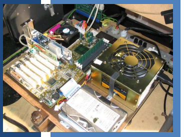
TCSS 422: OPERATING SYSTEMS



The Abstraction: The Process

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

OBJECTIVES

- Process API
- Process states
- Process data structures

CPU VIRTUALIZING

- How should the CPU be shared?
- Time Sharing:
  Run one process, pause it, run another
- How do we SWAP processes in and out of the CPU efficiently?
  - Goal is to minimize **overhead** of the swap

PROCESS

A process is a running program.

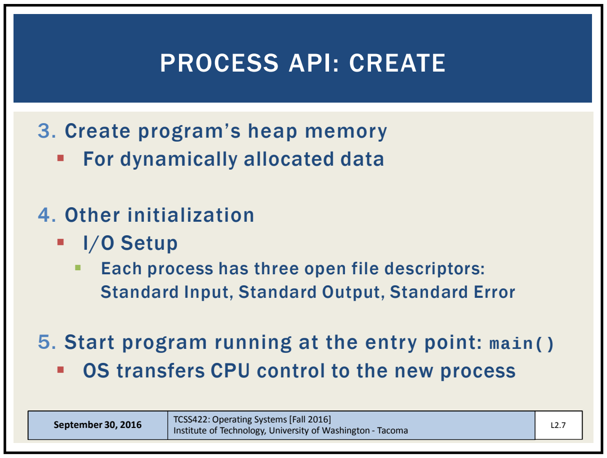
- Process comprises of:
  - Memory
    - Instructions ("the code")
    - Data (heap)
  - Registers
    - PC: Program counter
    - Stack pointer

PROCESS API

- Modern OSes provide a Process API for process support
- Create
  - Create a new process
- Destroy
  - Terminate a process (ctrl-c)
- Wait
  - Wait for a process to complete/stop
- Miscellaneous Control
  - Suspend process (ctrl-z)
  - Resume process (fg, bg)
- Status
  - Obtain process statistics: (top)

PROCESS API: CREATE

1. Load program code (and static data) into memory
   - Program executable code (binary): loaded from disk
   - Static data: also loaded/created in address space
   - Eager loading: Load entire program before running
   - Lazy loading: Only load what is immediately needed
     - Modern OSes: Supports paging & swapping
2. Run-time stack creation
   - Stack: local variables, function params, return address(es)

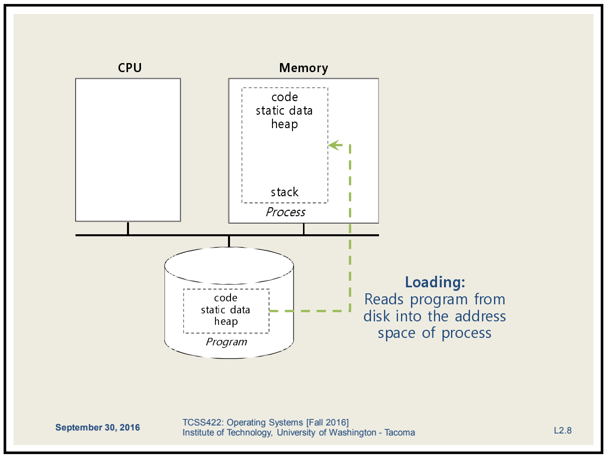


PROCESS API: CREATE

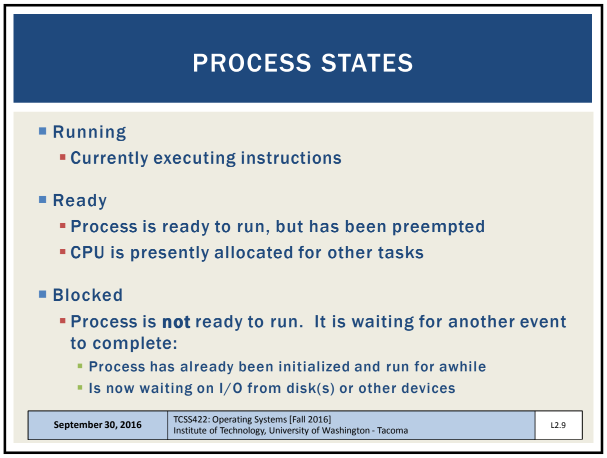
3. Create program's heap memory
   - For dynamically allocated data

4. Other initialization
   - I/O Setup
     - Each process has three open file descriptors: Standard Input, Standard Output, Standard Error

5. Start program running at the entry point: `main()`
   - OS transfers CPU control to the new process

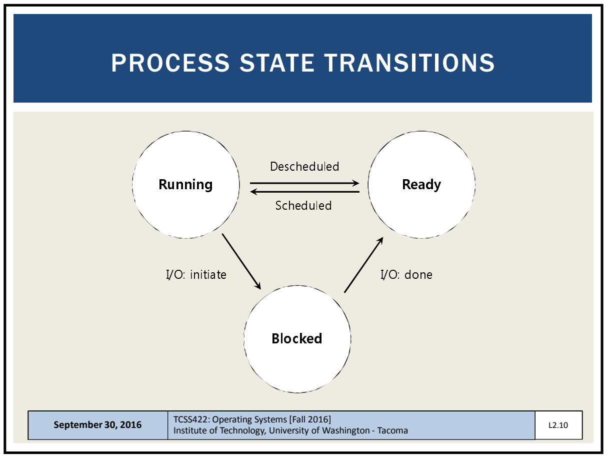


Loading:
Reads program from disk into the address space of process

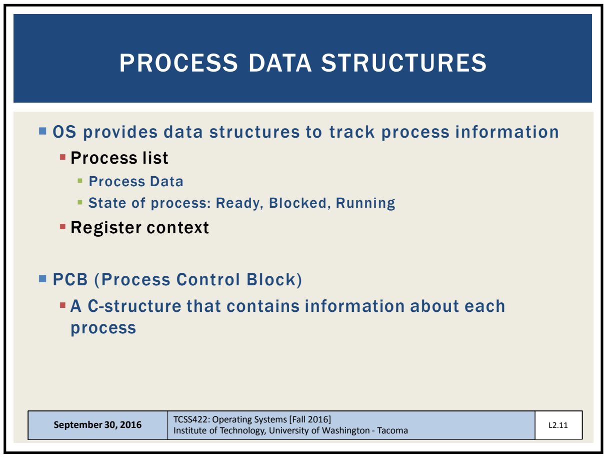


PROCESS STATES

- Running
  - Currently executing instructions
- Ready
  - Process is ready to run, but has been preempted
  - CPU is presently allocated for other tasks
- Blocked
  - Process is **not** ready to run. It is waiting for another event to complete:
    - Process has already been initialized and run for awhile
    - Is now waiting on I/O from disk(s) or other devices

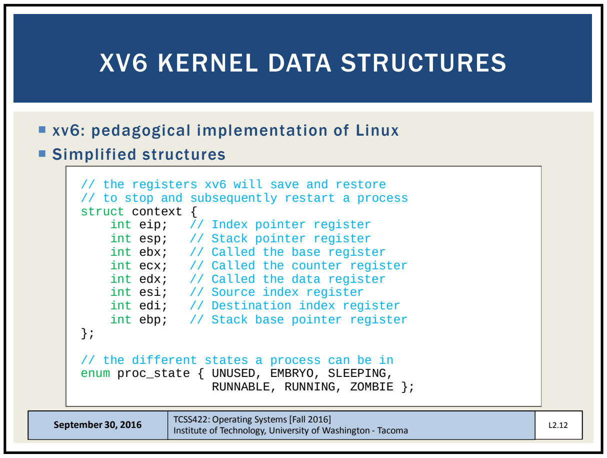


PROCESS STATE TRANSITIONS



PROCESS DATA STRUCTURES

- OS provides data structures to track process information
  - Process list
    - Process Data
    - State of process: Ready, Blocked, Running
  - Register context
- PCB (Process Control Block)
  - A C-structure that contains information about each process



XV6 KERNEL DATA STRUCTURES

- xv6: pedagogical implementation of Linux
- Simplified structures

```
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
    int eip;   // Index pointer register
    int esp;   // Stack pointer register
    int ebx;   // Called the base register
    int ecx;   // Called the counter register
    int edx;   // Called the data register
    int esi;   // Source index register
    int edi;   // Destination index register
    int ebp;   // Stack base pointer register
};

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

XV6 KERNEL DATA STRUCTURES - 2

```
// 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 non-zero, sleeping on chan
    int killed;               // If non-zero, have 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
};
```

LINUX: STRUCTURES

- struct task_struct, equivanelnt to struct proc
  - Provides process description
  - Large: 10,000+ bytes
  - /usr/src/linux-headers-{kernel version}/include/linux/sched.h
    - 1227 – 1587
- Struct thread_info, provides "context"
  - thread_info.h is at:
    /usr/src/linux-headers-{kernel version}/arch/x86/include/asm/

LINUX: THREAD_INFO

```
struct thread_info {
        struct task_struct      *task;          /* main task structure */
        struct exec_domain      *exec_domain;   /* execution domain */
        __u32                   flags;          /* low level flags */
        __u32                   status;         /* thread synchronous flags */
        __u32                   cpu;            /* current CPU */
        int                     preempt_count;  /* 0 => preemptable,
                                                   <0 => BUG */
        mm_segment_t            addr_limit;
        struct restart_block    restart_block;
        void __user             *sysenter_return;
#ifdef CONFIG_X86_32
        unsigned long           previous_esp;   /* ESP of the previous stack in
                                                   case of nested (IRQ) stacks
                                                */
        __u8                    supervisor_stack[0];
#endif
        int                     uaccess_err;
};
```

LINUX STRUCTURES - 2

- List of Linux data structures:
  http://www.tldp.org/LDP/tlk/ds/ds.html
- Description of process data structures:
  http://www.makelinux.net/books/lkd2/ch03lev1sec1
  2nd edition is online (dated from 2005):
  Linux Kernel Development, 2nd edition
  Robert Love
  Sams Publishing

QUESTIONS