Operating System (OS)
CS232

Process

Dr. Muhammad Mobeen Movania

Outlines

- What is a process?
- Multitasking vs Multithreading
- Process abstraction
- Process API
- Process creation
- Process states and state transitions
- Process data structure
- Summary

What is a process?

- A running instance of a program
- Process abstraction
  - allows operating system to run many programs simultaneously like a browser and word process running at the same time
  - Makes system easy to use
- How does OS give illusion of many CPUs to allow so many programs to run at the same time?
  - OS virtualizes the CPU

Illusion of an infinite supply of CPU

- At any instant, there can be 100 or 1000 processes running at the same time on an OS
- OS creates this illusion by virtualizing the CPU
- Time Sharing of CPU
  - OS runs one process, then stops it and then runs another process (context switching) and so on

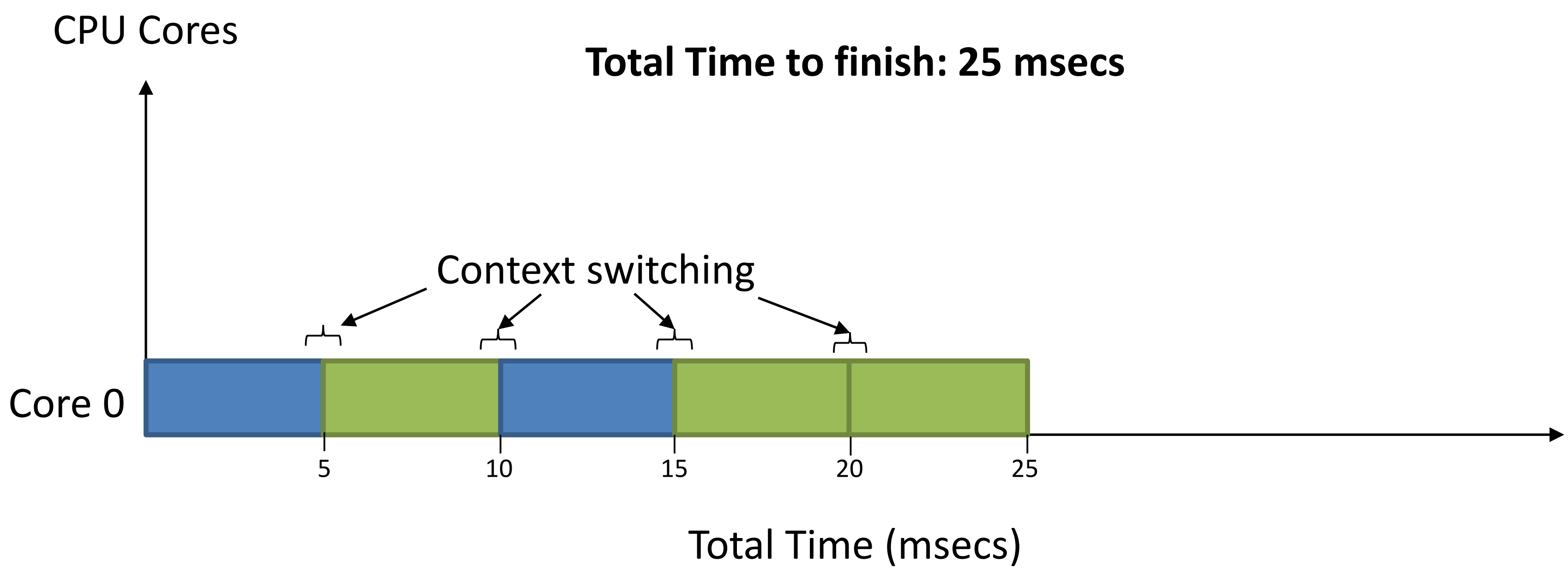
Multitasking vs Multithreading

- Multitasking is also call time slicing or time sharing of CPU
  - Idea is to rapidly switch processes to give an illusion of many processes running at the same time
  - At any time, only one process is running on the CPU
- Multithreading
  - Idea is to run several concurrent processes at the same time
  - CPU must have more than one core to truly appreciate the benefits
  - At any time, more than one processes are running on the CPU

Single Threaded Multitasking (Single Core CPU)

Num of CPU Cores : 1
Time slice: 5 msecs

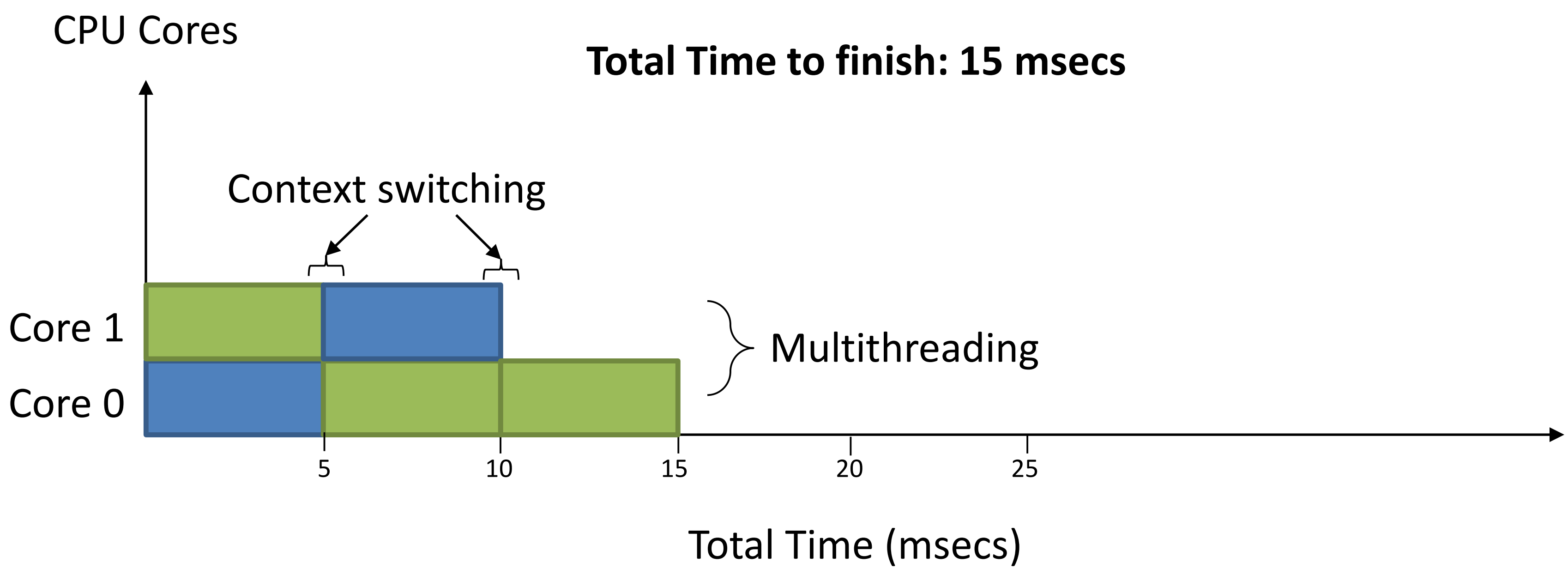
Total Time to finish: 25 msecs



Multithreaded Multitasking (Dual Core CPU)

Num of CPU Cores : 2
Time slice: 5 msecs

Total Time to finish: 15 msecs



Process Abstraction

- Has to store
  - process or machine state (values of CPU registers, PC/IP, stack pointer (SP), frame pointer(FP) etc.)
  - memory the process can address (address space)
  - I/O information (list of open files)

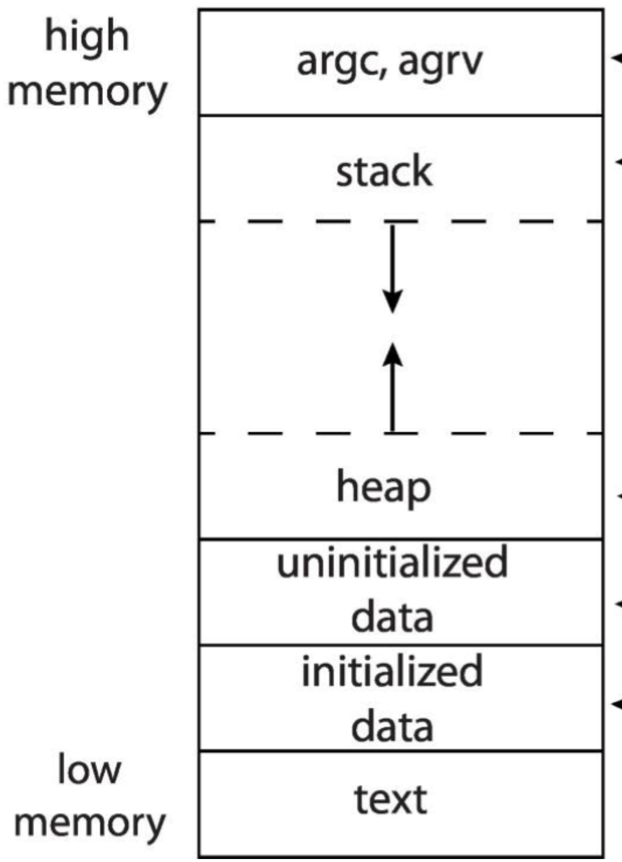
Process API

- OS must provide API to
  - Create
  - Destroy
  - Wait
  - Misc. Control (suspension of a running process)
  - Status (time and state of process)

Process Creation

- What happens when you double click an exe?
  - OS loads code and static data into the address space of process in RAM
  - Loading can be eager (all at once) or lazy (on demand)
  - OS allocates some memory for process's run-time stack (for storing function local variables and return addresses, command line args) and some for heap (dynamic memory allocations)
  - OS also initializes I/O related interfaces
  - OS finally transfers control to the process main function

Process Creation

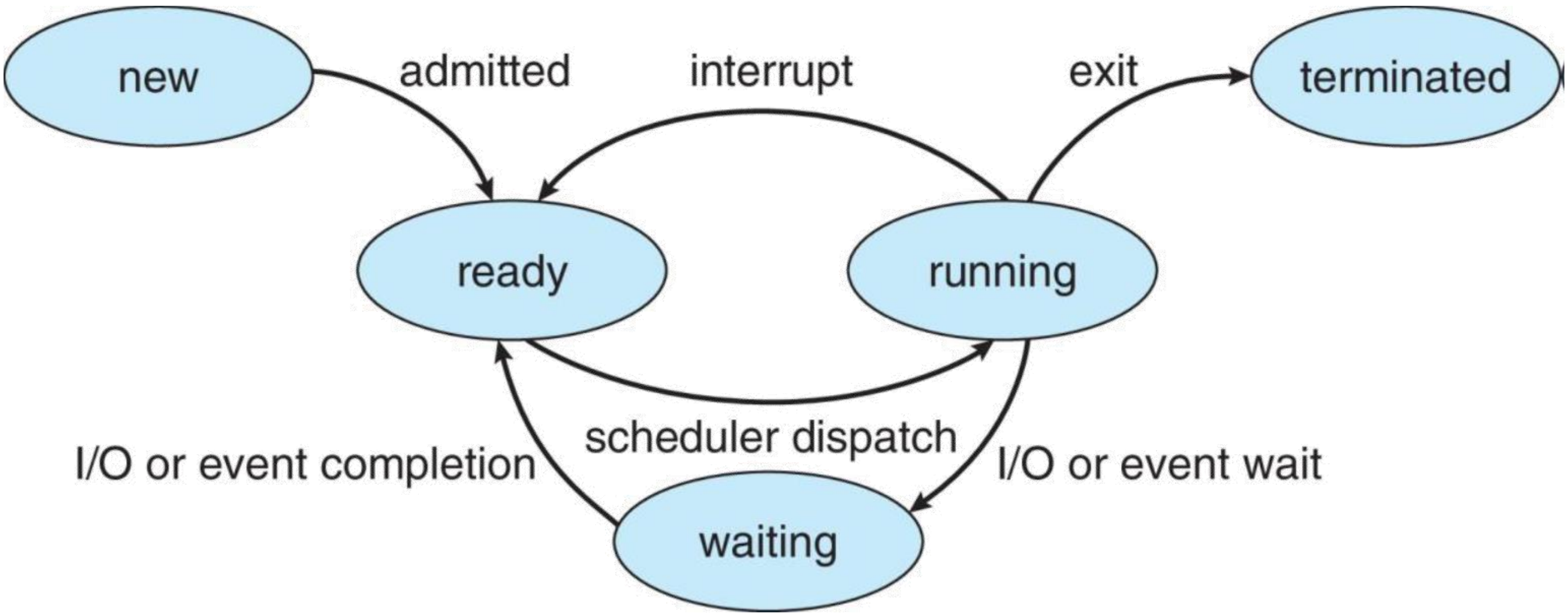



```
#include <stdio.h>
#include <stdlib.h>

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 States

- A process can be in one of the five states
  - New (Process is created)
  - Running (Process is executing instructions)
  - Ready (Process is ready but OS has not chosen it)
  - Waiting/Blocked (Process has performed some operation like I/O request so it cannot continue until that operation finishes)
  - Terminated (Process has finished execution)

Process State Transition



Examples of Process Transitions without I/O

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	$Process_0$ now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	$Process_1$ now done

Figure 4.3: Tracing Process State: CPU Only

Examples of Process Transitions with I/O

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	$Process_0$ initiates I/O
4	Blocked	Running	$Process_0$ is blocked,
5	Blocked	Running	so $Process_1$ runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	$Process_1$ now done
9	Running	–	
10	Running	–	$Process_0$ now done

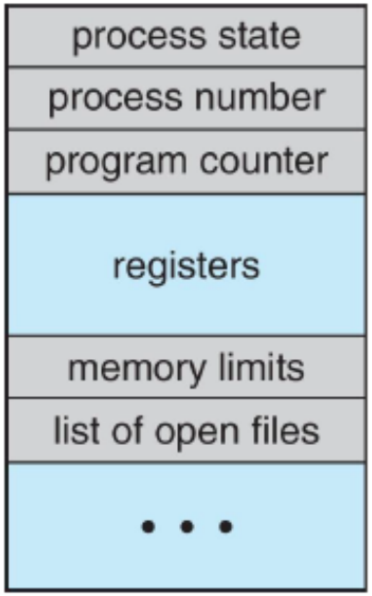
Figure 4.4: Tracing Process State: CPU and I/O

Process Data Structure

- Process list contains information of all processes in system
- Each entry in the process list is a 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 - Implementation

- Register context holds contents of registers for the stopped process so that execution can continue when the process is scheduled again

```
struct context {
   int eip;   //insn ptr
   int esp;   //stack ptr
   int eax;   //gp, originally accumulator
   int ebx;   //gp
   int ecx;   //gp, originally counter
   int edx;   //gp
   int esi;   //gp, for string ops is source index
   int edi;   //gp, for string ops is dest. index
   int ebp;   // base ptr
 }; //segment registers: CS, DS, SS, ES
```

PCB – Implementation (2)

```
// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };

// the information xv6 tracks about each process
// including its register context and state
struct proc {
  char *mem;                  // Start of process memory
  uint sz;                    // Size of process memory
  char *kstack;               // Bottom of kernel stack
                              // for this process
  enum proc_state state;      // Process state
  int pid;                    // Process ID
  struct proc *parent;        // Parent process
  void *chan;                 // If !zero, sleeping on chan
  int killed;                 // If !zero, has been killed
  struct file *ofile[NOFILE]; // Open files
  struct inode *cwd;          // Current directory
  struct context context;     // Switch here to run process
  struct trapframe *tf;       // Trap frame for the
                              // current interrupt
};
```

Summary

- Process is a major **OS abstraction** of a **running program**
- A **process list** contains information about all processes in the system. Each entry is found in what is sometimes called a **process control block** (**PCB**)
- Process state is stored in **PCB** which includes
  - Contents of memory in process address space
  - Values of CPU registers (PC/IP,SP,FP)
  - I/O information (open files list)
- Process goes through **5 states** during its lifetime and transitions among them due to events (scheduled or descheduled, or waiting for I/O)