Operating System Overview and Processes

Outline

- Overview of what operating systems do
- What a process is
- How a process is managed

What an Operating Systems Does

- I/O Functions
- Process functions
- Memory functions
- Secondary storage functions
- User interface functions
- Other desirable features

I/O Functions

- Read, write data
  - Polling
  - Interrupts, traps
- Byte-oriented devices
- Direct memory access (DMA)

Process Functions

- Create, delete process
- Process status: get information about . . .
  - Resources used
  - Time used
  - UID/owner of process
  - GID/group of process
- Process control
  - Limit resources, increase resources
  - Control access to files
  - Handle interrupts and traps

Memory Functions

- Allocate, deallocate memory
- Share memory among processes
- Translate virtual addresses into physical addresses
- Manage pages, segments, and variants
- Map files into memory
- Protext parts of memory from being read, written, or executed

Secondary Storage Functions

- Manage space on devices
- Map file addresses into secondary storage addresses and *vice versa*
- Scheduling reads, writes to secondary storage
- Manage writing directly into, from main memory locations

User Interface Functions

- Enable users to run processes easily
- Enable users to manage files easily
- Allow easy configuration and control of system by administrators

Other Desirable Features

- Efficiency
- Reliability
- Maintainability
- No larger than necessary

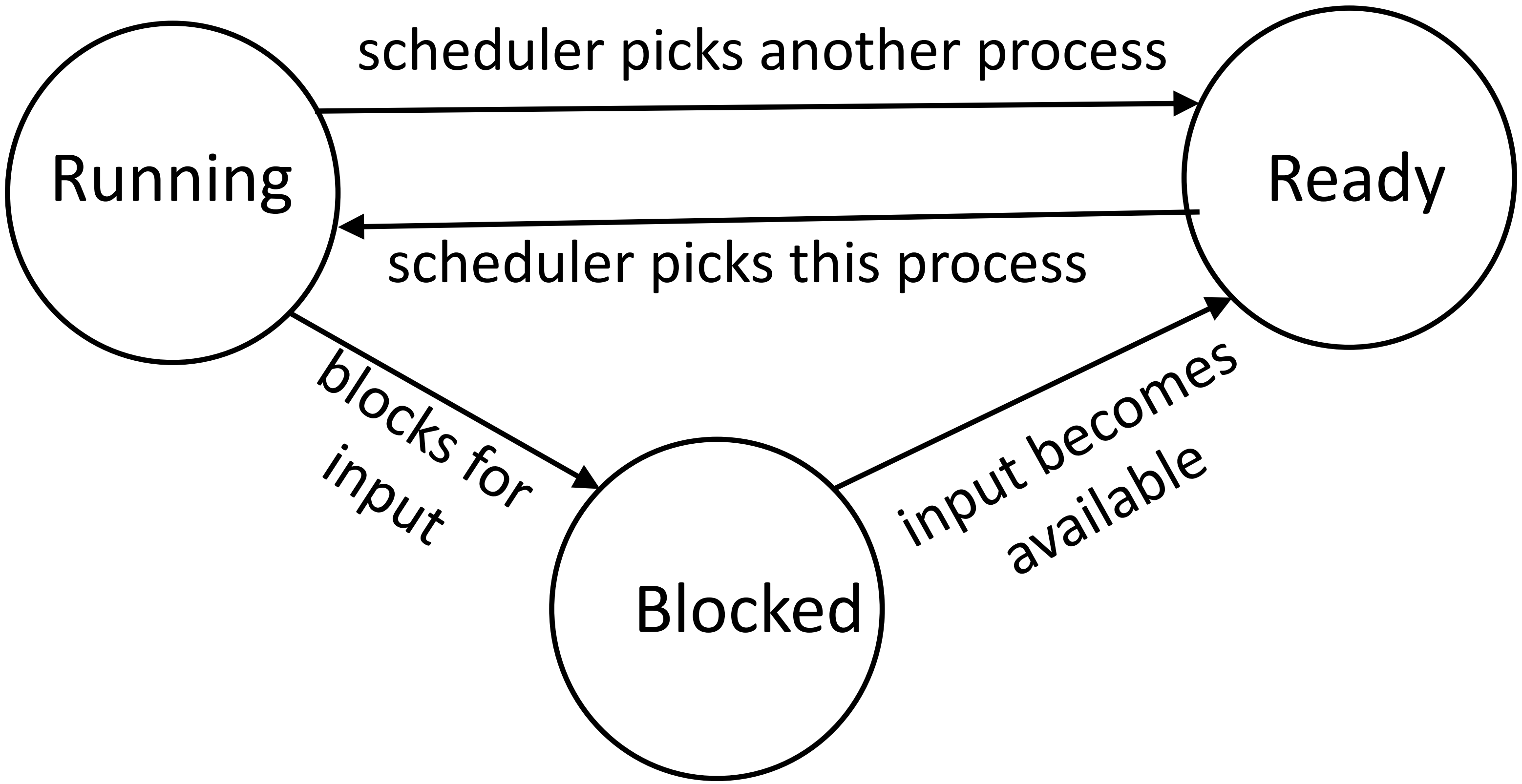
Process

- Obvious definition: A running program
- But more complicated when on a multiprogrammed system
  - As process execution is interleaved, the process does not execute continuously
  - CPU is virtualized, so each process thinks it is running continuously
  - Processes are scheduled by the job/process scheduler
- In this sense, process is an abstraction

Process State

- Running: process is executing
  - Part of it lies in memory
  - The memory that the process can address is called the process' *address space*
  - Special registers:
    - PC, program counter, gives address of next instruction to be executed
    - SP, stack pointer, points to the word beyond the stack top
    - Frame pointer used to manage the stack for function arguments and (local) variables
- Blocked: process is not running but is waiting on some event
- Ready: process ready to run but the operating system has chosen not to run it for some reason

Process State Diagram



Examples

time	process A	process B	notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	A done
5		Running	
6		Running	
7		Running	
8		Running	B done

time	process A	process B	notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	A initiates I/O
4	Blocked	Running	A blocks, B runs
5	Blocked	Running	
6	Blocked	Running	
7	Ready	Running	A's I/O done
8	Ready	Running	B now done
9	Running		
10	Running		

Process APIs

- Create: initiate a new process
  - When command entered at shell, shell creates (spawns) new process of that program executing
- Destroy: terminate a process
  - Use this to terminate (kill) an existing process
- Wait: block for some event
  - Useful when the process has to stop for some event
- Status: get status information about a process
- Other controls: varied ways to interact with the process
  - Most systems allow a user to suspend, resume a process

Details of Process Creation

- Operating system loads code, any initialized data into memory
  - This will become the process, so it is assigned an address space
  - Only parts of the program may be loaded at this time; the rest will be moved into memory when needed, and moved out when no longer needed
    - Paging, segmentation, swapping
- Operating system allocates memory for stack
  - On Linux, program arguments and environmental variables are put onto the stack
  - Stack can grow or shrink
- Operating system allocates space for heap
  - Uninitialized memory, accessed by the process allocating itself memory
  - More heap space can be allocated if needed

Details of Process Creation

- Operating system does other initializations
  - Opens input, output files and assigns them to the process
    - On Linux, this is done by assigning file descriptors
- Operating system then marks the process as ready to run, putting it in the READY state
  - It may execute it immediately, changing the state to RUNNING

Data Structures for Process Management

- Process information kept in *process table*
  - May be fixed size or be able to grow
- Process table entry contains information about a process
  - Often split into 2 parts, one remaining always in memory, the other part (*not* part of h process table) able to be moved out of memory

Process Table Entry Example

- Example is from UNIX V6
  - See the handout “Process Information in UNIX V6”
- Modern systems may have more complex entries, but the idea is the same

Process Execution

- Direct execution
  - Kernel runs process on CPU, without allowing any other process to run
- Limited direct execution
  - Kernel runs process, but on trap or interrupt, kernel takes control and allows another process to run
  - Raises issue of switching between process

Direct Execution

operating system	program/process
create entry for process in process table	
allocate memory for program	
load program into memory	
setup stack (including arguments, environment, etc.)	
clear registers	
call start of program (eg., call **main()**)	
	run **main()**
	execute **return** from **main()**
free memory of process	
delete process table entry	

Problems

- How does operating system prevent program from doing something we don’t want, wile impacting performance minimally (if at all)?
- How does the operating system stop a program and switch to another process?
    - In other words, how do we do time-sharing and multiprogramming?

Limited Direct Execution

- Make some instructions privileged
  - These are instructions enabling interference with, or directly interfering with, other processes or system management functions
  - This way, the process carries out its need but does not have control of the complete system
- To do this, introduce modes or levels of privilege
  - Kernel mode: privileged instructions can only be executed in this mode
  - User mode: normal, processes run in this mode but cannot execute privileged instructions

System Calls

- Cause a trap
- When a trap or interrupt (of any kind) occurs:
  - Do a context switch to kernel
  - Service the trap or the interrupt
  - Select the next process to run
  - Do a context switch to that process
- So at boot time, the trap table/interrupt vector must be initialized

Context Switch

- PC, processor status word, registers, pushed onto a small kernel stack allocated for the process
- Jump to routine in kernel indicated by trap table/interrupt vector
- Kernel services the trap/interrupt
- Kernel selects the next process to run
  - It may be a different one
- Kernel pops the information from the kernel stack and restores them to the registers, processor status word, and PC
- PC popped last, as when it is restored, process restarts

Context Switch Example

- Example is from XINU on an LSI-11 system
  - See the handout “Context Switch Routine for XINU System on LSI-11”
  - LSI-11 has same instruction set as PDP-11
- Modern systems may have more complex entries, but the idea is the same

Limited Direct Execution

operating system at boot (kernel mode)	hardware	program/process (user mode)
initialize interrupt/trap vectors		
	remember address of syscall handler	
create entry for process in process table		
allocate memory for program		
load program into memory		
setup stack (including arguments, environment, etc.)		
fill kernel stack with registers, PC		
return-from-trap		
	restore registers from kernel stack	

Limited Direct Execution

operating system at boot (kernel mode)	hardware	program/process (user mode)
	change to user mode	
	jump to **main()**	
		run **main()**
		system call causes trap into OS
	save registers to kernel stack	
	change to kernel mode	
	jump to trap handler	
handle trap, ie system call		
return-from-trap		
	restore registers from kernel stack	

Limited Direct Execution

operating system at boot (kernel mode)	hardware	program/process (user mode)
	change to user mode	
	jump to PC after trap	
		return from **main()**
		causes trap
free memory of process		
delete process table entry		