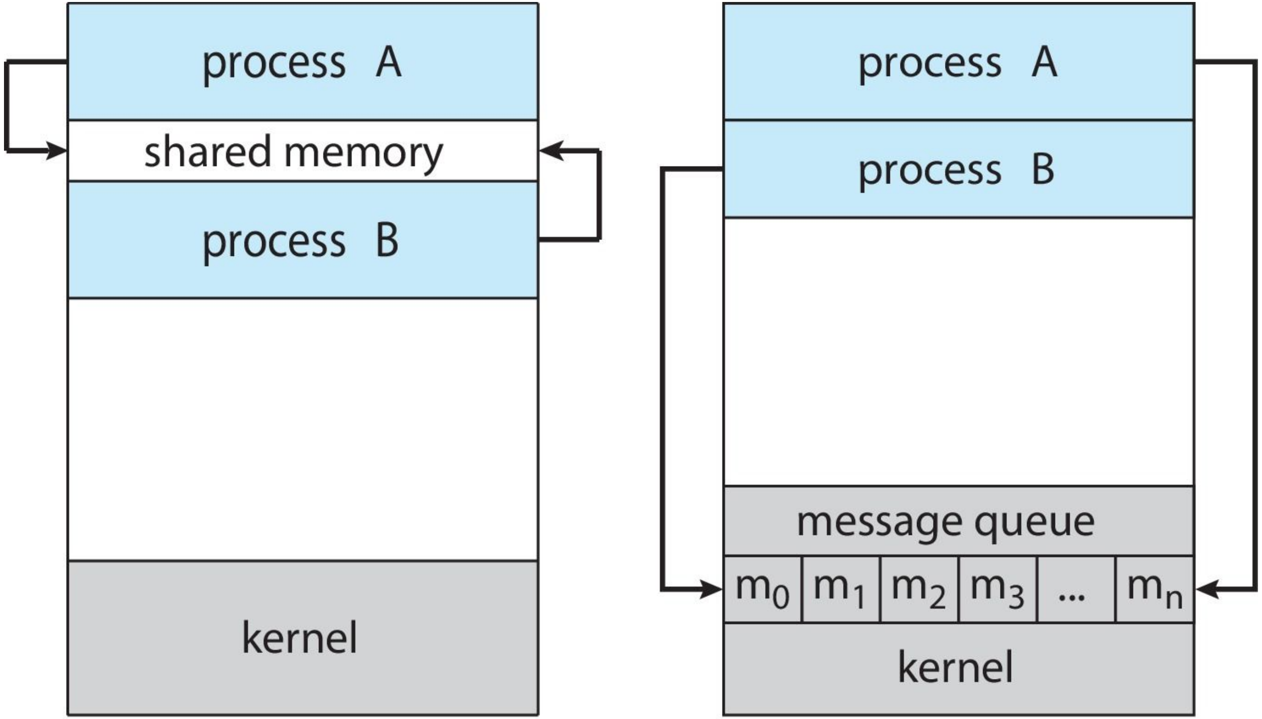
Context Switch Details

- Very machine dependent. Typical things include:
  - Save program counter and integer registers (always)
  - Save floating point or other special registers
  - Save condition codes
  - Change virtual address translations
- Non-negligible cost
  - Save/restore floating point registers expensive
    - Optimization: only save if process used floating point
  - May require flushing TLB (memory translation hardware)
    - HW Optimization 1: don’t flush kernel’s own data from TLB
    - HW Optimization 2: use tag to avoid flushing any data
- Usually causes more cache misses (switch working sets)

Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
  - Cooperating process can affect or be affected by other processes
  - Reasons for cooperating processes:
    - Information sharing
    - Computation speedup
    - Modularity
    - Convenience
- Cooperating processes need **interprocess communication** (IPC)
- Two models of IPC
  - Shared memory
  - Message passing

Communications Models



(a) (b)

Producer-Consumer Problem

- Paradigm for cooperating processes:
  - Producer process produces information that is consumed by a consumer process
- Two variations:
  - Unbounded-buffer places no practical limit on the size of the buffer:
    - Producer never waits
    - Consumer waits if there is no buffer to consume
  - Bounded-buffer assumes that there is a fixed buffer size
    - Producer must wait if all buffers are full
    - Consumer waits if there is no buffer to consume

Shared Memory

- An area of memory **shared among the processes** that wish to communicate
- The communication is **under the control of the users processes** not the operating system
- Major issues is to provide mechanism that will allow the user processes to **synchronize** their actions when they access shared memory.
  - **Synchronization** is discussed in great details later on

Message Passing

- Processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - send(message)
  - receive(message)
  - The message size is either fixed or variable
- If processes P and Q wish to communicate, they need to:
  - Establish a communication link between them
    - How are links established?
    - Can a link be associated with more than two processes?
    - How many links can there be between every pair of communicating processes?
    - What is the capacity of a link?
    - Is a link unidirectional or bi-directional?
  - Exchange messages via send/receive
    - Is the size of a message that the link can accommodate fixed or variable?

Implementation of Communication Link

- Physical:
  - Shared memory
  - Hardware bus
  - Network
- Logical:
  - Direct or indirect
  - Synchronous or asynchronous
  - Automatic or explicit buffering

Direct Communication

- Processes must name each other explicitly:
  - send(P, message) – send a message to process P
  - receive(Q, message) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes
  - Each mailbox has a unique ID
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional
- Operations
  - Create and delete a mailbox
  - Send and receive messages through mailbox
- Primitives are defined as:
  - send(A, message) – send a message to mailbox A
  - receive(A, message) – receive a message from mailbox A

Examples of IPC Systems - POSIX

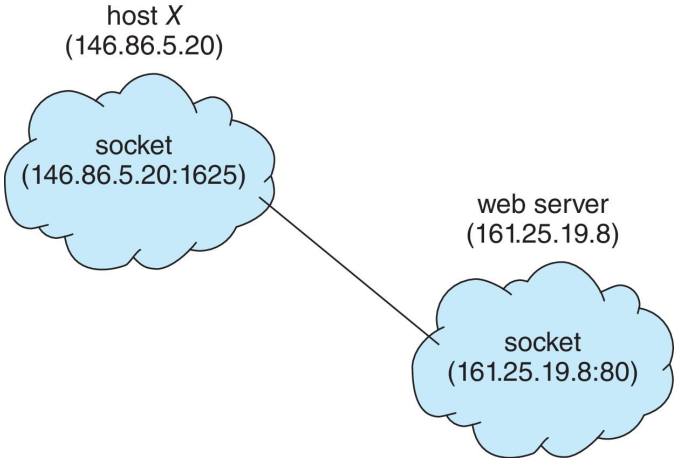
- POSIX Shared Memory
  - Process first creates shared memory segment
    - shm_fd = **shm_open**(name, O_CREAT | O_RDWR, 0666);
    - Also used to open an existing segment
  - Set the size of the object
    - **ftruncate**(shm_fd, 4096);
  - Use **mmap()** to memory-map a file pointer to the shared memory object
  - Reading and writing to shared memory is done by using the pointer returned by **mmap**()

Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls

Sockets

- A socket is defined as an endpoint for communication
- Concatenation of IP address and port
  - The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
  - Special IP address 127.0.0.1 (loopback) to refer to system on which process is running
  - All ports below 1024 are well known, used for standard services
- Communication consists between a pair of sockets

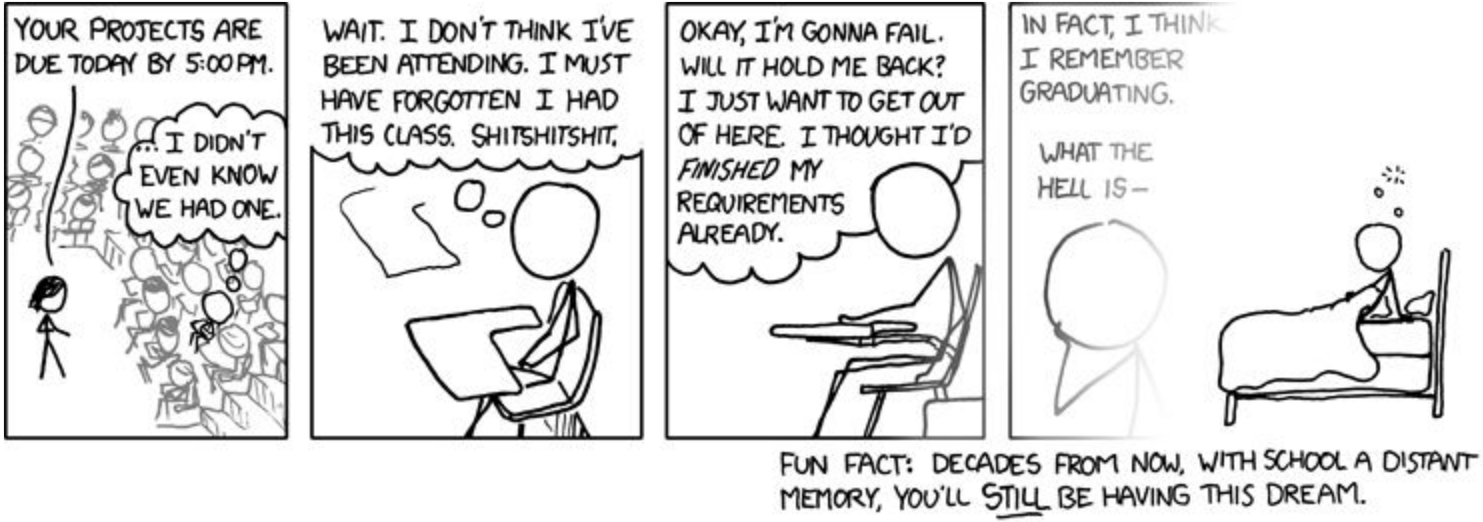


Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
  - Again uses ports for service differentiation
- **Stubs** – client-side proxy for the actual procedure on the server
  - The client-side stub locates the server and marshalls the parameters
  - The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server
- Data representation handled via External Data Representation (XDL) format to account for different architectures
  - Big-endian and little-endian
- OS typically provides a rendezvous service to connect client and server

Next Lecture

- We learn process creation and termination (assignment 1 is due tomorrow)



Credit: https://xkcd.com/557/