Operating Systems and Networks

Lecture 05:

Introduction to OS-part 4

Behzad Bordbar

Recap

- Different types of system call
  - process control
  - file manipulation
  - device manipulation
  - information maintenance
  - communications
  - protection.
- process
- process =/= program (and =/= thread)

Contents

From C to Assembly

- http://gcc.godbolt.org/
- What does this piece of C code do?

```
#include <stdio.h>
 #define LAST 10
   int main()
   {
       int i, sum = 0;
       for ( i = 1; i <= LAST; i++ ) {
         sum += i;
       }
       printf("sum = %d\n", sum);
       return 0;
   }
```

From C to Assembly (continue)

- ❑Explore assembly output with different compilers (remove – O2 from Compiler options and activate Colourise)
- ❑process a program which is running.
- ❑What do we need for that?
- ❑observe: push, pop, mov…

stack and heap

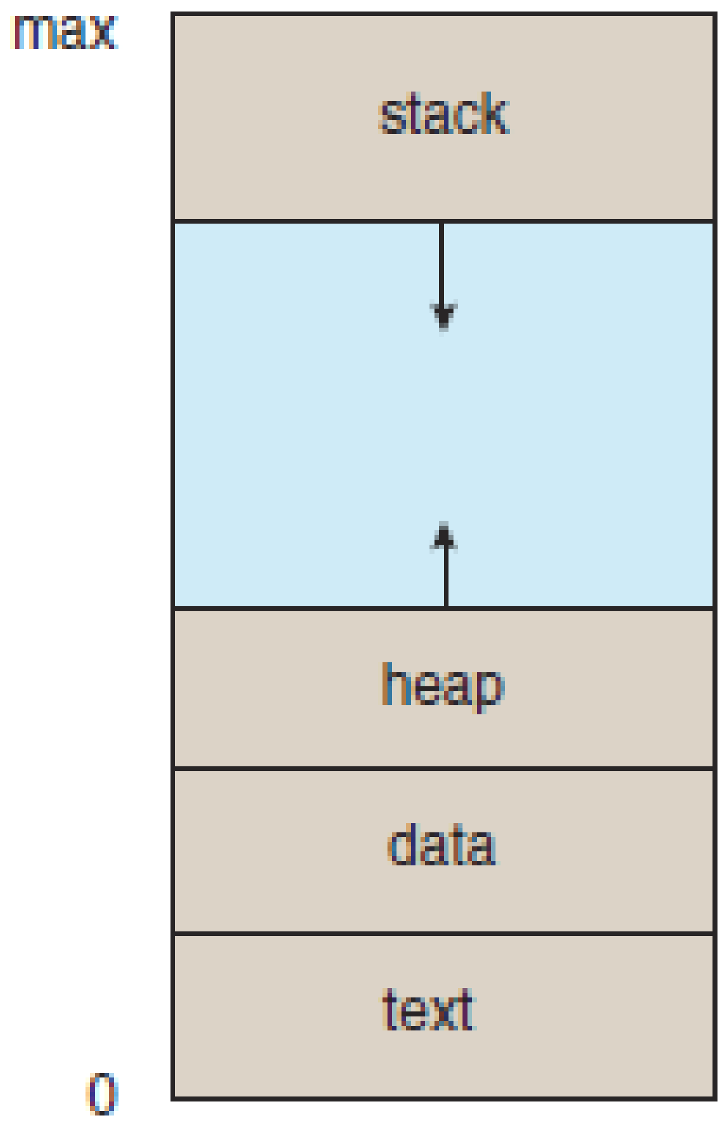
- Stack is a contiguous block of memory that is allocated to each process.
- Stack is LIFO (last in first out).
- Stack has stack frame: contains parameter to functions, local variables and data for recovering previous stack frame.
- When a function is called frame for that function is pushed into stack, when done executing we pop. When we call stack grow, when we execute stack shrinks and we end up with the caller and then frame pops.

heap and stack

- ❑stack frames

picture from dragonbook

What is difference between heap and stack?



stack

- in java we have a data type jave.util.stack (not here!)
- part of computer memory that stores temporary variables created by each function (including the main() function). T
- FILO data structure managed and optimized by the CPU
- push and pop of stack frames
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

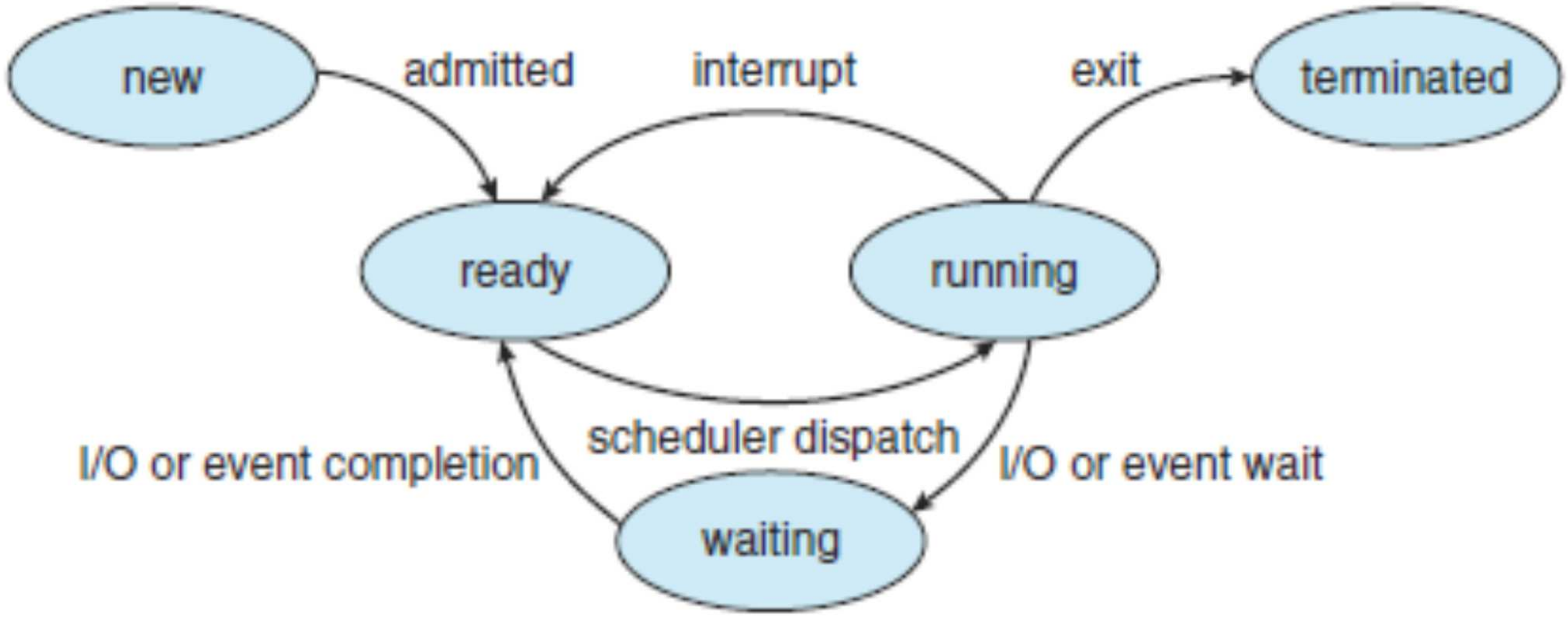
stack

- advantage: that memory is managed for you [no need to allocate memory yourself or free it once you don't need it any more]
- CPU organizes stack memory so efficiently (very fast).
- stack is limited
- $pgrep chrome
- $ cat /proc/<PID>/limits

heap

- part of computer memory that is not managed automatically.
- in C you must use malloc() or calloc() to allocate
- you are responsible for using free() to de-allocate that memory
- if you don’t: memory leak (memory on the heap will still be set aside (and won't be available to other processes)
- Unlike the stack, the heap does not have size restrictions on variable size (apart from the physical limitations of your machine).
- Heap memory is slightly slower to be read from and written to, because one has to use pointers to access memory on the heap.
- Heap variables are essentially global in scope

process state



picture from dragonbook

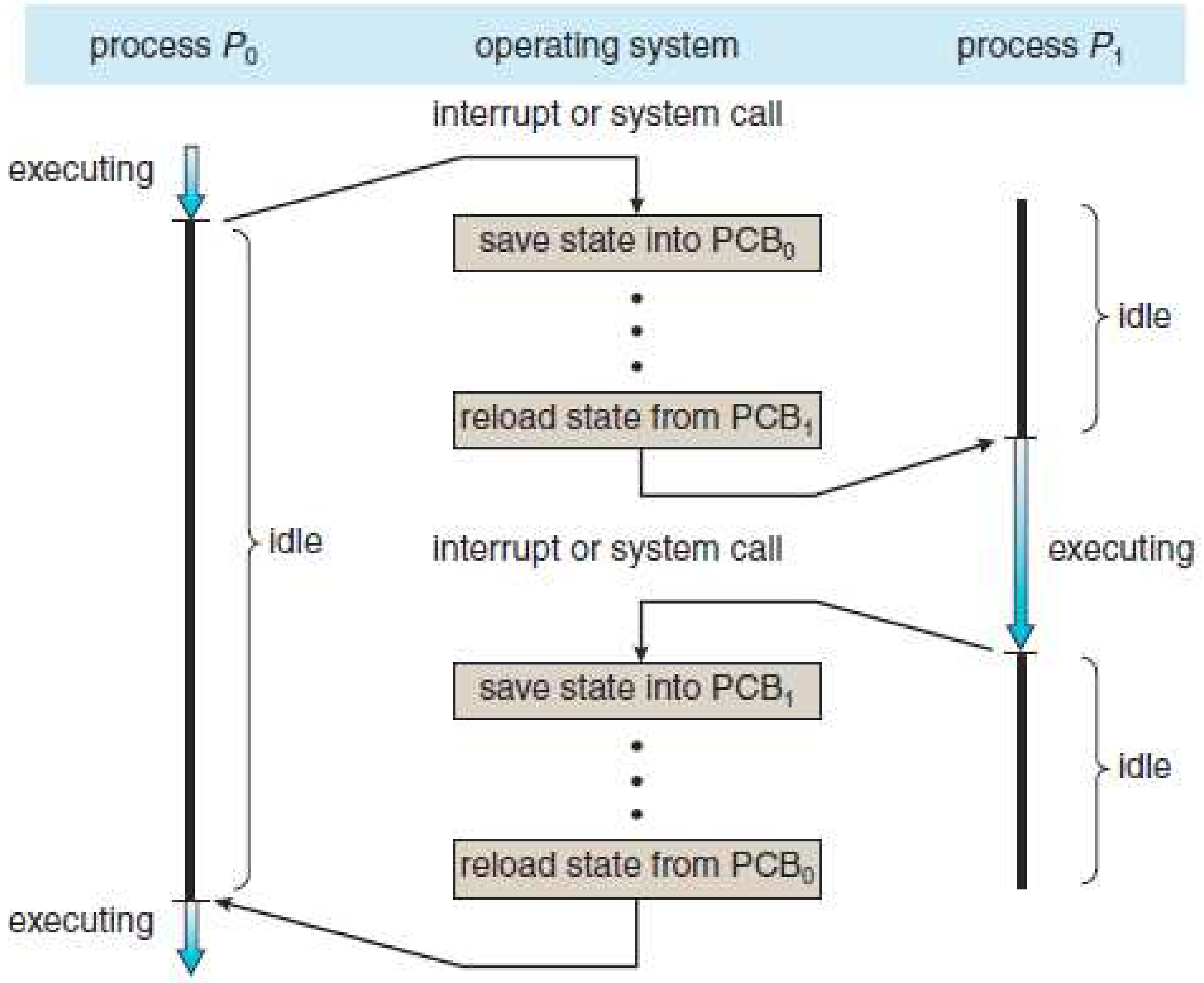
- only one process can be running on any processor at any instant.
- Many processes may be ready and waiting,

process control block (PCB)

information required for context switching

- ❑Process state
- ❑ Program counter.
- ❑CPU registers
- ❑ CPU-scheduling information
- ❑ Memory-management information
- ❑Accounting information
- ❑ I/O status information

How does CPU execute multiple processes?



dragonbook. so… how do process and program compare?

Multiprogramming (revisited)

- ❑Maximising CPU utilisation: some process running at all time
- ❑time sharing: is to switch the CPU among processes so frequently that users can interact with each program while it is running
- ❑ Only one process can run at a time, the rest of processes must wait

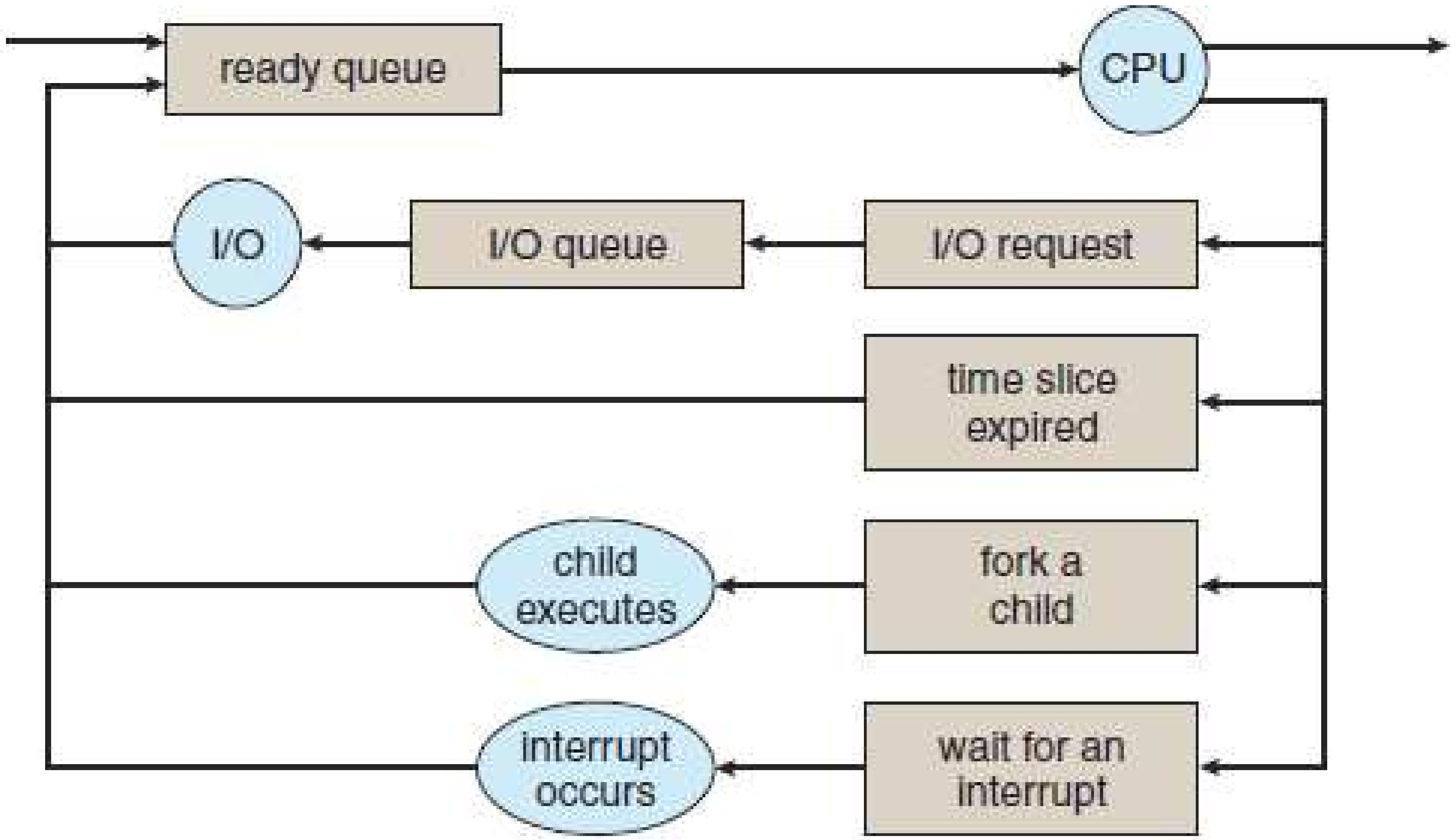
Process Scheduler:

- ❑ selects an available process for program execution on the CPU.

Queues

- job queue: all processes in the system
- ready queue: processes that are residing in main memory and are ready and waiting to execute are kept on a list (linked list)
- PCBs that have a link to the next PCB
- Device queue: list of processes waiting for a particular I/O device is called a

Queuing Diagram



source Dragon book

Context switching

- interrupt >> a state save of the current state of the CPU and then a state restore to resume operations
- kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context-switch time is pure overhead.
- Switching speed varies from machine to

machine (few milliseconds)

Process and Thread

- ❑McDonald on high street: thread and process
- ❑Example: a document with 1) editor 2) spell checker
- ❑fork in unix (two process)
- ❑can we share the context… thread
- ❑Suitable particularly for multicore

Process in Linux: called task

- ❑processes form a tree $ps -el
- ❑unique id number (pid integer)
- ❑The init process (pid of 1) is the root parent and spwans other proceses
- ❑kthreadd: creating additional processes that perform tasks on behalf of the kernel

- ❑ps -ef |more
- ❑ ps -efH |more
- ❑ pstree

Process in Linux

- fork() system call: makes new process
- New process has a copy of the address of the original process. Why?
- better communication between father & child
- father continues executing and child calls exec() sys. call and replaces memory space with new prog.
- Draw picture + example of shell and editor
- both proc. can create more children or
- father invokes wait() sys. call to move to ready queue until the termination of the child (background proc. &)
- termination, cascading termination, zombie process

Process in Linux

- ❑termination: A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit() system call.

Then what happens:

- ❑return status value (typically an integer) to its parent process (via the wait() system call)
- ❑All the resources of the process—including physical and virtual memory, open files, and I/O buffers—are deallocated by the operating system.

A process can be terminated directly by other process

- ❑cascading termination: if parents terminate all children terminated (show via shell and background process)

Process in Linux (continue)

- ❑What is exit status?
- ❑wait() system call by parents
- ❑a pointer is passed to return exit status of child
- ❑zombie process
- ❑orphan process
- ❑terminating of orphan process by init()

summary

- ❑Heap vs stack (what size is a proc stack?)
- ❑proc states and control block
- ❑context switching
- ❑process and thread
- ❑process in linux